Mari Carmen Suárez-Figueroa
Asunción Gómez-Pérez
Enrico Motta
Aldo Gangemi *Editors*

Ontology Engineering in a Networked World

Springer

Ontology Engineering in a Networked World

Mari Carmen Suárez-Figueroa
Asunción Gómez-Pérez • Enrico Motta
Aldo Gangemi
Editors

Ontology Engineering in a Networked World

Springer

Editors

Mari Carmen Suárez-Figueroa
Universidad Politécnica de Madrid
Facultad de Informática
Ontology Engineering Group
Campus de Montegancedo sn.
Boadilla del Monte
Madrid
Spain
mcsuarez@fi.upm.es

Asunción Gómez-Pérez
Universidad Politécnica de Madrid
Facultad de Informática
Ontology Engineering Group
Campus de Montegancedo sn.
Boadilla del Monte
Madrid
Spain
asun@fi.upm.es

Enrico Motta
The Open University
Knowledge Media Institute
Milton Keynes
United Kingdom
e.motta@open.ac.uk

Aldo Gangemi
Semantic Technology Lab,
Institute for Cognitive Sciences
and Technology, CNR
Via Nomentana 56
Rome
Italy
aldo.gangemi@cnr.it

ISBN 978-3-642-24793-4 e-ISBN 978-3-642-24794-1
DOI 10.1007/978-3-642-24794-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011945419

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To our families

Foreword

I am very pleased that this book has seen the light. I have been involved in the NeOn project from its early beginning: first, during the proposal writing and project negotiation phase, then as a project member responsible for one of the use cases and the exploitation of the results and, finally, when I moved on to a different professional challenge, as a member of the Advisory Board. Whatever the collaboration form, it has always been a pleasure to work with this excellent consortium.

Even after my leave from the NeOn project, I remained active in semantic research, but then from a technology transfer point of view. During the past 5 years, I have monitored the impact of European research projects on the economic competitiveness of Europe. Results show that although there is much investment in EU research, tangible results – at least in ICT – in terms of economic impact are more the exception than the rule. It is therefore with great pleasure that I can write the foreword to this NeOn book, more than a year after the end of the project, which is a clear manifestation that NeOn has turned into an initiative beyond its initial funding of the European Commission, a necessary step towards economic impact.

The first time I heard the term 'ontology' was in the early 1990s, when – in computer science – it was merely an academic concept. Now, 20 years later, the term ontology and more general, semantic technology, is penetrating increasingly more applications and areas, spearheaded by areas such as life sciences. This book contributes to putting ontological engineering in a more realistic environment: out of the labs and into the real world (wide web), where reuse and interrelationships are more the rule than exceptions.

Of course, there is still a long way to go for ontologies and semantic technology to be fully taken up by mainstream markets, but this book certainly will help to speed up the process.

Dr. V. Richard Benjamins
Director User Modelling
Telefónica Digital

Preface

The Semantic Web is characterized by the existence of a very large number of distributed semantic resources, which subscribe to alternative but often overlapping modelling schema (i.e. ontologies). Together these resources define a *network of ontologies*. This emerging scenario is radically different from the relatively narrow contexts in which ontologies have been traditionally developed and applied. Thus, there is a need for new practical methodologies and technologies to support effectively the development of a new kind of network-oriented semantic applications. This new support should assist a variety of users, dealing with a variety of ontology engineering tasks.

To address this methodological need, this book describes the *NeOn Methodology Framework*, which includes a set of nine scenarios for collaboratively building ontologies and ontology networks, a glossary of processes and activities potentially involved in ontology development and a collection of ontology life cycle models. Other important aspects of this framework include (a) a pattern-based design approach and (b) the provision of various models which can be used to represent information about ontology networks. In addition, the framework provides a set of methodological guidelines for the different processes and activities relevant to the development of networked ontologies. These guidelines are presented in a prescriptive way to facilitate their adoption by students and practitioners. The guidelines are supported by a comprehensive software environment, which provides effective and integrated support for all the processes and activities described in the book. Hence, the book also includes (a) an overview of the NeOn Toolkit, focusing in particular on the user interaction side, and (b) a detailed description of several plugins, which are most critical to the ontology development process.

Finally, the book shows how the NeOn methods and tools have been applied in three real-world case studies in the fishery and pharmaceutical domains. These descriptions reveal effectively the value of the proposed methods and tools.

This book aims to be a self-contained compendium of material for students and practitioners in ontology engineering. We aim to provide the necessary level of detail to allow readers to adopt the proposed methods and tools in practical ontology engineering projects. This book can be used as a textbook for undergraduate and postgraduate courses on ontology engineering, together with other books which focus specifically on the use of OWL for ontology engineering.

The content presented in this book is the result of the work done in the NeOn project (life cycle support for networked ontologies), which was funded by the European Commission's Sixth Framework Programme under grant number FP6-027595. Several dozens people collaborated on the NeOn project, and the research described in this book would have not been possible without such massive collaborative effort. Hence, we would like to thank all the people who collaborated in the project for the excellent contribution to advancing research in ontology engineering and for making this book possible.

In addition, we are extremely thankful to our colleagues, Nathalie Aussenac-Gilles, Vadim Ermolayev, Mouna Kamel, Pierluigi Miraglia, Sofia Pinto, Elena Simperl, Vojtech Svátek and Valentina Tamma, who provided very interesting comments and feedback and helped unselfishly to improve the quality of this book.

We are also grateful to Ralf Gerstner, Frank Holzwarth, Viktoria Meyer and Tanja Jäger at Springer-Verlag for their support and assistance during the production of the manuscript.

Finally, we are very thankful for the love and support from our families without which we could not have finished this book.

Mari Carmen Suárez-Figueroa
Asunción Gómez-Pérez
Enrico Motta
Aldo Gangemi

Contents

Part I NeOn Methodology Framework

Part II Ontology Engineering Activities

Part III The NeOn Toolkit

Part IV Case Studies

Chapter 1
Introduction: Ontology Engineering in a Networked World

Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Enrico Motta, and Aldo Gangemi

Abstract While ontology engineering is rapidly entering the mainstream, expert ontology engineers are a scarce resource. Hence, there is a need for practical methodologies and technologies, which can assist a variety of user types with ontology development tasks. To address this need, this book presents a scenario-based methodology, the NeOn Methodology, which provides guidance for all main activities in ontology engineering. The context in which we consider these activities is that of a networked world, where reuse of existing resources is commonplace, ontologies are developed collaboratively, and managing relationships between ontologies becomes an essential aspect of the ontological engineering process. The description of both the methodology and the ontology engineering activities is grounded in a comprehensive software environment, the NeOn Toolkit and its plugins, which provides integrated support for all the activities described in the book. Here we provide an introduction for the whole book, while the rest of the content is organized into 4 parts: (1) the NeOn Methodology Framework, (2) the set of ontology engineering activities, (3) the NeOn Toolkit and plugins, and (4) three use cases. Primary goals of this book are (a) to disseminate the results from the NeOn project in a structured and comprehensive form, (b) to make it easier for students and practitioners to adopt ontology engineering methods and tools, and

M.C. Suárez-Figueroa (✉) • A. Gómez-Pérez
Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid,
Campus de Montegancedo sn., 28660 Boadilla del Monte, Madrid, Spain
e-mail: mcsuarez@fi.upm.es; asun@fi.upm.es

E. Motta
Knowledge Media Institute (KMi), The Open University, Walton Hall, Milton Keynes
MK7 6AA, UK
e-mail: e.motta@open.ac.uk

A. Gangemi
Semantic Technologies Lab, Institute of Cognitive Sciences and Technologies (National Research
Council – CNR), Via Nomentana 56, 00161 Rome, Italy
e-mail: aldo.gangemi@cnr.it

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_1, © Springer-Verlag Berlin Heidelberg 2012

(c) to provide a textbook for undergraduate and postgraduate courses on ontology engineering.

1.1 Introduction

The Semantic Web is characterized by the existence of a very large number of distributed semantic resources, which subscribe to alternative but often overlapping modeling schema (i.e., ontologies). Together these resources define a *network of ontologies* related through a variety of different meta-relationships such as versioning, inclusion, inconsistency, similarity, and others. This emerging scenario is radically different from the relatively narrow contexts in which ontologies have been traditionally developed and applied, and calls for new methods and tools to support effectively the development of a new kind of network-oriented semantic applications.

Hence, ontologies on the Web are not stand-alone artifacts. They relate to each other in ways that might affect their meaning, and are inherently distributed in a network of interlinked semantic resources. More precisely, a *network of ontologies* or an *ontology network* is a collection of ontologies related together via a variety of relationships, such as alignment, modularization, version, and dependency. Accordingly, a *networked ontology* is an ontology included in such a network, sharing relationships with a potentially large number of other ontologies.

Intuitively, this aspect of considering ontologies as included in a network implies that they are defined not only through their content but also in terms of ontology metadata, which provide information about their provenance, purpose, and the relations with other ontologies and semantic resources, among other things.

One of the most common ways for two ontologies to relate is to be dependent on each other. More precisely, it is often the case that in order to define its own model, an ontology refers to the definitions included in another ontology. The OWL language includes a primitive (owl:imports) allowing an ontology developer to declare such a relationship, merging the definitions of the imported ontology with those from the importing one.

Aligning ontologies is a way to put different models in correspondence by declaring which entities in one ontology are the same as those in another ontology, or a generalization or specialization. The main purpose of alignments is to ensure semantic interoperability, making it possible to merge ontologies in a meaningful way by representing information in one ontology in terms of the entities in another.

Large, monolithic ontologies are hard to manipulate, use, and maintain. Modular ontologies on the contrary divide the ontological model in self-contained, interlinked components, which can be considered independently, while at the same time participate to the definition of a specific aspect of an ontology. Therefore, modules share the relation that they are common components of a larger ontology, and often include dependencies and alignments to other modules.

Finally, versioning relates to the activity of keeping track of the different versions of an ontology. This is of particular importance in a collaborative ontology engineering environment, where the ontology evolution process needs to be carefully monitored and managed. The OWL language includes primitives to declare versioning relations between ontologies, but these do not consider fine-grained changes and are not often used in practice.

In this networked world, ontology practitioners need both methodological and technological support for the development and use of ontology networks. We aim to provide such a support in this book.

1.2 NeOn Methodology Framework

One of the main contributions of this book is the NeOn Methodology framework, which is described in the first part of this book. Although methodological approaches already exist in the literature – e.g., METHONTOLOGY, On-To-Knowledge, and DILIGENT, they do not provide the comprehensive set of methods described in the NeOn Methodology, especially with respect to key activities in a network-centric scenario, such as those related to reusing and managing the dynamics of ontologies.

The *NeOn Methodology* (Chap. 2) uses a scenario-based approach to ontology development and provides a comprehensive set of methods and guidelines for carrying out the variety of activities required when developing ontologies in a networked world.

The NeOn Methodology includes (1) a set of nine scenarios that involve different activities for collaboratively building ontologies and ontology networks, (2) a glossary of processes and activities relevant to ontology development, (3) a collection of ontology life cycle models, and (4) a set of methodological guidelines. The NeOn Methodology defines each process or activity in a precise manner, stating its purpose, inputs and outputs, the actors involved, when its execution is more appropriate, and a set of proposed methods, techniques, and tools to be used. The methodology is presented in a prescriptive way to facilitate its adoption by students and practitioners.

Current methodologies for ontology engineering, such as METHONTOLOGY, On-To-Knowledge, and DILIGENT, mainly include guidelines for single ontology construction, ranging from ontology requirements specification to ontology implementation, and they are mainly targeted to ontology researchers. In contrast to the aforementioned approaches, the NeOn Methodology does not prescribe a rigid workflow but instead suggests pathways and activities for a variety of scenarios. The nine scenarios described in the book cover commonly occurring situations, e.g., when existing ontologies need to be reengineered, aligned, modularised, localized to support different languages and cultures, or integrated with non-ontological resources (NORs), such as folksonomies or thesauri.

Another important aspect of the NeOn Methodology is the pattern-based design approach described in Chap. 3. In this chapter, different types of ontology design patterns (ODPs) are presented as well as an associated method (named eXtreme Design) to assist in ontology development. Ontology design patterns provide modeling solutions which can be applied to solve recurrent ontology design problems. The availability of a library of ontology design patterns is an important step toward achieving the ultimate goal of turning ontology design into a structured and reproducible engineering process. The pattern library also includes patterns for reengineering non-ontological resources (such as thesauri, classification schemas, etc.) into ontologies.

In addition, as part of the methodological framework, three models are proposed to represent information about ontology networks. They play a critical role, as they allow keeping track of the provenance, purpose, and design of ontologies, as well as covering multilinguality issues. The three models are:

- The *Ontology Metadata Vocabulary (OMV).* An ontology that defines classes and relations to describe authoring aspects, ontology type, purpose, etc.
- The *Collaborative Ontology Design Ontology (C-ODO).* An ontology network that enables designers to describe design entities (ontologies, modules, ontology elements, requirements, activities, tools, reusable knowledge, teams, people, etc.).
- The *Linguistic Information Repository (LIR).* An ontology that defines a set of linguistic classes, whose nature accounts for the localization of ontology terms in a particular language.

1.3 Ontology Engineering Activities

The second part of the book provides the reader with a description of the key activities relevant to the ontology engineering life cycle in a networked world. For each activity, a general introduction, methodological guidelines, practical examples (where possible), and the technological support within the NeOn Toolkit (if available) are provided. Methodological guidelines are explained using a common structure, which includes process or activity definition, goal, input and output, actors involved, and a graphical workflow, which describes how the process or activity should be carried out. This structured way of explaining the guidelines maximizes the pedagogical value of the book.

The starting point to develop an ontology network is the gathering of the requirements the ontology should fulfill. This activity is called ontology requirements specification and is described in Chap. 5.

Once requirements are collected, ontology practitioners are encouraged to follow a reuse approach in the ontology building process, which allows speeding up the ontology network development process, saving time and money, and promoting the application of good practices. In this context, both non-ontological resources (Chap. 6) and ontological resources (Chap. 7) can be reused.

An important aspect in a networked world, which involves different natural languages and cultures, is the localization of the ontologies. This activity is described in Chap. 8.

Another key aspect in the ontology network development is the ontology evaluation activity, which is performed at different levels and according to different criteria, as explained in Chap. 9.

Additionally, modularization also needs to be taken into account in the ontology network development according to three different aspects: (1) designing modular ontologies, (2) modularizing existing ontologies, and (3) reusing ontology modules. Methodological guidelines for modularizing existing ontologies are presented in Chap. 10.

Ontology networks need to be kept up to date in order to reflect changes and updates. To this purpose, methodological guidelines for ontology evolution are provided in Chap. 11.

Finally, finding alignments between ontologies is an important task for ontology engineering in a networked world, and is covered in Chap. 12, which provides methodological guidelines for this activity.

1.4 The NeOn Toolkit

The third part of the book presents an overview of the NeOn Toolkit (Chap. 13), focusing in particular on the user interaction side and a detailed description of the plugins, which are most critical to the ontology development process.

Proper management of ontology engineering projects in a networked world requires careful planning, and to this purpose it is recommended that an ontology project plan and schedule is defined. To support this activity, a NeOn Toolkit plugin, called gOntt, has been developed, which is described in Chap. 14.

The tasks of locating, selecting, and accessing NeOn Toolkit plugins are supported by the Kali-ma plugin (Chap. 15). This plugin, which exploits the versatility of the C-ODO Light model, assists ontology engineers and project managers in carrying out such tasks through a unified, shared interaction mode.

Visualizing and navigating ontology networks is a key issue for ontology engineering. In this sense, the NeOn Toolkit provides a novel plugin called KC-Viz (Chap. 16), which exploits an innovative ontology summarization method to support a “middle-out ontology browsing” approach, where it becomes possible to navigate ontologies starting from the most information-rich nodes (*key concepts*).

Finally, reasoning with ontology networks is another key activity in ontology engineering. Chapter 17 presents (a) the NeOn Toolkit query plugin, which allows users to query ontologies in the NeOn Toolkit via the RDF query language SPARQL, (b) the NeOn Toolkit reasoning plugin, which allows for standard reasoning tasks, such as materializing inferences and checking consistency in ontologies, and (c) the RaDON plugin, which supports users in diagnosing and resolving inconsistencies in networked ontologies.

1.5 Case Studies

The fourth and last part of the book describes how the NeOn methods and tools have been applied in three real-world case studies in the fishery and pharmaceutical domains.

- Knowledge management at FAO (Chap. 18). This case study is centered on fisheries data[1] and aims to build a system to enable fisheries experts to have a unified view of the distributed data relevant to fisheries stocks. The result is a prototype of a Fisheries Stock Depletion Assessment System (FSDAS), which illustrates the advantages derived from enriching data with explicit semantics.
- Electronic invoice management in the pharmaceutical sector: the PharmaInnova case (Chap. 19). This chapter deals with the development of an ontology network for automating the exchange of electronic invoices in the pharmaceutical sector.
- Integrating product information in the pharmaceutical sector (Chap. 20). This case study focuses on the development of a network of interconnected pharmaceutical ontologies to provide an integrated view of different drug terminologies.

[1] http://www.fao.org/fishery/en

Part I
NeOn Methodology Framework

Chapter 2
The NeOn Methodology for Ontology Engineering

Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Mariano Fernández-López

Abstract In contrast to other approaches that provide methodological guidance for ontology engineering, the NeOn Methodology does not prescribe a rigid workflow, but instead it suggests a variety of pathways for developing ontologies. The nine scenarios proposed in the methodology cover commonly occurring situations, for example, when available ontologies need to be re-engineered, aligned, modularized, localized to support different languages and cultures, and integrated with ontology design patterns and non-ontological resources, such as folksonomies or thesauri. In addition, the NeOn Methodology framework provides (a) a glossary of processes and activities involved in the development of ontologies, (b) two ontology life cycle models, and (c) a set of methodological guidelines for different processes and activities, which are described (a) functionally, in terms of goals, inputs, outputs, and relevant constraints; (b) procedurally, by means of workflow specifications; and (c) empirically, through a set of illustrative examples.

2.1 Introduction

Given the large increase in the number of ontologies, which are available online, ontology development is more and more becoming a reuse-centric process (Simperl 2009). In particular, the level of reuse may vary significantly, depending on whether

M.C. Suárez-Figueroa (✉) • A. Gómez-Pérez
Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo sn., 28660 Boadilla del Monte, Madrid, Spain
e-mail: mcsuarez@fi.upm.es; asun@fi.upm.es

M. Fernández-López
Escuela Politécnica Superior, Universidad San Pablo CEU, Urbanización Montepríncipe sn., 28668 Boadilla del Monte, Madrid, Spain
e-mail: mfernandez.eps@ceu.es

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_2, © Springer-Verlag Berlin Heidelberg 2012

it concerns (a) other ontologies, such as DOLCE[1], SUMO (Pease et al. 2002), and Kowien[2]; (b) ontology modules (Cuenca-Grau et al. 2007); (c) ontology statements and ontology design patterns (Gangemi 2007; Presutti and Gangemi 2008); and (d) non-ontological resources (Jimeno-Yepes et al. 2009), such as thesauri, lexicons, DBs, UML diagrams, and classification schemas (e.g., NAICS[3] and SOC[4]).

Thus, in this context ontology development can be then characterized as the construction of a network of ontologies, where the different resources may be managed by different people, possibly in different organizations.

Given this new vision of ontology engineering by reuse, it then becomes important to provide strong methodological support for the collaborative development of ontology networks.

Methodological frameworks are widely accepted in different mature fields (Fernández-López 1999), like Software Engineering and Knowledge Engineering. Such methodological frameworks cover aspects, such as development process, life cycle models, as well as the methods, techniques, and tools that can be used to support the development process. Accordingly, a mature methodology for developing ontologies should also cover these aspects.

This chapter describes the *NeOn Methodology* for building ontologies and ontology networks, a scenario-based methodology that supports different aspects of the ontology development process, as well as the reuse and dynamic evolution of networked ontologies in distributed environments, where knowledge is introduced by different people (domain experts, ontology practitioners) at different stages of the ontology development process.

This methodology includes the following components:

- The *NeOn Glossary* (Sect. 2.2), which identifies and defines the processes and activities potentially involved in the ontology network construction.
- A *set of nine scenarios for building ontologies and ontology networks,* which are described in Sect. 2.3. Each scenario is decomposed in different processes and activities taken from those included in the *NeOn Glossary*.
- *Two ontology network life cycle models* (Sect. 2.4) that specify how to organize the processes and activities of the NeOn Glossary into *phases*[5].
- A *set of prescriptive methodological guidelines for processes and activities* (Sect. 2.5).

[1] http://www.loa-cnr.it/DOLCE.html

[2] Skill Ontology from the University of Essen, which defines concepts representing the competencies required to describe job position requirements and job applicant skills. Available at http://www.kowien.uni-essen.de/publikationen/konstruktion.pdf

[3] North American Industry Classification System, which provides industry-sector definitions for Canada, Mexico, and the United States to facilitate uniform economic studies across the boundaries of these countries. Available at http://www.census.gov/epcd/www/naics.html

[4] Standard Occupational Classification, which classifies workers into occupational categories (23 major groups, 96 minor groups, and 449 occupations). Available at http://www.bls.gov/soc/

[5] A phase is a distinct period or stage in a process of development.

In addition to applying the NeOn Methodology to the development of the ontology networks associated with use cases of the NeOn project as shown in Chaps. 18, 19, and 20. This methodology has been used to build ontology networks in different domains and areas and by people with diverse background, for example, and just to name a few, in e-employment (Villazón-Terrazas et al. 2011), in education (Clemente et al. 2011), in tourism (Lamsfus et al. 2009), and in mobile environments (Poveda-Villalón et al. 2010).

Finally, it is worth mentioning that the NeOn Methodology can also be used within the Linked Data initiative (Bizer et al. 2009) since this is based on knowledge resource reused and re-engineering as well as on mapping resources. Publishing Linked Data is a process that involves a high number of activities, design decisions as well as a wide range of technologies. The main activities are (1) identification of the data sources, (2) vocabulary modeling, (3) generation of the RDF data, (4) publication of the RDF data, and (5) linking the RDF data with other datasets in the cloud. In the vocabulary modeling activity, ontologies to model the data contained in the selected sources should be developed. The most important recommendation here is to reuse as much as possible available knowledge resources that model the knowledge needed. In this regard, the NeOn Methodology provides precise guidelines to help practitioners to create the vocabularies needed. One example of the use of the NeOn Methodology in this initiative can be found in (Vilches-Blázquez et al. 2010).

2.2 The NeOn Glossary

The *NeOn Glossary* identifies and defines the processes and activities potentially involved in the ontology network construction. This glossary has been established by a consensus reaching process among ontology experts and is a first step in addressing the lack of a standard glossary in Ontology Engineering – in contrast with the Software Engineering field that can claim the IEEE Standard Glossary of Software Engineering Terminology (IEEE 1990). The NeOn Glossary of Processes and Activities (Suárez-Figueroa 2010)[6] includes 59 processes and activities listed in Table 2.1.

2.3 Nine Scenarios for Building Ontology Networks

In the NeOn Methodology framework, a set of nine flexible scenarios for collaboratively building ontologies and ontology networks, placing special emphasis on reusing and re-engineering knowledge resources (ontological and non-ontological), has been identified.

[6] http://mayor2.dia.fi.upm.es/oeg-upm/files/pdf/NeOnGlossary.pdf

Table 2.1 NeOn Glossary of processes and activities

Processes	
Ontology aligning	Non-ontological resource reuse
Ontology design pattern reuse	Ontological resource reuse
Ontology module reuse	Ontology reuse
Ontology re-engineering	Ontology statement reuse
Non-ontological resource re-engineering	Ontology validation
Activities	
Ontology annotation	Ontology merging
Ontology assessment	Ontology modification
Ontology comparison	Ontology modularization
Ontology conceptualization	Ontology module extraction
Ontology configuration management control	Ontology partitioning
Ontology customization	Ontology population
Ontology diagnosis	Ontology pruning
Ontology documentation	Ontology quality assurance
Ontology elicitation	Ontology repair
Ontology enrichment	Ontology requirements specification
Ontology environment study	Non-ontological resource reverse Engineering
Ontology evaluation	Non-ontological resource transformation
Ontology evolution	Ontology restructuring
Ontology extension	Ontology reverse engineering
Ontology feasibility study	Scheduling
Ontology formalization	Ontology search
Ontology forward engineering	Ontology selection
Ontology implementation	Ontology specialization
Ontology integration	Ontology summarization
Knowledge acquisition for ontologies	Ontology translation
Ontology learning	Ontology update
Ontology localization	Ontology upgrade
Ontology mapping	Ontology verification
Ontology matching	Ontology versioning

Figure 2.1 presents the set of the nine most plausible scenarios for building ontologies and ontology networks. Directed arrows with associated numbered circles represent the different scenarios. Each scenario is decomposed into different processes or activities. Processes and activities are represented with colored circles or with rounded boxes and are defined in the NeOn Glossary of Processes and Activities presented in Sect. 2.2. Figure 2.1 also shows (as dotted boxes) the existing knowledge resources to be reused, and the possible outputs that result from the execution of some of the presented scenarios.

This section includes, as independent subsections, the most common scenarios that may unfold during the ontology network development. However, the reader should keep in mind that this list is not meant to be exhaustive.

- *Scenario 1: From specification to implementation*. The ontology network is developed from scratch, that is, without reusing available knowledge resources.

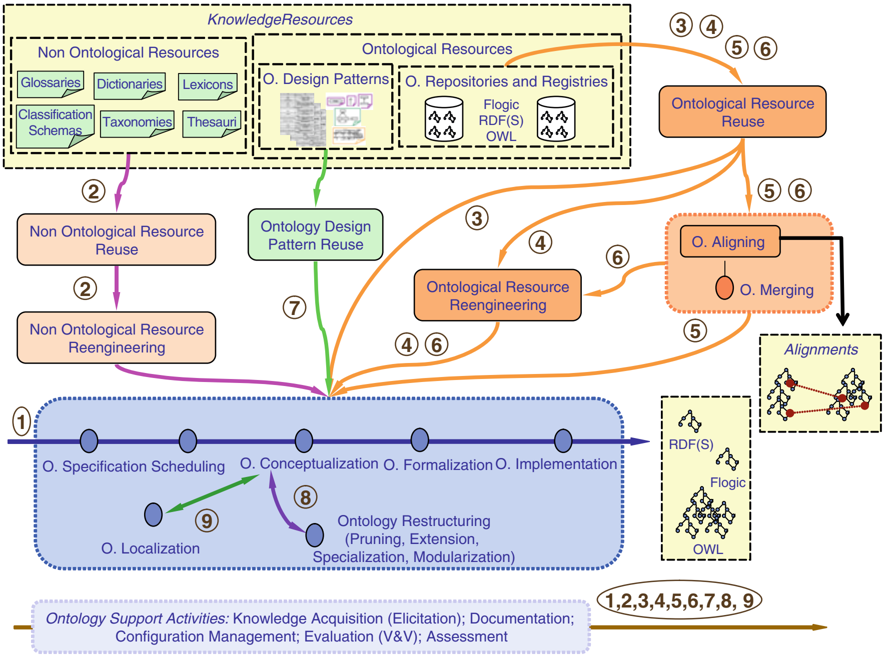

KnowledgeResources
Non Ontological Resources
Glossaries
Dictionaries
Lexicons
Classification Schemas
Taxonomies
Thesauri
Ontological Resources
O. Design Patterns
O. Repositories and Registries
Flogic
RDF(S)
OWL
③ ④ ⑤ ⑥
Ontological Resource Reuse
②
Non Ontological Resource Reuse
Ontology Design Pattern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

Fig. 2.1 Scenarios for building ontologies and ontology networks

- *Scenario 2: Reusing and re-engineering non-ontological resources*. This scenario covers the case where ontology developers need to analyze non-ontological resources and decide, according to the requirements the ontology should fulfill which non-ontological resources can be reused to build the ontology network. The scenario also covers the task of re-engineering the selected resources into ontologies.
- *Scenario 3: Reusing ontological resources*. Here, ontology developers reuse ontological resources (ontologies as a whole, ontology modules, and/or ontology statements).
- *Scenario 4: Reusing and re-engineering ontological resources*. Here, ontology developers both reuse and re-engineer ontological resources.
- *Scenario 5: Reusing and merging ontological resources*. This scenario unfolds only in those cases where several ontological resources in the same domain are selected for reuse and when ontology developers wish to create a new ontological resource from two or more ontological resources.
- *Scenario 6: Reusing, merging, and re-engineering ontological resources*. This scenario is similar to Scenario 5; however, here developers decide not to use the set of merged resources as it is, but to re-engineer it.
- *Scenario 7: Reusing ontology design patterns (ODPs)*. Ontology developers access ODPs repositories to reuse them.

- *Scenario 8: Restructuring ontological resources*. Ontology developers restructure (modularizing, pruning, extending, and/or specializing) ontological resources to be integrated in the ontology network being built.
- *Scenario 9: Localizing ontological resources*. Ontology developers adapt an ontology to other languages and culture communities, thus producing a multilingual ontology.

Knowledge acquisition, documentation, configuration management, evaluation, and assessment should be carried out during the whole ontology network development, that is, in any scenario used for developing the ontology network. The intensity of such support activities depends on the concrete phase of the development progress.

It is worth mentioning that these scenarios can be combined in different and flexible 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 Fig. 2.1 shows, the results of any other scenario should be integrated in the corresponding activity of Scenario 1.

The following subsections present the various scenarios identified; each subsection includes (a) motivation for the scenario; (b) sequence of processes, activities, and tasks to be carried out, where the processes and activities included are taken from the NeOn Glossary of Processes and Activities (Sect. 2.2); and (c) outcomes for the scenario.

2.3.1 Scenario 1: From Specification to Implementation

This scenario refers to the development of ontologies from scratch. The scenario is made up of the core activities that have to be performed in any ontology development and should be combined with the rest of scenarios.

In this scenario, ontology developers[7] should specify first the requirements that the ontology should fulfill, by means of the *ontology requirements specification activity*. The objective of this activity is to output the ontology requirements specification document (ORSD) that includes the purpose, the scope, and the implementation language of the ontology network, the target group, and the intended uses of the ontology network, as well as the set of requirements that the ontology network should fulfill, mainly in the form of *competency questions* (CQs)[8] and a pre-glossary of terms. Prescriptive methodological guidelines for this activity are provided in Chap. 5.

[7] In this book, ontology developers refer to software developers and ontology practitioners involved in the development of ontologies.

[8] An example of CQ can be “where is located the device Z? The device Z is at coordinates X, Y”.

After the ontology requirements specification activity, it is recommended to carry out a look for candidate knowledge resources (ontologies, non-ontological resources, and ontology design patterns) to be reused in the development, using as input terms included in the ORSD. These candidate resources provide clues for the identification of the scenarios to be followed during the ontology development. Then, the *scheduling activity* must be carried out, using the ORSD and the results of such a look for resources. During the scheduling activity, the team establishes the ontology network life cycle and the human resources needed for the ontology project. Chapter 14 presents guidelines and a tool for performing the scheduling of ontology development projects.

Then, the ontology developers assigned to the ontology project should carry out (1) the *ontology conceptualization activity*, in which knowledge is organized and structured into meaningful models at the knowledge level; (2) the *ontology formalization activity*, in which the conceptual model is transformed into a semi-computable model; and (3) the *ontology implementation activity*, in which a computable model (implemented in an ontology language) is generated.

The principal output is a network of ontologies that represents the expected domain implemented in an ontology language (OWL[9], F-Logic, etc.). In addition, a broad range of documents, such as the ontology requirements specification document, the ontology description document, and the ontology evaluation document, will be generated as output by the different activities.

2.3.2 Scenario 2: Reusing and Re-engineering Non-Ontological Resources

Currently, ontology developers are realizing the benefits of “not reinventing the wheel” at each ontology development. They are starting to reuse as much as possible non-ontological resources, such as classification schemes, thesauri, lexicons, and folksonomies, built by others that already have reached some degree of consensus, with the aim of speeding up the ontology development process (Villazón-Terrazas et al. 2010). The reuse of such resources involves necessarily their re-engineering into ontologies. Therefore, this scenario unfolds in those cases in which ontology developers wish to reuse the non-ontological resources at their disposal.

As Fig. 2.1 shows (by arrows with the number 2), ontology developers should accomplish first the non-ontological resource reuse process and then choose the most suitable non-ontological resources (thesauri, glossaries, databases, etc.) to be used for building the ontology network. Such non-ontological resources cover to some extent the domain of the ontology network being built. If ontology developers

[9] http://www.w3.org/TR/owl-ref/

decide that one or more resources are useful for the ontology network development, then the non-ontological resource re-engineering process should be carried out to transform the selected non-ontological resources into ontologies. After this process, ontology developers should use the resultant ontologies as input of some of the activities included in Scenario 1 (explained in Sect. 2.3.1), as shown in Fig. 2.1.

The activities for carrying out the non-ontological resource reuse process are briefly explained below; prescriptive methodological guidelines for this activity are described in Chap. 6:

1. *Activity 1. Search non-ontological resources*. The goal of the activity is to find non-ontological resources in highly reliable websites, domain-related sites, and resources within organizations. The input for this activity is the ontology requirements specification document (ORSD).
2. *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 obtained in Activity 1. To carry out this activity, the following criteria should be used: coverage, precision, and consensus about the knowledge and terminology used in the resource, which is a subjective criterion.
3. *Activity 3. Select the most appropriate non-ontological resources*. The goal of this activity is to select the most appropriate non-ontological resources from those candidates obtained in Activity 2.

As mentioned before, the goal of the non-ontological resource re-engineering process is to transform a non-ontological resource into an ontology. This process can be divided into the following activities, and prescriptive methodological guidelines for performing them are included in Chap. 6:

1. *Activity 1. Non-ontological resource reverse engineering*. The goal of this activity is to analyze a non-ontological resource in order to identify its underlying components and create representations of the resource at the different levels of abstraction (design, requirements, and conceptual).
2. *Activity 2. Non-ontological resource transformation*. The goal of this activity is to generate a conceptual model from the non-ontological resource.
3. *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 identified in Activity 2.

The principal output is an ontology network that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). Furthermore, a broad range of documents containing the requirements specification, the ontology documentation, the ontology evaluation, etc. will be generated as output of different activities. Additionally, the non-ontological resources selected to be reused have been “ontologized” by means of the non-ontological resource re-engineering activity.

2.3.3 Scenario 3: Reusing Ontological Resources

As more ontological resources are available in ontology repositories and on the Internet[10], ontology developers are starting to reuse them not only with the idea of "not reinventing the wheel", but also with the aim of taking advantage of them. Thus, this scenario unfolds in those cases in which ontology developers have at their disposal ontological resources useful for their problem and that can be reused in the ontology development.

As Fig. 2.1 shows (by arrows with the number 3), ontology developers should perform the *ontological resource reuse process*, which is composed of the following activities:

1. *Activity 1. Ontology search*. Ontology developers search for candidate ontological resources that satisfy the requirements in repositories and registries like Swoogle[11], Watson[12], and Sindice[13]. These ontological resources could be implemented in different languages or could be available in different ontology tools.
2. *Activity 2. Ontology assessment*. Ontology developers must inspect the content and granularity of the ontological resources obtained in Activity 1. The goal of this activity is to find out if such resources satisfy the needs identified in the ORSD.
3. *Activity 3. Ontology comparison*. Ontology developers should compare the ontological resources assessed in Activity 2, taking into account a set of criteria identified by developers (e.g., reuse economic cost, code clarity, and content quality).
4. *Activity 4. Ontology selection*. Ontology developers should select the set of ontological resources that are the most appropriate for their ontology network requirements, based on the comparisons obtained in Activity 3.

   After selecting the most appropriate ontological resources, ontology developers should define the reuse mode; that is, ontology developers need to decide how they will reuse the selected ontological resources. There are three possible modes:

   - The ontological resources selected will be reused as they are.
   - The ontology re-engineering activity should be carried out with the ontological resources selected.
   - Some ontological resources will be merged to obtain a new ontological resource.

[10] See, for example, a list of novel ontology search engines described at: http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/SemanticWebSearchEngines

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

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

[13] http://sindice.com/

Before reusing the selected ontological resources by means of any reuse mode, it is also convenient to evaluate these resources through the *ontology evaluation activity*.

5. *Activity 5. Ontology integration*. Ontology developers should include, as they are, the ontological resources selected (the code) in Activity 4 into the ontology network being built following the activities of Scenario 1 (Sect. 2.3.1).

Prescriptive methodological guidelines to reuse general ontologies are provided in Chap. 7.

The principal output is an ontology network that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). Additionally, a broad range of documents including the requirements specification, the ontology documentation, the ontology evaluation, etc. will be generated as output of different activities.

2.3.4 Scenario 4: Reusing and Re-engineering Ontological Resources

This scenario unfolds in those cases in which ontology developers have at their disposal ontological resources useful for their problem, which can be reused in the ontology network development. However, such resources are not exactly useful as they are, so they should be modified (i.e., re-engineered) to serve to the intended purpose or problem.

As Fig. 2.1 shows (by arrows with the number 4), ontology developers should perform first the *ontological resource reuse process* to select the most suitable ontological resources to be used for building the ontology network. Then, they should carry out the *ontological resource re-engineering process* to modify the selected ontological resources. Finally, they should use the resultant ontological resources as input to some of the activities included in Scenario 1 (explained in Sect. 2.3.1), as shown in Fig. 2.1.

Specifically, ontology developers should carry out some activities as part of the ontological resource reuse process; such activities are the following: *ontology search*, *ontology assessment*, *ontology comparison*, and *ontology selection* as already explained in Scenario 3 (Sect. 2.3.3).

After the ontology selection activity, ontology developers should decide how they will reuse the ontological resources. They should also decide whether to perform the ontological resource re-engineering process with the selected ontological resources because these resources may not absolutely correct for the concrete use case as they are and they need to be transformed in some way.

The ontological resource re-engineering process proposed here has been created taking as inspiration the software re-engineering process (Byrne 1992). It is composed of the following activities: *ontological resource reverse engineering*, *ontological resource restructuring*, and *ontological resource forward engineering*.

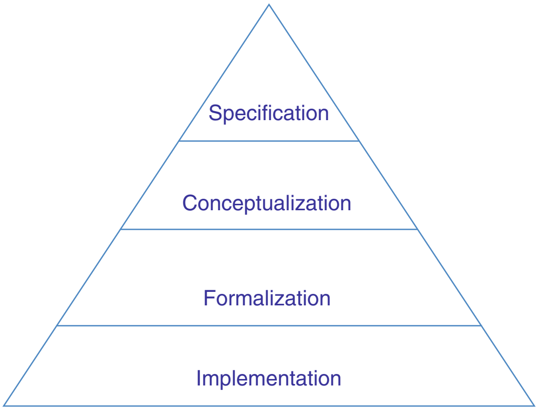


Fig. 2.2 Levels of abstraction for the ontological resource re-engineering process

Additionally, this process is related to the levels of abstraction shown in Fig. 2.2 that are based on (Byrne 1992) and are described below.

- Specification is the highest level of abstraction. In this level, requirements, purpose, and scope, among other components of the specification, are described.
- In the conceptualization level, ontology characteristics such as structure and components are described. The knowledge that the ontology represents is organized following a set of knowledge representation primitives (concepts, relations, etc.). In this level, the knowledge is structured in meaningful models at the knowledge level (Newell 1982). To organize the knowledge, intermediate representations based on tabular and graphical notations (Gómez-Pérez et al. 2003), which can be understood by ontology practitioners, can be used.
- In the formalization level, the formal or semi-computable model that was used to transform the conceptual model is described.
- The implementation level is the lowest abstraction level. Here, the ontology description focuses on implementation characteristics and is represented in an ontology language understandable by computers and usable by automatic reasoners.

Figure 2.3 presents the ontological resource re-engineering model. This model suggests different paths to re-engineer an ontological resource, taking into account the levels of abstraction presented in Fig. 2.2. Examples of these paths are:

- At implementation level: from ontological resource 1 code to ontological resource 2 code
- At formalization level: reverse engineering (from code 1 to formalization 1), restructuring formalization 1 to obtain formalization 2, and forward engineering to obtain code of resource 2
- At conceptualization level: reverse engineering (from code 1 to conceptualization 1), restructuring conceptualization 1 to obtain conceptualization 2, and forward engineering to obtain formalization or implementation 2

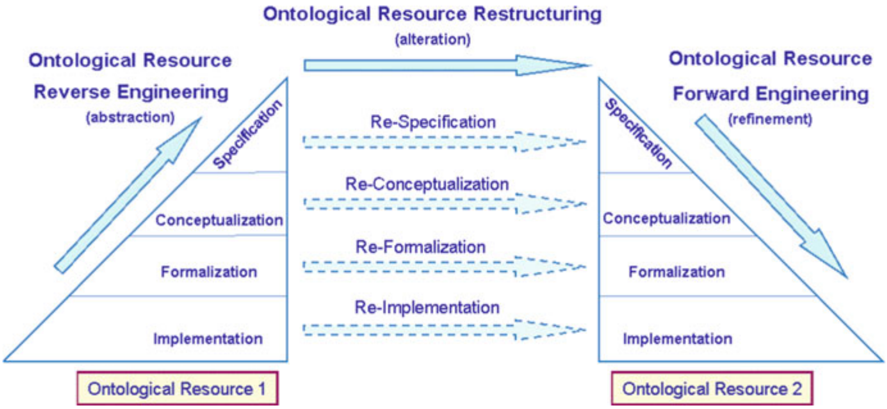


Fig. 2.3 Ontological resource re-engineering model

- At specification level: reverse engineering (from code 1 to specification 1), restructuring specification 1 to obtain specification 2, and forward engineering to obtain conceptualization, formalization, or implementation 2

The choice of a concrete path depends on the ontological resource characteristics that have to be changed. Thus, in Fig. 2.3 the following types of changes can be distinguished:

- Re-specification. If the ontology developer restructures the requirements specification, she changes requirements, purpose and scope, among other elements of the requirements specification. For example, changes in requirements, addition or deletion of requirements, etc.
- Re-conceptualization. If she restructures the conceptualization, changes might refer to modification of ontology structure, modification of granularity and richness of the knowledge, removal or addition of axioms, restructuration of ontology architecture (modularization), inclusion of new concepts, use of ontology design patterns, etc.
- Re-formalization. If she restructures the formalization level, the changes refer to formalization characteristics (such as changing the ontology paradigm from description logic to frames).
- Re-implementation. If she restructures the implementation level, the changes are focused on implementation characteristics that are tightly related to the ontology implementation language (e.g., translation from RDF(S) to OWL). Other changes could be conforming to coding standards, improving code readability, renaming code items, etc.

Ontology developers should decide at which level they need to carry out the ontological resource re-engineering process. Once ontology developers have decided the level, they should carry out the ontological resource re-engineering process, and then they should integrate the result of such a process (code, formalization, conceptualization, or specification) into the corresponding activity of Scenario 1 (Sect. 2.3.1).

The principal outcome is an ontology network that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). Additionally, a broad range of documents including requirements specification, ontology documentation, ontology evaluation, etc. will be generated as output of different activities. Furthermore, new ontological resources from those selected for their reuse are generated through the ontological resource re-engineering process. Such new resources can be considered as new versions of the ontological resources after the re-engineering process.

2.3.5 Scenario 5: Reusing and Merging Ontological Resources

This scenario unfolds in those cases where several ontological resources in the same domain can be selected for reuse and when the ontology developer wishes to create a new ontological resource from two or more, possibly overlapping, ontological resources. It could also occur that the ontology developer wishes only to establish alignments among the ontological resources selected in order to create the ontology network.

As Fig. 2.1 shows (by arrows with the number 5), ontology developers should perform first the *ontological resource reuse process* to select the most suitable ontological resources that will be used for building the ontology network. Concretely, ontology developers should carry out the activities presented in Scenario 3 (Sect. 2.3.3) as part of the ontological resource reuse process. After the ontology selection activity, ontology developers should decide how they will reuse the ontological resources selected. In this scenario, ontology developers decide to perform the following activities because the selected resources are valid as they are, but not in a complete way, if they were reused in a separate fashion. The activities to be performed are the following:

1. *Activity 1. Ontology aligning*. Ontology developers carry out this activity with the aim of obtaining a set of alignments among the selected ontological resources. Prescriptive methodological guidelines for this activity are described in Chap. 12.
2. *Activity 2. Ontology merging*. Ontology developers can merge the selected ontological resources using the alignments (output of Activity 1) to obtain a new ontological resource from the overlapping selected ones.

Ontology developers have here two different possibilities: (1) to establish the mappings among such selected resources and (2) to establish the mappings and also to merge the selected resources.

After this activity, ontology developers should use the resultant merged ontological resource as input of some of the activities included in Scenario 1 (explained in Sect. 2.3.1), as shown in Fig. 2.1.

The principal outputs are (a) a set of alignments among the selected ontological resources and (b) a set of new ontological resources to be integrated as they are in the ontology network.

2.3.6 Scenario 6: Reusing, Merging, and Re-engineering Ontological Resources

This scenario unfolds in those cases in which several ontological resources in the same domain can be selected to build the ontology network. Ontology developers decide to create a new ontological resource merging two or more, possibly overlapping, ontological resources. Such a merged ontological resource is not useful as it is, so it should be modified (i.e., re-engineered) to serve to the intended purpose.

As Fig. 2.1 shows (see arrows with number 6), ontology developers should perform first the *ontological resource reuse process* to select the most suitable ontological resources for building the ontology network (as explained in Scenario 3 (Sect. 2.3.3)). Then, they should decide how they will reuse the selected ontological resources. It is in this scenario where ontology developers decide to perform the *ontology aligning and ontology merging* activities because the selected resources are valid but not in a complete way for the concrete case if they are considered separately, as explained in Scenario 5 (Sect. 2.3.5). After merging the selected resources, they should carry out the *ontological resource re-engineering process* as described in Scenario 4 (Sect. 2.3.4). After that, they should use the resultant ontological resource as input of some of the activities included in Scenario 1 (explained in Sect. 2.3.1), as shown in Fig. 2.1.

The principal output is an ontology network that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). Additionally, a broad range of documents including the requirements specification, the ontology documentation, the ontology evaluation, etc. will be generated as output of different activities.

Furthermore, a merged ontological resource, taken from those selected for reuse, and a re-engineered merged ontological resource are generated. Alignments between the ontological resources selected are also outputs of this scenario.

2.3.7 Scenario 7: Reusing Ontology Design Patterns

Recently, within the Ontology Engineering field, *ontology design patterns* (ODPs) have emerged as (1) a way of helping ontology developers to model OWL ontologies (Gangemi 2005; Pan et al. 2007) and (2) a new mode of encoding best practices, based on experiences and knowledge of "good" solutions. As any other type of patterns, ODPs are perceived as having three kinds of benefits (Blomqvist et al. 2009): (1) reuse benefits, (2) guidance benefits, and (3) communication benefits. ODPs can be found in online libraries that include both the description and the OWL code associated to the patterns as, for example, "the Ontology Design Pattern Wiki"[14], or they can be obtained from the "Semantic Web Best Practices and Deployment"[15] working group. Thus, this scenario unfolds in those cases where best practices can be applied to the development of ontology networks.

Ontology developers work on the development of an ontology network and very often encounter problems regarding the way in which certain knowledge should be modeled. This may happen during the ontology conceptualization activity, the ontology formalization activity, or during the ontology implementation activity. In these situations, ontology developers can access on-line libraries in order to find modeling solutions.

Ontology developers should perform the *ontology design pattern reuse process* to select the most suitable ODPs for building the ontology network. The principal output of this reuse process is a set of ontology design patterns integrated into the ontology network being developed. Guidelines to perform this reuse are provided in Chap. 3.

2.3.8 Scenario 8: Restructuring Ontological Resources

This scenario unfolds in those cases where the knowledge contained in the conceptual model of the ontology network should be corrected and reorganized to obtain the network that covers the ontology requirements.

Ontology developers should perform the *ontology restructuring activity* to modify the ontology network being built, after the ontology conceptualization activity. The *ontology restructuring activity* can be performed by executing any of the following sub-activities, combining them in any manner and order:

- *Ontology modularization activity*. Ontology developers create different ontology modules in the ontology network, which facilitates the reuse of the knowledge included in the network. Prescriptive methodological guidelines to carry out this activity are presented in Chap. 10.
- *Ontology pruning activity*. Ontology developers prune those branches of the taxonomies included in the ontology network that are considered not necessary to cover the ontology requirements.
- *Ontology enrichment activity*. This activity can be carried out by performing any of the two sub-activities that follow:

[14] http://ontologydesignpatterns.org/

[15] http://www.w3.org/2001/sw/BestPractices/

- *Ontology extension activity*. Ontology developers extend the ontology network, including (in width) new concepts and relations.
- *Ontology specialization activity*. Ontology developers specialize those branches of the ontology network that require more granularity and include more specialized concepts and relations.

Note that this activity (ontology restructuring) can be performed (1) in an independent way as explained in this scenario or (2) as part of the ontological resource re-engineering process, as described in Scenario 4 in Sect. 2.3.4.

The principal output is a conceptual model of the ontology network that represents the expected domain.

2.3.9 Scenario 9: Localizing Ontological Resources

Although access to top-quality ontologies (e.g., Galen, CYC, or AKT) 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 their own languages. Moreover, more and more ontology-based systems are being built for multilingual applications (e.g., multilingual machine translation or multilingual information retrieval). For these reasons, the need for multilingual ontologies has increased. Thus, this scenario unfolds in those cases in which the ontology network to be developed should be written in different natural languages.

Ontology developers should perform the *ontology localization activity* once the ontology has been conceptualized and restructured. This activity requires the translation of all the ontology terms into another natural language (Spanish, French, German, etc.) different from the language used in the conceptualization, using multilingual thesauri and electronic dictionaries (e.g., EuroWordNet[16]). This ontology localization activity is composed of the following tasks (Espinoza et al. 2009):

1. *Task 1. Selecting the most appropriate linguistic assets*. The goal of this task is to select the most appropriate linguistic assets that help to reduce the cost, to improve the quality of the localization, and to increase the consistency of the localization activity.
2. *Task 2. Selecting ontology label(s) to be localized*. The goal of this task is to select the ontology label(s) to be localized.
3. *Task 3. Obtaining ontology label translation(s)*. The goal of this task is to obtain the most appropriate translation in the target language for each ontology label.
4. *Task 4. Evaluating label translation(s)*. The goal of this task is to evaluate the label translations in the target language.

[16] http://www.illc.uva.nl/EuroWordNet/

5. *Task 5. Updating the ontology*. The goal of this task is to update the ontology with the label translations obtained for each localized label. The task output is an ontology enriched with labels in the target language associated to each localized term.

Prescriptive methodological guidelines for localizing ontologies are presented in Chap. 8.

After this localization activity, the resulting conceptual model should be integrated in the conceptualization activity of Scenario 1 (Sect. 2.3.1).

The principal outcome is a conceptual model of the ontology network in different natural languages (i.e., a multilingual conceptual model) that represents the expected domain.

2.4 Two Ontology Network Life Cycle Models

Ontologies are artifacts designed for the purpose of satisfying certain requirements and needs that are emerging in the real world.

Thus, the *ontology network development process* is defined as the process by which user's needs are translated into an ontology network. This means that the ontology network development process can be seen as a specific case of the software development process.

An *ontology network life cycle model* is defined as a model to describe how to develop (and maintain) an ontology network project; in other words, how to organize the processes and activities of the NeOn Glossary into phases or stages.

This section includes the two ontology network life cycle models, which include the *waterfall model* (Sect. 2.4.1) and the *iterative-incremental model* (Sect. 2.4.2). Additionally, it is worth mentioning that these two models are intrinsically related to the set of nine flexible scenarios for collaboratively building ontologies and ontology networks, presented in Sect. 2.3. Such a relation is due to the creation of both models and scenarios, taking into account the importance of reusing and re-engineering knowledge resources and merging resources.

2.4.1 Waterfall Ontology Network Life Cycle Models

The main characteristic of the waterfall life cycle model family proposed for the ontology network development is the representation of the stages of an ontology network as sequential phases. This model represents the stages as a waterfall. In this model, a concrete stage must be completed before the following stage begins, and no backtracking is permitted except in the case of the maintenance phase.

The main assumption for using the waterfall ontology network life cycle model proposed is that the requirements are completely known, without ambiguities, and unchangeable at the beginning of the ontology network development.

This model could be used in the following situations:

- In ontology projects with a short duration (e.g., 2 months)
- In ontology projects in which the goal is to develop an existing ontology in a different formalism or language
- In ontology projects in which the requirements are closed, for instance, to implement an ontology based on an ISO standard, or based on resources with previous consensus in the included knowledge
- In ontology projects when ontologies cover a small, well-understood domain

Taking into account the characteristics of the ontology development scenario, this model includes a set of support activities that should be performed in all of the phases. This set of support activities includes the acquisition of knowledge in the domain in which the ontology network is being developed, the evaluation (from a content-oriented perspective) and the assessment (from user and need perspectives) of the different phase outputs, project and configuration management, and documentation.

Because of the importance of reusing and re-engineering knowledge resources and merging ontological resources, the following five significantly different versions of the waterfall ontology network life cycle model have been defined. These versions have been created incrementally (i.e., the four-phase is the basis for the five-phase, the five-phase is the basis for the six-phase, etc.).

Before detailing the different versions, they can be summarized in the following way:

- The four-phase waterfall model. It represents the stages of an ontology network, starting with the initiation phase and going through the design phase and the implementation phase to the maintenance phase.
- The five-phase waterfall model. It extends the four-phase model with the reuse of ontological resources as they are.
- The five-phase + merging phase waterfall model. It is a special case of the five-phase model. It includes the merging phase to obtain a new ontological resource from two or more ontological resources previously selected in the reuse phase.
- The six-phase waterfall model. It extends the five-phase model with re-engineering phase. It allows the re-engineering of knowledge resources (ontological and non-ontological). It could happen that several knowledge resources are transformed into ontologies in the re-engineering phase.
- The six-phase + merging phase waterfall model. It extends the six-phase model by including the merging phase after the reuse phase.

2.4.1.1 The Four-Phase Waterfall Ontology Network Life Cycle Model

This model represents the stages of an ontology network, starting with the initiation phase and going through the design phase, the implementation phase to the maintenance phase.

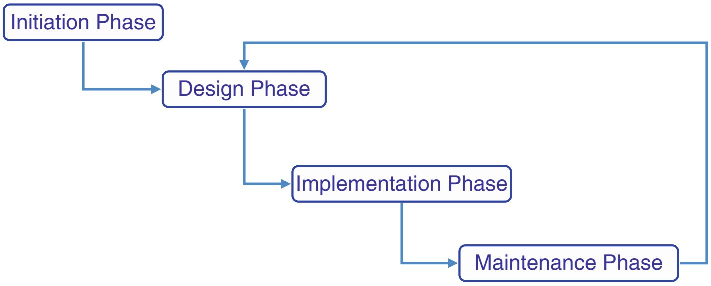


Fig. 2.4 The four-phase waterfall ontology network life cycle model

The model proposed is shown in Fig. 2.4, and the main purposes and outcomes for each phase in the model are the following:

- *Initiation phase*. In this phase, it is necessary to produce an ontology requirement specification document (ORSD) (explained in Chap. 5), including the requirements that the ontology network should satisfy and taking into account knowledge about the concrete domain. Also in this phase, the approval or rejection of the ontology network development should be obtained. This phase has also as requisite to identify the development team and to establish the resources, responsibilities, and timing (i.e., the scheduling for the ontology project).
- *Design phase*. The output of this phase should be both an informal model and a formal one that satisfy the requirements obtained in the previous phase. The formal model cannot be used by computers, but it can be reused in other ontology networks.
- *Implementation phase*. In this phase, the formal model is implemented in an ontology language. The output of this phase is an ontology implemented in RDF(S), OWL, or other language that can be used by semantic applications or by other ontology networks.

  It is worth mentioning that the last two phases (design and implementation ones) are normally performed together when ontology development tools (such as NeOn Toolkit, Protégé, etc.) are used.
- *Maintenance phase*. If, during the use of the ontology network, errors or missing knowledge are detected, then the ontology development team should go back to the design phase. Additionally, in this phase the generation of new versions for the ontology network should also be carried out.

2.4.1.2 The Five-Phase Waterfall Ontology Network Life Cycle Model

This model extends the four-phase model with a new phase in which the reuse of already implemented ontological resources is considered. The main purpose in the *reuse phase* is to obtain one or more ontological resources to be reused in the ontology network being developed. The output of this reuse phase could be either an informal model or a formal one to be used in the design phase, or an implemented model (in an ontology language) to be used in the implementation phase.

For the other phases, the purposes and outcomes are the same as those presented in the four-phase model.

2.4.1.3 The Five-Phase + Merging Phase Waterfall Ontology Network Life Cycle Model

This model is a special case of the five-phase model. Now, a new phase (the *merging phase*) is added after the reuse one. This merging phase has as a main purpose to obtain a new ontological resource from two or more ontological resources selected in the reuse phase.

For the other phases, the purposes and outcomes are the same as those presented in the five-phase model.

2.4.1.4 The Six-Phase Waterfall Ontology Network Life Cycle Model

In this model, the five-phase model is taken as general basis, and a new phase (*re-engineering phase*) is included after the reuse one. This model allows the reuse of knowledge resources (ontological and non-ontological) and their later re-engineering. In this model, the reuse phase has as output one or more knowledge resources to be reused in the ontology network that is being developed. After this phase, the non-ontological resources are transformed into ontologies in the re-engineering phase; the ontological resources, on the other hand, can or cannot be re-engineered, a decision that should be taken by the ontology development team.

For the other phases, the purposes and outcomes are the same as those presented in the six-phase model.

2.4.1.5 The Six-Phase + Merging Phase Waterfall Ontology Network Life Cycle Model

This model, extended from the six-phase model, includes the *merging phase* after the reuse phase. For the other phases, the purposes and outcomes are the same as those presented in the six-phase model.

2.4.2 Iterative-Incremental Ontology Network Life Cycle Model

The main feature of this model is the development of ontology networks organized in a set of iterations (or short mini-projects with a fixed duration). Each individual iteration is similar to an ontology network project that uses any

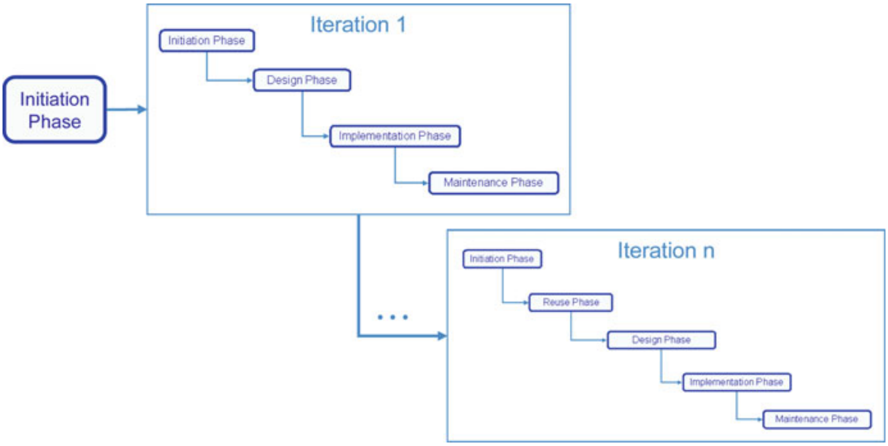


Fig. 2.5 Schematic vision of the iterative-incremental model

type of waterfall model from those presented in Sect. 2.4.1, as shown schematically in Fig. 2.5.

This model could be used in the following situations:

- In ontology projects with large groups of developers having different profiles and roles
- In ontology projects in which the development involves several different domains that are not well understood
- In ontology projects in which requirements are not completely known or can change during the ontology development

Ontology requirements specified in the ORSD can be divided in different subsets. The result of any iteration is a functional and partial ontology network that meets a subset of the ontology network requirements. Such a partial ontology network can be used, evaluated, and integrated in any other ontology network.

This model is based on the continuous improvement and extension of the ontology network resulted from performing multiple iterations with cyclic feedback and adaptation. In this way, the ontology network grows incrementally along the development. Generally, in each iteration new requirements are taken into account, but, occasionally, in a particular iteration, the partial ontology network could be only enhanced.

This model focuses on a set of basic requirements; from these requirements, a subset is chosen and considered in the development of the ontology network. The partial result is reviewed, the risk of continuation with the next iteration is analyzed and the initial set of requirements is increased and/or modified in the next iteration until the complete ontology network is developed.

The main benefit of this model is to identify and alleviate the possible risks as soon as possible. Other benefits are:

- The development team is motivated by rapidly producing an adequate ontology.
- Some priorities can be established in the set of requirements.
- The development can be possibly adapted to changes in the requirements.
- The scheduling of each iteration can be adapted based on the experience of previous iterations.

It is worth mentioning that at the beginning of the ontology network project, the number of iterations during the ontology project is influenced by:

- The decision of performing a more complete and detailed ontology requirements specification. In this case, the number of iterations will be lower.
- The decision of carrying out a simpler and less complete requirements specification, in which case more number of iterations and more revisions will be needed.

Figure 2.5 shows the schematic vision of the iterative-incremental model. The first initiation phase shown in the figure has as main outcomes the ontology network requirements and the general and global plan for the whole ontology network development. Regarding the different iterations, as mentioned before, each iteration in the iterative-incremental model can follow a different version of the waterfall model from those presented in Sect. 2.4.1. However, any version of the waterfall model to be used in the iterative-incremental model should be modified in the following way:

- No backtracking is allowed between phases in a particular iteration, because the refinement should be performed in the next iterations.
- Revising the ontology network requirements and the global plan should be carried out in the initiation phase of each iteration. Additionally, a detailed plan for the particular iteration should be performed.

2.4.3 Relation Between Scenarios and Life Cycle Models

The set of nine flexible scenarios for building ontologies and ontology networks presented in Sect. 2.3 and the two proposed ontology network life cycle models presented in this section are intrinsically related because both scenarios and life cycle models have been created (1) taking into account the importance of reusing and re-engineering knowledge resources (ontological and non-ontological) and merging ontological resources and (2) assuming a controlled setting for ontology engineering in which approaches such as mining ontologies from tags are not considered.

Table 2.2 summarizes the relationships between scenarios for building ontology networks and ontology network life cycle models. These relationships have been established based on the following:

Table 2.2 Relation between scenarios and life cycle models

	Four-phase model	Five-phase model	Five-phase + merging phase model	Six-phase model	Six-phase + merging phase model
Scenario 1	X				
Scenario 2				X	
Scenario 3		X			
Scenario 4				X	
Scenario 5			X		
Scenario 6					X
Scenario 7		X			
Scenario 8	X				
Scenario 9	X				

- Scenario 1 (as stated in Sect. 2.3.1) is for building ontology networks from scratch. The scenario mainly includes core activities such as specification, conceptualization, and implementation. This way of building ontologies fits with the stages represented in the four-phase waterfall model (initiation phase, design phase, implementation phase, and maintenance phase).
- Scenario 2 (as stated in Sect. 2.3.2) is for building ontology networks by reusing and re-engineering non-ontological resources, which is represented in the six-phase waterfall model.
- Scenario 3 (as stated in Sect. 2.3.3) is for building ontology networks by reusing ontological resources. This way of building ontologies is represented by the five-phase waterfall model.
- Scenario 4 (as stated in Sect. 2.3.4) refers to the development of ontology networks by reusing and re-engineering ontological resources. This way of building ontologies is represented by the six-phase waterfall model.
- Scenario 5 (as stated in Sect. 2.3.5) is for building ontology networks by reusing and merging ontological resources, which is represented by the five-phase + merging phase waterfall model.
- Scenario 6 (as stated in Sect. 2.3.6) refers to the development of ontology networks by reusing, merging, and re-engineering ontological resources. This way of building ontology networks is represented by the six-phase + merging phase waterfall model.
- Scenario 7 (as stated in Sect. 2.3.7) is for building ontology networks by reusing ontology design patterns, which is represented by the five-phase waterfall model.
- Scenario 8 (as stated in Sect. 2.3.8) is for building ontology networks by restructuring ontological resources. This is mainly related to the core activities already mentioned in Scenario 1. Thus, this Scenario 8 is also represented by the four-phase waterfall model.
- Scenario 9 (as stated in Sect. 2.3.9) refers to the development of ontology networks by localizing ontologies. This way of building ontologies is mainly related to Scenario 1 and thus represented by the four-phase waterfall model.

As explained in Sect. 2.4.2, the iterative-incremental model is basically formed by a set of iterations that can follow any version of waterfall ontology network life cycle model. Thus, the relation between scenarios and the iterative-incremental model depends on the different versions of waterfall model used in the iterative-incremental one, and for this reason, the relations presented in Table 2.2 are also valid for this model.

2.5 Methodological Guidelines for Processes and Activities

In the second part of this book (called Ontology Engineering Activities), methodological guidelines for a subset of the processes and activities included in the NeOn Glossary are provided. To describe each of the processes and activities included in the NeOn Methodology presented in this book, the following content is provided for most of the cases:

- A general introduction to the process or activity, where the value of the process or activity is discussed.
- The detailed guidelines proposed for carrying out the process or the activity, including the following fields: (a) *definition*, which is taken from the NeOn Glossary of Processes and Activities and included in Sect. 2.2; (b) *goal*, which explains the main objective intended to be achieved by the process or the activity; (c) *input*, which includes the resources needed for carrying out the process or the activity; (d) *output*, which includes the results obtained after carrying out the process or the activity; (e) *who*, which identifies the people or teams involved in the process or the activity; and (f) *when*, which explains in which stage of the development the process or the activity should be carried out.

  All the aforementioned information is provided in the so-called *filling cards*. These filling cards explain the information of each process and activity of the NeOn Methodology in a practical and easy way. Each card is filled according to the *filling card template* shown in Table 2.3.
- A graphical *workflow* that shows how the process or the activity should be carried out is also included. This workflow contains the inputs, outputs, actors involved, and details for carrying out a process or activity in a prescriptive manner. Additionally, methods, techniques, and tools supporting the process or activity are proposed.
- Examples explaining the guidelines proposed are also given.

It should be noted that in the framework of the NeOn Methodology, there are a wide range of prescriptive methodological guidelines for carrying out different processes and activities. Along this book, the reader can find guidelines for *Scenario 1*, particularly for ontology requirements specification (Chap. 5) and scheduling (Chap. 14), *Scenario 2* (Chap. 6), *Scenario 3* (Chap. 7), *Scenario 5* (Chap. 12), *Scenario 7* (Chap. 3), *Scenario 8*, for ontology modularization (Chap. 10), and *Scenario 9* (Chap. 8). In addition, there are also guidelines for ontology evaluation (Chap. 9) and for ontology evolution (Chap. 11).

Table 2.3 Template for the process and activity filling card

Process or Activity Name	
Definition	
Goal	
Input	Output
Who	
When	

References

Bizer C, Heath T, Berners-Lee T (2009) Linked data – the story so far. Int J Semant Web Inf Syst 5 (3):1–22

Blomqvist E, Gangemi A, Presutti V (2009) Experiments on pattern-based ontology design. In: Proceedings of the 5th international conference on Knowledge Capture (K-CAP 2009), Redondo Beach, CA, USA, 1–4 Sept 2009. ISBN: 978-1-60558-658-8

Byrne EJ (1992) A conceptual foundation for software re-engineering. In: Proceedings of the international conference on software maintenance and reengineering. IEEE Computer Society Press, Orlando, pp 226–235

Clemente J, Ramírez A, de Antonio A (2011) A proposal for student modeling based on ontologies and diagnosis rules. Expert Syst Appl 38(7):8066–8078

Cuenca-Grau B, Horrocks I, Kazakov Y, Sattler U (2007) Just the right amount: extracting modules from ontologies. In: Proceedings of the 16th international conference on world wide web, Banff, Alberta, Canada, pp 717–726. ISBN: 978-1-59593-654-7

Espinoza M, Montiel-Ponsoda E, Gómez-Pérez A (2009) Ontology Iocalization. In: Proceedings of the fifth international conference on Knowledge Capture (KCAP 2009), Redondo Beach, CA, USA, pp 33–40. ISBN: 978-1-60558-658-8

Fernández-López M (1999) Overview of methodologies for building ontologies. In: Proceedings of the IJCAI-99 workshop on ontologies and problem-solving methods: lessons learned and future trend, Stockholm, Sweden, August 1999. http://oa.upm.es/5480/

Gangemi A (2005) Ontology design patterns for semantic web content. In: Musen M et al (eds) Proceedings of the fourth international semantic web conference, Galway, Ireland. Springer, Berlin

Gangemi A (2007) Design patterns for legal ontology construction. In: Casanovas P, Noriega P, Bourcier D, Galindo F (eds) Trends in legal knowledge: the semantic web and the regulation of electronic social systems. European Press Academic Publishing, Florence

Gómez-Pérez A, Fernández-López M, Corcho O (2003) Ontological engineering. Advanced information and knowledge processing series. Springer, Heidelberg, ISBN 1-85233-551-3

IEEE Standard Glossary of Software Engineering Terminology (1990) IEEE Std. 610.12–1990 (Revision and redesignation of IEEE Std. 792–1983)

Jimeno-Yepes A, Jimenez-Ruiz E, Berlanga-Llavori R, Rebholz-Schuhmann D (2009) Reuse of terminological resources for efficient ontological engineering in life sciences. BMC Bioinformatics 10:S4. ISSN: 1471–2105

Lamsfus C, Alzua-Sorzabal A, Martin D, Salvador Z, Usandizaga A (2009) Human-centric ontology-based context modelling in tourism. In: Proceedings of KEOD 2009 Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Funchal - Madeira, Portugal, October 6–8, 2009. INSTICC Press 2009, ISBN 978-989-674-012-2, pp 424–434

Newell A (1982) The knowledge level. Artif Intell 18(1):87–127

Pan JZ, Lancieri L, Maynard D, Gandon F, Cuel R, Leger A (2007) Knowledge web deliverable D1.4.2.v2. Success stories and best practices. Available at http://knowledgeweb.semanticweb.org/semanticportal/deliverables/D1.4.2v2.pdf

Pease RA, Niles I, Li J (2002) The suggested upper merged ontology: a large ontology for the semantic web and its applications. In: Workshop on ontologies and the semantic web at the AAAI 2002, Edmonton

Poveda-Villalón M, Suárez-Figueroa MC, García-Castro R, Gómez-Pérez A (2010) A context ontology for mobile environments. In: Workshop on Context, Information and Ontologies (CIAO 2010) co-located with EKAW 2010, Lisbon

Presutti V, Gangemi A (2008) Content ontology design patterns as practical building blocks for web ontologies. In: Proceedings of the 27th international conference on conceptual modeling (ER2008), Barcelona, Spain

Simperl E (2009) Reusing ontologies on the Semantic Web: A feasibility study. Data Knowledge Engineering 68(10):905–925

Suárez-Figueroa MC (2010) NeOn Methodology for building ontology networks: specification, scheduling and reuse. PhD thesis, Universidad Politécnica de Madrid, España. Available at http://oa.upm.es/3879/

Vilches-Blázquez LM, Villazón-Terrazas B, Saquicela V, de Leon A, Corcho O, Gómez-Pérez A (2010) GeoLinked data and INSPIRE through an application case. In: 18th ACM SIGSPATIAL international conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2010), San Jose, CA, 2–5 Nov 2010

Villazón-Terrazas B, Suárez-Figueroa MC, Gómez-Pérez A (2010) A pattern-based method for re-engineering non-ontological resources into ontologies. Int J Semant Web Inf Syst 6(4):27–63

Villazón-Terrazas B, Ramírez J, Suárez-Figueroa MC, Gómez-Pérez A (2011) A network of ontology networks for building e-employment advanced systems. Expert Syst Appl 38(11):13612–13624

Chapter 3
Pattern-Based Ontology Design

Valentina Presutti, Eva Blomqvist, Enrico Daga, and Aldo Gangemi

Abstract In this chapter, we present ontology design patterns (ODPs), which are reusable modeling solutions that encode modeling best practices. ODPs are the main tool for performing pattern-based design of ontologies, which is an approach to ontology development that emphasizes reuse and promotes the development of a common "language" for sharing knowledge about ontology design best practices. We put specific focus on content ODPs (CPs) and show how they can be used within a particular methodology. CPs are domain-dependent patterns, the requirements of which are expressed by means of competency questions, contextual statements, and reasoning requirements. The eXtreme Design (XD) methodology is an iterative and incremental process, which is characterized by a test-driven and collaborative development approach. In this chapter, we exemplify the XD methodology for the specific case of CP reuse. The XD methodology is also supported by a set of software components named XD Tools, compatible with the NeOn Toolkit, which assist users in the process of pattern-based design.

3.1 Introduction

One of the most challenging and neglected areas of ontology design is reusability, which is getting more and more important partly due to the increased spread of the Linked Data concept (Bizer et al. 2009). The notion of "pattern" has proved useful in design, as exemplified in diverse areas, such as software engineering (Gamma et al. 1994). In this chapter, we introduce the notion of ontology design patterns (ODPs) along with a description of their different types and characteristics.

V. Presutti (✉) • E. Blomqvist • E. Daga • A. Gangemi
Semantic Technologies Lab, Institute of Cognitive Sciences and Technologies (National Research Council – CNR), Via Nomentana 56, 00161 Rome, Italy
e-mail: valentina.presutti@cnr.it; eva.blomqvist@istc.cnr.it; enrico.daga@cnr.it; aldo.gangemi@cnr.it

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_3, © Springer-Verlag Berlin Heidelberg 2012

Then we focus on content ODPs (CPs), which are domain-dependent practices of modeling, encoded as reusable computational components.

ODPs have recently been the subject of a series of workshops (Blomqvist et al. 2009b, 2010b), and they are collected in online repositories such as the ODP portal[1]. Section 3.2 defines and describes ODPs, their types and characteristics, while in Sect. 3.3, we describe how CPs can be reused by means of a set of operations, such as import, specialization, and composition. In the second part of the chapter, i.e., Sect. 3.4, we introduce a pattern-based ontology design approach and describe a particular iterative and incremental method named eXtreme Design (XD), supporting this practice with a collaborative and test-driven approach. At the end of Sect. 3.4, we show a set of tools that provide software support to XD in the NeOn Toolkit environment, before we summarize some conclusions in Sect. 3.5.

3.2 What Are Ontology Design Patterns (ODPs)?

During the past decade, as remarked by (Gangemi and Presutti 2009), an average user that is trying to build or reuse an ontology, or an existing knowledge resource, has typically been left with very limited assistance in using unfriendly logical structures, some large, hardly comprehensible ontologies, and a bunch of good practices that must be discovered from the literature. A typical usage scenario includes, for instance, a large set of web ontologies that are evaluated (usually in an implicit way, e.g., by inspecting them) against the intended domain and tasks of the ontology that is needed. The selected ontology (if any) is reused, and then an adaptation process is started in order to cope with the implicit requirements underlying the ontology project that originally created the reused ontology[2]. This scenario is costly in many cases. As noted by (Rector and Stevens 2008), usability of large OWL ontologies from a human perspective is often low, and automatic selection mechanisms do not help with the adaptation process.

Another typical scenario includes so-called "reference" or "core" ontologies that are supposed to be directly reused and specialized. Unfortunately, even if well designed, they are usually large and cover more knowledge than what a designer might need. In this case, it is hard to reuse only the "useful pieces" of the ontology, and consequently, the cost of reuse can be higher than developing a new ontology from scratch. On the other hand, the success of very simple and small ontologies, such as FOAF[3] and SKOS (Miles and Bechhofer 2009), shows the potential of

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

[2] Even in cases when ontology requirements are explicitly expressed, e.g., as described in Chap. 5, there are commonly other implicit domain assumptions that need to be addressed at reuse time. In our experience, it is also quite rare that explicit requirements are distributed together with their corresponding ontology.

[3] See the FOAF project website: http://www.foaf-project.org/

really portable or "sustainable" ontologies. These lessons learned support a new approach to ontology design, which is sketched here.

Under the assumption that there exist classes of problems that can be solved by applying common solutions (as has been experienced in software engineering), it is suggested to support reusability on the design side specifically. We need a way to express commonly applicable solutions and "best practices" and what ontological requirements they solve (see Chap. 5); this is where ODPs come into play. An ODP is a modeling solution to a recurrent ontology design problem (Gangemi and Presutti 2009). However, with the term ODP we refer to a wide range of modeling solution types. ODPs can be grouped into six types, or families, each addressing different kinds of modeling problems:

Structural ODPs include *Logical ODPs* and *Architectural ODPs*. *Logical ODPs* are compositions of logical constructs that solve a problem of expressivity. They help solving design problems when the used representation language does not directly support certain logical constructs, such as representing *n*-ary relations in OWL (Noy and Rector 2004). *Architectural ODPs* are defined in terms of compositions of *Logical ODPs* and affect the overall shape of the ontology, e.g., a certain OWL 2 profile could be viewed as an *Architectural ODP*.

Correspondence ODPs include *Re-engineering ODPs* and *Alignment ODPs*. *Re-engineering ODPs* provide designers with solutions to the problem of transforming a conceptual model, which can be either an ontology or a non-ontological resource[4]. to an ontology, e.g., transforming an OWL ontology in order to make it comply with a certain vocabulary, transforming a classification scheme to an OWL ontology, and so on. *Alignment ODPs* are patterns for creating semantic associations between two existing ontologies. They provide designers with solutions to align two ontologies without changing the logical types of the ontology entities involved, e.g., relating two ontologies both defining the concept "author," one by a class and the other by a property (Scharffe and Fensel 2008).

Reasoning ODPs are procedures that perform automatic inference. Examples of *Reasoning ODPs* are so-called normalizations (Vrandečić and Sure 2007). Other *Reasoning ODPs* include common reasoning tasks, such as *classification*, *subsumption*, *inheritance*, *materialization*, *de-anonymizing*, etc.

Presentation ODPs deal with usability and readability of ontologies from a human perspective. They are best practices facilitating ontology evaluation and selection, hence supporting reuse. Examples of *Presentation ODPs* are so-called *Naming ODPs,* which identify best practices for naming, i.e., naming conventions (Svátek et al. 2009).

Lexico-Syntactic ODPs are linguistic structures consisting of a sequence of types of words associated with an assessment of the meaning they express (Aguado de Cea et al. 2009). For example, the sequence of two noun phrases connected by

[4] For further details, and a definition of "non-ontological resource", see Chap. 6 of this book.

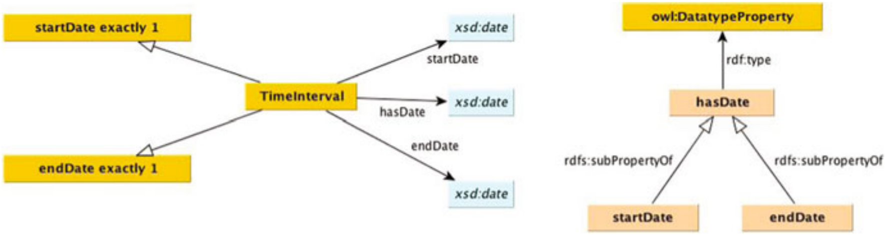


Fig. 3.1 UML-like diagram showing the OWL encoding of the *time interval* CP taken from the online catalog of CPs (http://www.ontologydesignpatterns.org/wiki/Submissions:TimeInterval)

the verb *be* such as "Dolphins *are* warm blooded mammals" often identifies a "subClassOf" relation between a class that represents dolphins and a class that represents warm-blooded mammals.

Content (or domain) ODPs (CPs) are instantiations (and compositions) of *Logical ODPs*. They have an explicit non-logical vocabulary for a specific domain of interest, i.e., they are content (domain) dependent, although the domain might be very general. An example of a Content ODP is depicted in Fig. 3.1. It represents the concept of a "time interval" as a class of things characterized by an arbitrary number of "dates" (i.e., points in time), but which has exactly one start date and one end date.

Much more important than the type of a pattern is its nature of being a reusable modeling solution. Ideally, an ontology project can be completely developed by reusing existing solutions, i.e., ODPs, by appropriately combining them – however, this ideal situation will most likely be very rare in practice, whereby we need to combine the approach presented here with the ones targeted in other chapters of this book. An interesting question is nevertheless: How can we make ODP reuse as easy and useful as possible? In order to identify candidate ODPs for reuse in a certain ontology project, ODPs and the specific ontology design problems to be addressed have to be comparable, i.e., need to be described in a similar way. For instance, we need to know what requirements a certain ODP helps us to solve, as well as what requirements are present in our current ontology design project.

Figure 3.2 depicts the idea of pattern-based design. The ontology development project is divided into two spaces: (1) the problem space, which contains a set of requirements (explicitly represented, e.g., as competency questions (CQs), as discussed in Chap. 5) that describe the ontology design problems to be addressed, and (2) the solution space, which contains all available ODPs, where each ODP should be well documented, e.g., described through what ontological requirements it solves. First, the requirements of the current project (problem space) are compared with the requirements of the available ODPs (solution space), and a set of candidate matching ODPs are identified. Second, the most appropriate ODPs among the candidate ones are selected for reuse, as illustrated by "dropping them into the project basket" in Fig. 3.2.

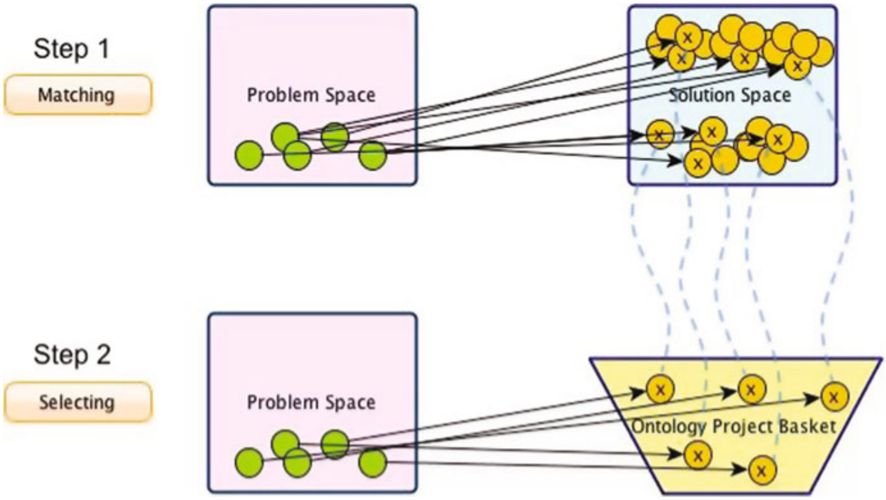


Fig. 3.2 The idea of pattern-based design. The ontology project is divided into the *problem* and *solution spaces*. The *problem space* contains a set of requirements (see Chap. 5), while the *solution space* contains a set of ODPs. The two spaces are compared in order to identify ODPs matching the requirements (matches are illustrated by *arrows*). A number of those ODPs are selected for reuse in the ontology project, i.e., dropped in the project basket

One of the most important raison d'être of ODPs is to enable this matching and selection task for supporting reuse. Hence, regardless of its type, an ODP is associated with a set of requirements, explicitly represented, which describes the problem it provides a solution for. For example, the requirements associated with the CP shown in Fig. 3.1 refer to the problem of representing time intervals, their start and end points. If expressed in the form of CQs, the requirements of that CP include "What is the starting point of a particular interval?" and "What is the end point of a particular interval?". Another example is the Logical ODP for expressing "*n*-ary relations," the requirements of which indicate the issue of representing relations with *n* arguments through a logical language including only primitives for expressing binary relations, e.g., a language such as OWL. The format and extent of formality of requirement representation depends on the type of ODP. In this chapter, we will focus on content ODPs (CPs) and how to use them for developing ontologies through the application of a particular pattern-based method, as well as its specific tool support.

3.3 Content Ontology Design Patterns (CPs)

CPs solve design problems for the domain classes and properties that populate an ontology; therefore, they solve content – domain-specific – problems (Gangemi and Presutti 2009). According to the general notion of ODP (see Sect. 3.2), each CP is associated with a set of requirements, which represents the problem it provides a solution for. Such requirements are usually represented in three different forms: (1) competency questions (CQs), i.e., based on work by (Gruninger and Fox 1994), (2) contextual statements, i.e., general axioms that hold within the domain, and (3) reasoning requirements. CQs indicate typical queries that a knowledge base will be able to answer if it is based on that CP. Contextual statements are general axioms that apply within the domain, which indicate conditions that hold for (and between) certain concepts encoded by the CP. Finally, reasoning requirements indicate what inferences are enabled by the CP, e.g., if it perhaps allows some form of classification or consistency checking of facts.

The *time interval* CP shown earlier, in Fig. 3.1 (Sect. 3.2), which is a very simple but useful CP, is associated with the following competency questions:

- *When does a certain time interval start?*
- *When does a certain time interval end?*
- *What are the points in time that belong to a certain time interval?*

Typically, a CP can also be associated with a set of SPARQL queries that formally encode its competency questions. This is very useful for an ontology designer who wants to test an ontology containing a CP against sample data since the SPARQL queries can be used to test the coverage of the CQs.

The *time interval* CP (see Fig. 3.1, Sect. 3.2) is also associated with the following contextual statement:

- *A time interval always has exactly one starting point and exactly one end point.*

This requirement is, in the OWL realization of the CP, addressed by cardinality restrictions on the data type properties that identify the start and the end of a time interval.

Finally, this CP is also associated with the following reasoning requirements:

- *The start and end dates of a time interval belong to the interval.*
- *Two time intervals with the same start and end dates should be recognized to be the same interval.*

The first reasoning requirement is, in the OWL realization of the CP, addressed by defining the two data type properties `startDate` and `endDate` as subproperties of `hasDate`, which is the property indicating that something belongs to the interval. This enables ontologies reusing the CP to also include the start and end date of an interval, when the model is queried for dates belonging to the interval, using the `hasDate` property. The second reasoning requirement is addressed by defining a `hasKey[startDate, endDate]` axiom, on the class `TimeInterval`. This enables an inference engine to infer that the `owl:sameAs` property holds between two instances of `TimeInterval` whenever they have the same start and end date values.

Where do CPs come from? This is a highly relevant question since we need a considerable catalog of CPs in order for them to be useful in practice. A CP can emerge from existing conceptual models as well as from data. It can be extracted from *foundational* (Masolo et al. 2005), *core* (Gangemi and Borgo 2004), or *domain* ontologies, re-engineered from other conceptual models (e.g., data model patterns (Hay 2000)). Informally, the distinction between foundational, core, and domain ontologies relates to the generality of the domain they address and to the extent of domain coverage: (1) *foundational ontologies*, e.g., DOLCE[5] and SUMO (Niles and Pease 2001), axiomatize general concepts and relations and are reusable across any domain; (2) *core ontologies* (Masolo et al. 2005), such as the Core Ontology of Fishery (Gangemi et al. 2004) and the Core Legal Ontology (Gangemi et al. 2005), focus on a specific domain without being restricted to specific applications or specific sub-areas. The latter can be built as extensions of foundational ontologies or based on general principles and well-founded methodologies; and (3) *domain ontologies*, such as the Gene ontology[6] and the Unified Medical Language System (UMLS)[7], deal extensively with a specific domain of interest, deepen the coverage of a certain area of a domain or address a specific use case within a domain. Informally, such general ontologies can be viewed as compositions of numerous CPs, and by modularizing such ontologies, i.e., decoupling certain "pieces" from the rest of their often large overall structure, the formal representation of those CPs can be extracted.

CPs can also be extracted from Linked Data (Bizer et al. 2009), i.e., in a more bottom-up fashion, where they emerge from the way data is actually modeled. By analyzing recurring semantic structures (if any) within the same, as well as across different, datasets addressing some domain of interest, CPs may be detected.

CPs can also be created by composing or specializing other CPs or by expanding them (see Sect. 3.3.1 describing operations on CPs). Figure 3.3 shows a composed CP. This CP allows to represent the relation between an information object, such as a novel, and its realizations, e.g., a book, an HTML page, etc., and to index this relation based on the place *where* and time *when* it holds. This CP reuses the *time interval* CP shown in Fig. 3.1 (Sect. 3.2) and combines it with two other CPs, the *information realization* CP[8] and the *place* CP[9]. The composition is realized by specializing a fourth CP, the *situation* CP[10]. which is a CP corresponding to the *n-ary relation* Logical ODP.

Since CPs can be represented as reusable building blocks, e.g., OWL modules, a natural question is how they are distinguished from any other small ontology. CPs show a number of pragmatic characteristics that allow to distinguish them from other ontologies. CPs are:

[5] DOLCE – Project Home Page: http://dolce.semanticweb.org

[6] See http://www.geneontology.org/

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

[8] http://ontologydesignpatterns.org/wiki/Submissions:Information_realization

[9] http://ontologydesignpatterns.org/wiki/Submissions:Place

[10] http://ontologydesignpatterns.org/wiki/Submissions:Situation

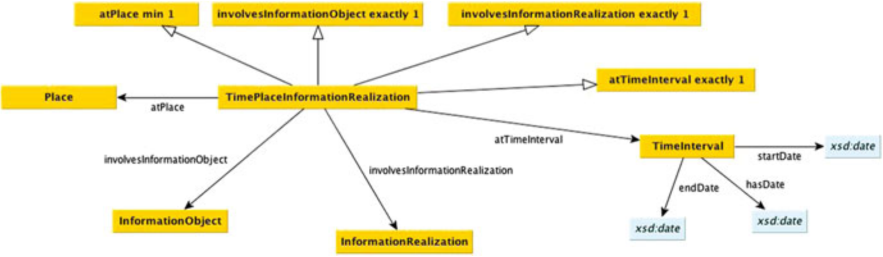

atPlace min 1
involvesInformationObject exactly 1
involvesInformationRealization exactly 1
atTimeInterval exactly 1
Place
atPlace
TimePlaceInformationRealization
atTimeInterval
TimeInterval
xsd:date
startDate
hasDate
endDate
xsd:date
xsd:date
involvesInformationObject
involvesInformationRealization
InformationObject
InformationRealization

Fig. 3.3 UML-like diagram showing the OWL encoding of the *time and place indexed information realization* CP present in the online catalog of CPs

- *Computational components*. They are represented and encoded in a computational logic language, e.g., OWL, so that they can be processed and reused as building blocks in ontology design, e.g., through the OWL import construct.
- *Small, autonomous components*. Smallness and autonomy of CPs facilitate the design of ontology networks, because they enforce modularization; by composing CPs, designers can better govern the complexity of the whole resulting ontology as opposed to governing a monolithic ontology.
- *Inference enabling components*. Each CP allows some logical conclusions to be drawn from the model. This means that a single element, e.g., a single class without any associated axioms, cannot be a CP since it does not enable any inferences (even simple ones) to be made.
- *Hierarchical components*. All CPs participate in a partial order, where the ordering relation is called *specialization* (see Sect. 3.3.1). Specialization requires that at least one entity of the more specific pattern, e.g., a class or property, is subsumed by at least one entity of another, more general, pattern. Figure 3.4 shows an example of CP specialization: (a) shows the *part-of* CP[11], which defines a transitive property between objects for representing parthood relationships between them; (b) shows the *componency* CP[12], which specializes *part-of* by defining object properties for representing direct parts of objects as sub-properties of the transitive parthood relation.
- *Cognitively relevant components*. CP visualization must be intuitive and compact and should catch relevant, "core" notions of a domain[13].
- *Best practices* of ontology modeling. How to evaluate the quality of a CP, e.g., to determine if it is truly a *best practice* is currently an open issue (Hammar and Sandkuhl 2010); hence at the moment, the quality of a CP can only be assessed through the personal experience of ontology designers and through its provenance. Additional criteria are evidence from reusability across different projects and large-scale applications such as Linked Data.

[11] http://ontologydesignpatterns.org/wiki/Submissions:PartOf

[12] http://ontologydesignpatterns.org/wiki/Submissions:Componency

[13] Independently of the generality at which a CP is singled out, it must contain the central notions that "make rational thinking move" for an expert in a given domain for a given task.

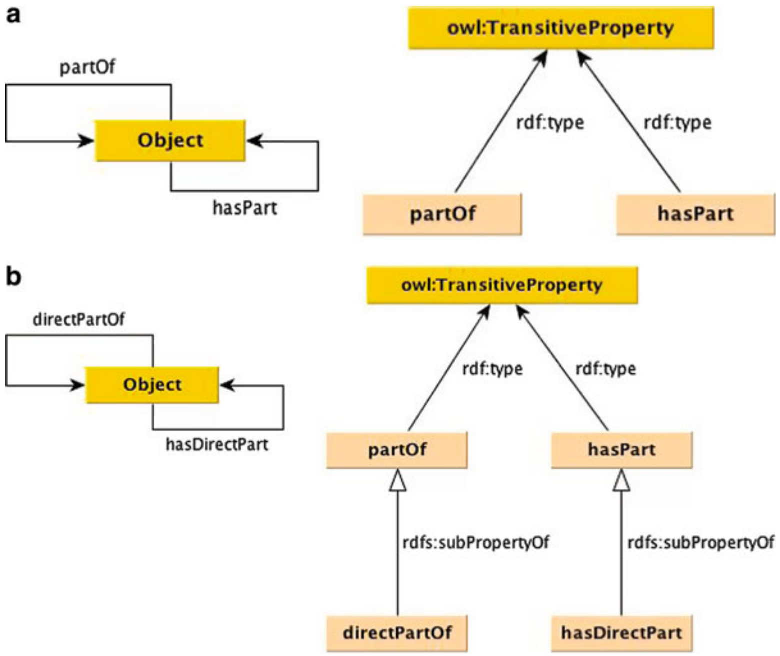


Fig. 3.4 Example of CP specialization: (**a**) depicts the *part-of* CP, which is specialized by the *componency* CP, shown in (**b**)

Additionally, CPs often match linguistic structures called frames. This could be formulated as an additional characteristic of being *linguistically relevant*, and the essence of most CPs can be expressed quite straightforward in natural language. The richest repository of frames is FrameNet (Baker et al. 1998). Informally, a frame is a lexically founded ontology design pattern. Frames typically encode argument structures for verbs, e.g., the frame *Desiring* defines associations between elements (or "semantic roles") such as `Experiencer`, `Event`, `FocalParticipant`, `LocationOfEvent`, etc. Frames can be used for validating CPs with respect to lexical coverage, for lexicalizing them, and can be re-engineered in order to populate a CP catalog such as the ODP portal.

As opposed to the concept of CP, there is that of *AntiCP*. AntiCPs are ontologies that implement wrong modeling practices, e.g., examples of bad practices or common mistakes. In other words, they are based on erroneous assumptions or rationales. For example, modeling transitive parthood relationships through subsumption, e.g., `City rdfs:subClassOf Country`, is considered an AntiCP. AntiCPs produce the side effect of inferring wrong or undesired knowledge, e.g., `Rome rdf:type Country`, or of preventing the capability to infer the desired knowledge. It is important to distinguish between ontologies that are not CPs and AntiCPs, i.e., only a subset of ontologies that are *not* CPs are *AntiCPs*.

3.3.1 Operations on Content Patterns

CPs are a special kind of ontologies, as discussed above, and their creation and usage rely on a set of operations that can be summarized as follows:

Cloning consists of duplicating an ontology entity (possibly into a new namespace), which can be reused in a CP or used as a prototype for the definition of a new ontology entity defined in a CP. This operation is, for instance, used when extracting CPs from foundational and core ontologies, i.e., a part of the larger structure becomes a CP through being cloned and given a new namespace.

Composition relates two CPs and results in a new ontology (which could in turn be a CP – as seen in Fig. 3.3, Sect. 3.3). The resulting ontology includes the union of the sets of ontology entities and axioms from the two CPs plus the ontology entities and axioms that are defined locally in the new ontology in order to relate the two CPs, e.g., disjointness axioms. Figure 3.5 depicts an example of a CP composition. At the left of the figure, the two CPs are shown separately, one CP (top left) represents membership relationships between persons and music bands, the other CP (bottom left) models objects and the roles they play. The axioms that are added for composing the two CPs are shown at the right side of the figure. The class `Person` is defined to be subclass of `Object`, and both `Person` and `Band` are defined to be disjoint with `Role`.

Specialization defines a new ontology (which could in turn be a CP) by specializing entities of an existing one, e.g., a CP. Specialization introduces a partial order between CPs, based on subsumption relations holding between their respective ontological entities. Specialization relies on `rdfs:subClassOf` and `owl:subPropertyOf` constructs. Figure 3.4 in Sect. 3.3, as mentioned previously, shows an example of specialization: the *component* CP (Fig. 3.4b) specializes *part-of* (Fig. 3.4a) by defining the object properties `directPart` and `directPartOf` as sub-properties of `hasPart` and `partOf`, respectively.

Import is the basic mechanism for explicit CP reuse, as well as a way to reuse ontologies in general. It is also the only operation described here that is directly

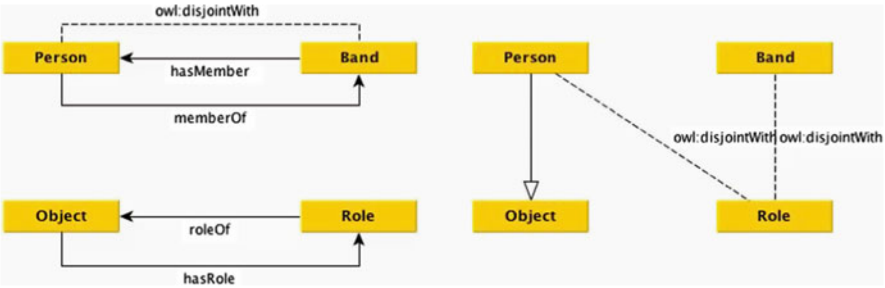


Fig. 3.5 Example of CP composition: A CP representing membership relationships between persons and bands (*top left*) is composed with the CP *object-role* (*bottom left*), which represents objects and the roles they play, and the result can be seen to the *right*

supported in the OWL vocabulary, through `owl:imports`. By importing a CP, the ontology includes all the axioms of the CP and hence ensures the set of inferences that the CP enables in its corresponding knowledge base.

3.4 eXtreme Design: An Agile Methodology for Pattern-Based Ontology Development

With the name *eXtreme Design* (XD), we refer to a family of methods that support the pattern-based design approach as depicted in Fig. 3.2 (Sect. 3.2), i.e., the matching between problem and solution spaces in order to reuse solution components, such as ODPs (Presutti et al. 2009). When XD is based on CPs, the problem space is expressed by means of *competency questions* (CQs), *contextual statements,* and *reasoning requirements*, as described at the beginning of Sect. 3.3, and the solution space contains CPs and their associated requirements, i.e., similarly expressed through CQs, contextual statements, and reasoning requirements. Hence, the matching process is performed through finding similarities between CQs as well as between the other two types of requirements, as exemplified in Fig. 3.6.

In the following sections, we describe the XD method, inspired by software engineering's *eXtreme Programming* (XP) (Shore and Warden 2007) and *experience factory* (Basili et al. 1994), for building ontologies through intensive CP reuse. XD is test-driven, i.e., testing is a central part of the development; it applies a divide-and-conquer approach similarly to XP and promotes pair design, which is analogous to pair programming.

3.4.1 eXtreme Design Principles

Similarly to XP, XD has evolved around a set of main principles. The principles both describe the essence of XD and act as guidelines when performing the design process.

The first principle is named *customer involvement and feedback*. Ideally, the customer should be involved in the ontology development team continuously.

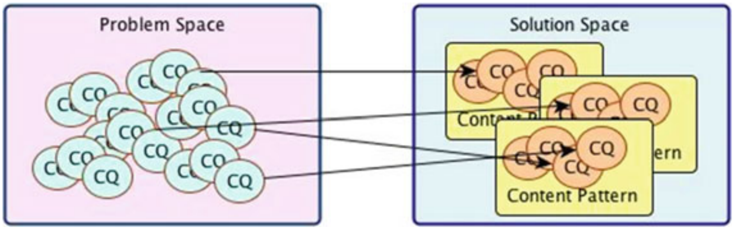


Fig. 3.6 The XD (eXtreme Design) approach with CPs, exemplified for CQ matching

This means that the customer should identify representatives that can be easily contacted during the development for quick feedback. Such representatives have to be aware of all parts, and needs, of the project. Here, depending on the project configuration, the "customer" could be either the organization containing the end users of the system to be built (including the ontology) or simply the software developers needing the ontology in order to perform some particular functionality in the overall system.

The second principle states that all requirements should be based on *customer stories*, from which CQs, contextual statements, and reasoning requirements are derived. The customer representatives describe the ontology requirements and the ontology tasks in terms of small stories. Designers work on those small stories and transform them into more rigorous and precise requirements, e.g., in the form of CQs, contextual statements, and reasoning requirements.

Next, an important principle is that of *iterative development*. XD is an iterative and incremental process. Each iteration produces a number of modules that contribute to an incremental release, produced through an integration phase.

Test-driven design means, in the case of XD, that testing is used as an integrated means for completing the modules. Stories, CQs, reasoning requirements, and contextual statements are used in order to develop unit tests, e.g., CQs can be transformed into SPARQL queries. By deciding how the query should be formed, a developer is actually partly designing the model, hence, the notion "test-driven" design. The ontology module representing a customer story can be passed to the integration phase, i.e., to be included in the next release, only if all its associated unit tests run successfully. This principle also enforces the *task-oriented* approach of the method, i.e., the principle that modules should realize exactly what is required (their intended task), nothing more and nothing less.

It has to be noted that ontology unit testing, first introduced by (Vrandečić and Gangemi 2006), has a different meaning than software unit testing. An ontology module developed for addressing (part of) a user story is tested by developing unit tests, i.e., dedicated ontology modules containing sample facts and appropriately documented with testing metadata, each importing the ontology module to be tested, based on one of the following three approaches: (1) through *verification tests* to test the fulfillment of basic requirements, i.e., SPARQL queries based on CQs that are run against valid sample data in order to check if expected results are returned by the SPARQL engine, (2) *inference tests*, i.e., through inference materialization performed on sample data which is expected to cause certain inferences to be materialized in accordance with the reasoning requirements and (3) through *stress tests*, e.g., through consistency checking performed on invalid sample data violating the contextual statements, thus expecting to provoke inconsistencies. While (1) and (2) are mainly intended for verifying the correct implementation of requirements, (3) could be viewed as more similar to the kind of software testing when a system is fed random or erroneous data, to make sure such cases are handled correctly, and there are no unexpected side effects or crashes.

One of the core principles of XD is *ODP reuse*, which inherently leads to a *modular design*. Iterations are based on identifying reusable CPs through matching

CQs and other requirements. Every time there is a positive match, the identified CPs are considered for reuse. If the solution space does not provide an adequate ready-to-use CP, a specific solution is developed in a modular way, preferably in the form of a new CP so that it can be shared with the team (and ideally on the web) for future reuse. This principle favors the creation of a common "language" based on shared patterns and eases both the understandability and the integration of developed modules. In addition, the *divide-and-conquer* paradigm leads to a natural modularisation of the problem, which facilitates a distributed ontology development, and assists in scoping the modeling issues that are addressed within a single iteration.

To handle this incremental ontology development, *collaboration* and *integration* are two essential principles. Integration is a key aspect of XD, as the ontology is developed in a distributed, modular, and collaborative way. Collaboration and continuous sharing of knowledge is needed when running an XD project. The result of each iteration, i.e., one or more ontology modules, is integrated with the rest of the ontology modules before releasing an increment. Typically, a sub-team of designers is devoted to the integration task.

As mentioned previously, *task-oriented design* is another main principle. The XD approach is based on developing a task-oriented ontology, covering only part of a domain of knowledge according to a specific application task. This is opposed to the more philosophically oriented approach of formal ontology design, where the aim is to comprehensively cover a certain domain of knowledge. XD proposes to provide solutions to the exact requirements stated, in the sense that the concepts should be defined according to the intended task of the ontology, rather than in some common sense notion of their "true" nature. Each XD iteration focuses on a specific part of the domain requirements, expressed in terms of a user story.

On the more organizational side, XD promotes *pair design*. The team of designers is organized in pairs. This practice is analogous to the pair programming of XP. While pair programming has empirically been proven efficient in software development, it still remains to rigorously test the efficiency for ontology engineering. Currently, this has to be considered a hypothesis, based on experience and observations made through collecting feedback of trainees and developers, through informal discussions and questionnaires after the execution of XD with different teams. Most of them felt that they benefit from on-the-fly brainstorming, and perceived to improve the effect of learning-by-doing within the pair design setting.

3.4.2 The eXtreme Design Process

Figure 3.7 shows the complete XD process for CP reuse. The process starts with the XD team, including the customer representatives, making themselves familiar with the knowledge domain, with the aim of identifying the scope of the ontology project, based on the desired application tasks (*Step 1. Project initiation and scoping*). The objective is twofold: (1) to make the customer representatives (domain experts) aware of what the XD team expects from them and (2) to provide the ontology design team with an overview of the problem from a domain expert perspective, its scope, and agree on initial terminology. The result of this step is the setup of a collaborative environment where the customer representatives and ontology designers will share documentation and collect argumentation and motivations of modeling issues, including terminology, e.g., through deploying a wiki for the project. Following this starting activity, the XD team identifies the sources of CPs to be used during the ontology project (*Step 2. Identify CP catalogs*); however, such a set can be extended during the process.

The customer representatives are then invited to write stories, preferably from real, documented scenarios, which act as samples of the typical facts that should be stored in the resulting ontology, and exemplify how these facts are connected and used (*Step 3. Collecting requirement stories*). All stories are organized in terms of priority, and possible dependencies between them are identified and made explicit. Each story is described by means of a small card, like the one depicted in Table 3.1, which includes the unique title of the story, a list of other stories that it depends on, a description in natural language, i.e., the story itself, and a priority value. The customer can add stories during the whole project life cycle, depending on how formal contracts and other commitments are formulated. Nevertheless, if a new requirement emerges, new stories can be written.

Once a sufficient number of stories for starting the development have been collected, each pair of designers selects a story that will be the focus of their work for the next iteration (*Step 4. Eliciting requirements and constructing module(s) from CPs*). The selection is based on the experience and competencies of the design pair and on the priority of the story. A new wiki page for the story is created, and its content is set up based on the information reported on the card. By performing this task, a pair enters a development iteration (the dashed rectangle in Fig. 3.7, which is detailed in Fig. 3.8).

Once a story has been completely modeled, it is carefully documented and released internally for integration into the next release (*Step 5. Releasing module(s)*). This task constitutes the end of a story iteration for a pair, and the result is one or more ontology modules, i.e., small ontologies. Before releasing a module, it is important to make sure that all tests run successfully and that the module is well documented, both in the shared wiki as well as through annotations in the module itself. All ontological elements have to have appropriate labels, they have to be commented (as well as the module itself), and the module should be associated with a description of its purpose, the requirements it solves, and even links to the unit

Table 3.1 A requirement story card, here exemplified through a story from the fishery domain

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

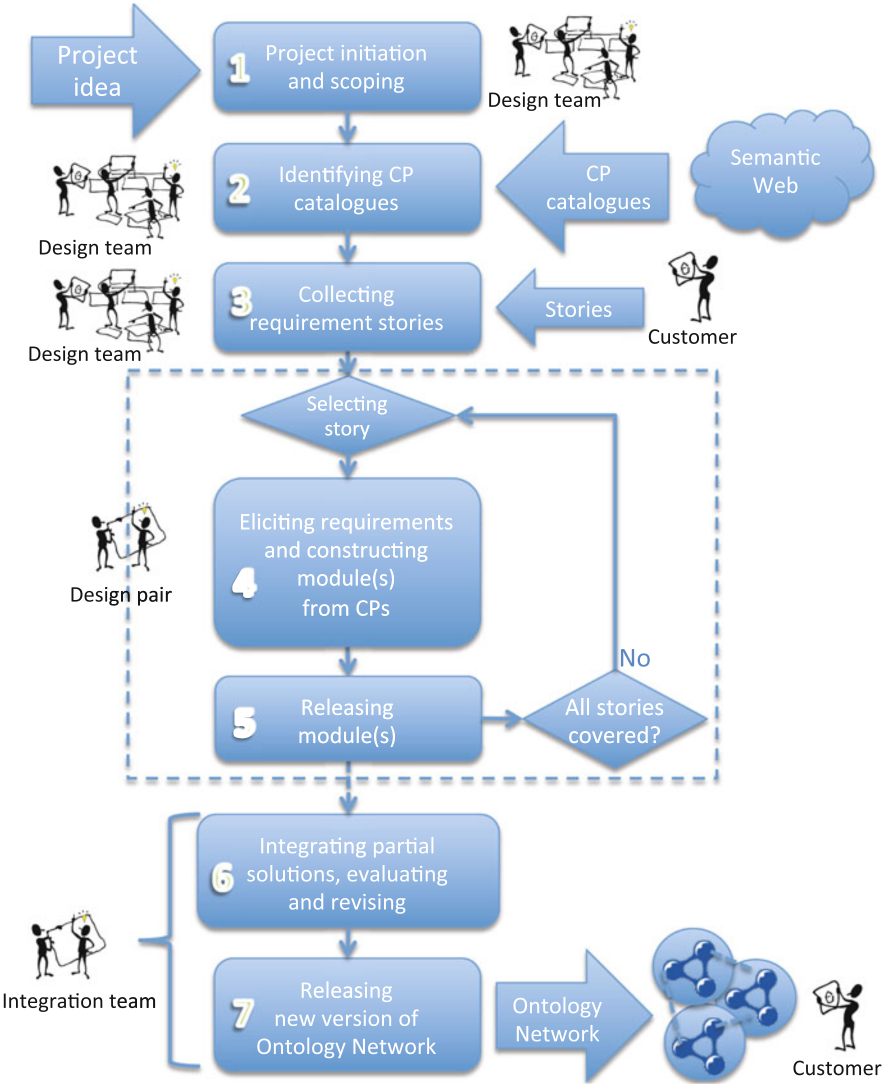

Project idea
1 Project initiation and scoping
Design team
2 Identifying CP catalogues
Design team
CP catalogues
Semantic Web
3 Collecting requirement stories
Design team
Stories
Customer
Selecting story
Design pair
4 Eliciting requirements and constructing module(s) from CPs
No
5 Releasing module(s)
All stories covered?
6 Integrating partial solutions, evaluating and revising
Integration team
7 Releasing new version of Ontology Network
Ontology Network
Customer

Fig. 3.7 The overall XD process, for CP reuse

tests that have been used. The modules are assigned a URI and are shared by the whole team. If a module can be publicly shared, and is considered highly reusable, it can be published in open web registries, such as the ODP portal.

Once a new module is released, it has to be integrated with all the others that constitute the current version of the ontology network (*Step 6. Integrating partial solutions, evaluating, and revising*). Usually, one pair is in charge of performing the integration and related tests. New unit tests are defined for the integrated ontology network, and all existing ones (unit tests of individual modules) are again executed

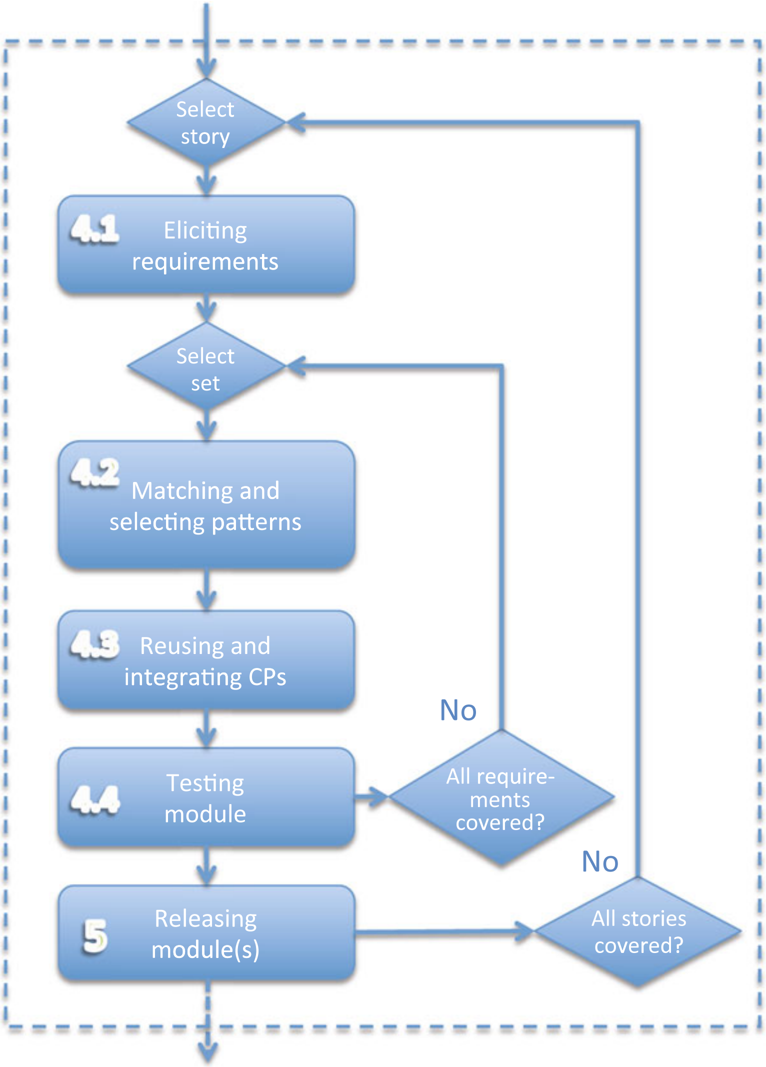

Select story
4.1 Eliciting requirements
Select set
4.2 Matching and selecting patterns
4.3 Reusing and integrating CPs
4.4 Testing module
No
All require-ments covered?
No
5 Releasing module(s)
All stories covered?

Fig. 3.8 Detailed breakdown of sub-steps in the design pair iteration (steps 4–5 in Fig. 3.7)

as regression tests before moving to next task. All contextual statements and reasoning requirements are taken into account, and all necessary alignment axioms are defined. The modules are now under the complete control and editing of the team in charge of the integration, and refactoring of the ontology modules may be performed in case inconsistent modeling choices are discovered. Integration can be done in multiple fashions, and an integration policy should be defined at the start of the project. For instance, if decoupling of modules is an essential feature of the resulting ontology network, then a minimum of refactoring should be performed in order to remove overlap between modules, instead integration should simply align the modules. On the other hand, in some cases, a coherent and non-redundant model is desired, whereby an alternative policy would be to refactor the modules, remove as many redundant definitions as possible and instead add import dependencies between them. The products of this step are new unit tests and alignment axioms, and possibly a set of changes to the ontology modules (results of refactoring), all properly documented in the wiki.

When all unit tests run successfully during the integration step, a new incremental version of the ontology network can be released (*Step 7. Releasing new version of ontology network*). The ontology is given a new version number, it is appropriately documented, and it is associated with its own version of the wiki documentation. It is important to note that the process depicted in Fig. 3.7 is usually not a sequential one, i.e., in most cases the arrows indicate an input/output dependency rather than a sequence of actions. For instance, the integration and release steps will be ongoing activities during the complete project (as soon as the first modules are ready); hence, integration will be performed in parallel with Steps 4–5 where new modules are produced, to create a series of incremental releases.

Step 4 of the XD process identifies the core iteration performed by a design pair, which is focused on the development of the ontology module(s) representing one user story. Figure 3.8 depicts the main steps of a single iteration (i.e., a detailed breakdown of the steps within the dashed rectangle of Fig. 3.7).

First, the development pair analyzes the selected user story and derives a set of CQs, contextual statements and reasoning requirements from it (*Step 4.1. Eliciting requirements*). In order to do that, designers could involve the customer for having feedback and clarifications. For example, the story "Tuna observation" (see Table 3.1) can be transformed into the following CQs, which are added to the story's wiki page:

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

Additionally, assume that the following contextual statements and reasoning requirements are derived based on a discussion with the customer representative:

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

The iteration continues by further breaking down the task, before starting to address it through modeling. This is done by selecting one of the competency questions, or a small set of them that constitutes a coherent modeling issue, and then start matching them to the competency questions associated with available CPs in order to identify candidates for reuse (*Step 4.2. Matching and selecting patterns*). In our example, let CQ_1 and CQ_2 be the selected competency questions. Candidate CPs for reuse would be *situation* and *time interval*. The competency question of *situation* – “What entities are in the setting of a certain situation?” – can be said to match the observation of the resource and the parameters that are in the setting of that observation. Additionally, the *time interval* CP may be seen as matching the question of what period a certain observation was made (CQ_2), although this could also be solved with just a simple data type property.

The following step is to select which of those patterns should actually be used for solving the modeling problem. In our case, there are only two patterns, and neither is an alternative solution to the other, but in many cases, this step involves making some modeling choices, i.e., deciding which pattern is most suitable for the particular case. Nevertheless, in our example, we still need to decide if both patterns are really needed or if they add too much overhead to our model. For instance, we may decide that *time interval* adds too many extra elements to the model since perhaps our customer simply wants to store the year of the observation, rather than an exact period of dates, in which case we will only select *situation*.

After selecting a set of CPs, it is time to start modeling, i.e., *reusing* the CPs (*Step 4.3. Reusing and integrating CPs*). The term “reuse” here refers to the application of typical operations that can be applied to CPs, i.e., *import*, *specialization* and *composition* (see Sect. 3.3.1). In some cases, one may also decide to clone a CP, e.g., if it is desirable not to rely on imports external to the project, which would result in replicating the modeling solution, but without importing the available building block. The latter has both advantages, e.g., reducing the size of the module in case the complete transitive closure of CP imports is not cloned, and disadvantages, e.g., the loss of a common “language” by not referring to the pattern explicitly and reduced support for automatic alignment with other pattern-based modules.

In our example, we import and specialize *situation* in order to address CQ_1, as shown in Fig. 3.9. Our particular situation is an “observation,” and the thing observed is an “aquatic resource.” Additionally, the exploitation state, climatic zone, and vertical distance of the observation are also involved in the setting. Thereby, we add a subclass of `situation:Situation` named `AquaticResourceObservation` and add the other entities as subclasses of `owl:Thing`. In addition, we define sub-properties of the `situation:isSettingFor` and

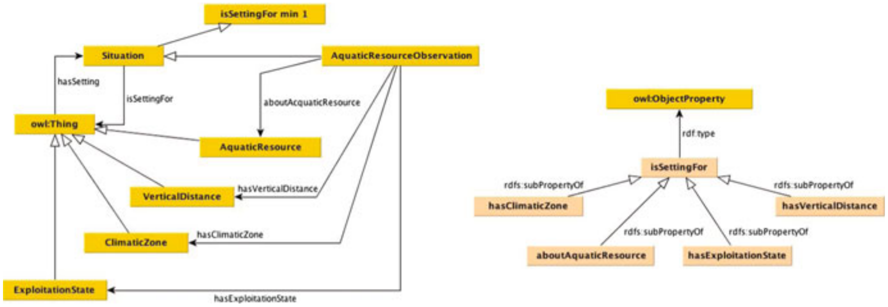


Fig. 3.9 Specialization of the situation CP for modeling aquatic resource observations, as for addressing CQ_1

its inverse, for connecting the observations to the resources and the different parameters. After iterating over all selected CPs (in our example, only one pattern was selected) and integrating them into the current module, the module also has to be extended to cover the complete set of CQs. In our example, no pattern was selected to solve the time period issue in CQ_2; hence, a data type property has to be added to the module in order to cover the complete CQ.

The goal of the following task (*Step 4.4. Testing module*) is to validate the ontology module against the requirements it is supposed to address, i.e., CQs, contextual statements, and reasoning requirements, through developing and executing verification tests, inference tests, and stress tests (see description of test types in Sect. 3.4.1). The ontology modules are revised until all unit tests run successfully. All unit tests are documented in the project wiki and are properly linked to their motivating user story, and requirement(s), in order to document the testing activity as well as to preserve the unit tests for the integration process. In our example, a unit verification test associated with CQ_2 could be the following SPARQL query, retrieving the exploitation state (?exp), vertical distance (?dist), climatic zone (?zone), and resources (?resource) of available observations (?obs):

```
SELECT ?exp ?dist ?resource ?zone
WHERE {

  ?obs a:AquaticResourceObservation.
  ?obs aboutAquaticResource ?resource.
  ?obs hasClimaticZone ?zone
  ?obs hasExploitationState ?exp.
  ?obs hasVerticalDistance ?dist

}
```

If all requirements derived from the story have been solved, and all tests run successfully, the design pair proceeds to internally release the module(s) (*Step 5. Releasing module(s))*, which are then ready for integration in the current overall increment iteration.

3.4.3 Example: A Music Industry Ontology

To illustrate the process of creating ontologies through XD, a small hypothetical project is described in this section. The domain, which is *music industry*, should be intuitive to most readers. The example is not intended as a case for validating the methodology, but merely as an illustration how it could be used in practice.

Step 1 – Project initiation and scoping. Let us assume that an ontology is needed in an online community platform for people who want to discuss music, share news, playlists, and music recommendations. The ontology will be used to store and retrieve information about music recordings and artists, as well as to reason over musical genres. As ontology developers, we are working closely together with the software developers, implementing the online community software; however, we also have access to some future administrators of the community, who are used as the "customer representatives", i.e., domain experts, during the project. A wiki is set up for the development project, where all information is stored and shared, both between developers and with the customer representatives.

Step 2 – Identifying CP catalogs. Let us assume that we decide to focus on CP reuse and to mainly reuse CPs from the ODP portal.

Step 3 – Collecting requirement stories. As soon as the project environment is set up, we ask the customer representatives to start entering their stories into the wiki. Some stories become examples of typical information that is to be stored by the ontology, while other stories focus more on reasoning tasks of the ontology, depending on how the customer representatives formulate them. Each story is entered into the story template, e.g., given a title and priority, and as soon as several stories are in place, they can be related to each other. A collected story can be seen in Table 3.2, and another one in Table 3.3. The "Albums"-story depends on the "Recordings of songs" story, since the notion of recorded track appears also in the story about albums but is the main focus of "Recordings of songs."

Step 4 – Eliciting requirements and constructing modules from CPs. At this point, the design team is divided into pairs, in order to develop the ontology modules using pair design. One pair is dedicated to the integration task, i.e., proceed directly to prepare Steps 6–7, while the rest of the pairs choose their first stories from the pool of collected ones. Each story with high priority has to be solved before the lower priority ones are addressed. Each pair then starts to elicit requirements from their chosen story, i.e., tries to derive CQs, contextual statements, and reasoning requirements.

Table 3.2 A requirement story from the music domain concerning albums

Title	Albums
Depends on	Recordings of songs
Description	An album is a collection of recorded tracks. The genre(s) of an album should be derived based on the genres of the tracks it contains
Priority	High

Table 3.3 A requirement story from the music domain concerning songs and recordings

Title	Recordings of songs
Depends on	
Description	Songs are recorded by artists. Many artists can record the same song. In the web interface, users will click on songs and get to see the artists that have recorded them and links to information on those recordings
Priority	High

Let us imagine that you are part of the design pair who picks up the story in Table 3.2. First, you start analyzing the story itself, to see if there are any obvious CQs, indicating information that should be stored and retrieved. The most obvious CQs are:

1. What are the tracks of this album?
2. What is the genre of this track or album?

There could however be other CQs possible; hence, the final list needs to be agreed with a customer representative, in order to ensure appropriate coverage of the domain and task and to avoid misinterpretations of the story. In addition, it is evident from the way the story is written that a reasoning requirement is needed, i.e., the following:

- An album should be automatically assigned all the genres of it contained tracks.

In other cases, it may not be as self-evident what needs to be possible to infer; however, in many cases, software requirements and interactions with the customer can clarify such issues.

Additionally, contextual statements can be proposed based on common sense knowledge, e.g., in our case:

- An album always has at least one track.

Other contextual statements may be given by the customer representative or be implicit in the software requirements, e.g., limitations set by the way the software will use the ontology. We have thus collected two CQs, one contextual statement and one reasoning requirement, based on the story in Table 3.2 and our interaction with the customer representatives.

Next, the pair proceeds to select a subset of the requirements, which represent some particular modeling issue. When analyzing the CQs, we note that the first one is focused on the album as a collection of tracks, while the second one adds the notion of genre. These are actually quite separate concerns, and in order to decouple these modeling issues, we decide to create one module for each CQ.

Choosing to start with the first CQ and the contextual statement, we now need to match the CQ to the requirements covered by the CPs in the ODP portal. When searching the portal's CP submission table[14], we find that there are several interesting CPs, e.g., there is the *collection* CP[15] for representing membership, and the *part of* CP for part-whole relations. In this case, both patterns have matching CQs, so the choice is instead based on how we wish to view the album, i.e., as an object divided into parts or as a collection that is the sum of its members. One of the main differences between the patterns is that the *part of* CP

[14] http://ontologydesignpatterns.org/wiki/Submissions:ContentOPs

[15] http://ontologydesignpatterns.org/wiki/Submissions:Collection

defines the part-whole relation as transitive, while the membership relation in the *collection* CP is not. Since we are not interested in creating a hierarchy of parts, we decide on the *collection* CP and document this choice in the project wiki (including our argumentation).

Then it is time to start modeling. Since we are creating a new module each time, we start by creating an empty ontology, with a new namespace (following a namespace convention agreed in Step 1). Next, we import the OWL building block of the *collection* CP into our empty ontology and start specializing it. As a subclass of `collection:Collection`, we create a new class `Album`, and then another new class called `Track` (subclass of `owl:Thing`). To complete the specialization, we create sub-properties of the pattern properties, with more domain-specific names, e.g., `containsTrack` and `containedInAlbum`, set them to be inverses, and define domain and range axioms. Figure 3.10 illustrates the result of this process. Each entity we create is commented, and given a label, and we additionally extend the specialized CP, by adding the contextual statement as a cardinality restriction over the `containsTrack` property on the `Album` class.

When the design pair is satisfied, it is time to test the module they have created. First, we formulate the CQ as a SPARQL query. Most often, missing parts are discovered already when formulating the query since the query formulation involves an inspection of the model. However, a new ontology (i.e., a "test case") is created, importing the ontology to be tested, and some test instances are added in the test case ontology. If the SPARQL query gives the expected result, based on our test data, then the test is successful. We can proceed to perform some stress testing. In this case, we should add data that violates some constraint, e.g., a contextual statement, and see that the ontology is able to detect the problem, e.g., through finding an inconsistency, and that there are no undesired

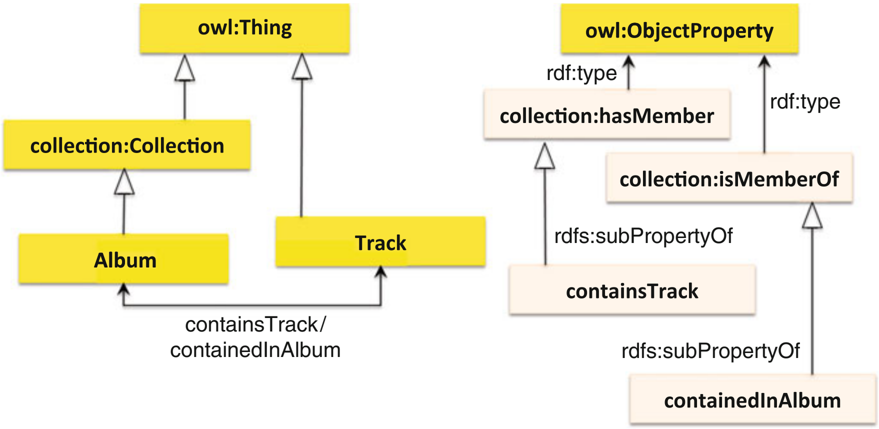


Fig. 3.10 Specialization of the *collection* CP

side effects. In our case, however, the open-world assumption of OWL makes it hard to detect violations of our contextual statement.

When all tests run successfully, and the module is fully annotated and documented in the wiki, it is time to proceed with the next set of requirements, i.e., the second CQ and the reasoning requirement. Similarly as before, we start by matching the requirements to the list of CPs in the ODP portal. This time, we do not immediately find a match, i.e., there is no pattern for music genres; however, there are patterns for expressing descriptions and parameters of a concept. Nevertheless, let us assume that we find these too abstract for our case, and instead choose to create the model on our own.

Just as in the previous iteration, we start by creating a new empty ontology with its own namespace. However, this time we realize that we need the tracks and albums that we just modeled in the previous module; hence, we import it into our new ontology module. Then we add the class `Genre` and a property `hasGenre` (including its inverse `genreOf`). The domain of `hasGenre` is set to the union of `Track` and `Album`, while the range is set to the `Genre` class. In addition, to solve the reasoning requirement, we add a property chain definition to the `hasGenre` property, stating that `hasGenre` can be derived from the combination of the `hasTrack` and `hasGenre` properties, meaning that if an album has a track which in turn has a certain genre, then that album should also be directly connected to the same genre.

Testing this time involves testing the CQ using one or more SPARQL queries but also to test the inferences produced based on the property chain, i.e., to confirm that the reasoning requirement is fulfilled. To do the latter, we create a new empty ontology, import our module to be tested and add some test data that should produce the correct inference. For instance, an album instance can be added, then associated to a track (through `hasTrack`), and the track's genre set to `rock` (through `hasGenre`). When the inferences are materialized, we expect to see that the album is now also associated with the genre `rock`. As soon as all tests run successfully, and the ontology module is appropriately commented, we are now ready to release the complete solution of the customer story in Table 3.2, consisting of our two ontology modules.

Step 5 – Releasing module(s). The modules, and all their wiki documentation, are now made available to the pair in charge of integration.

Step 6 – Integrating partial solutions, evaluating, and revising. As soon as the integration team have more than one solution to work with, i.e., more than one story is covered, they start integrating the modules. Integration is a crucial part and involves a trade-off between refactoring, to reduce overlap between modules, and keeping the decoupling of modules to facilitate later changes and reusability of individual modules. In some cases, integration is quite easy, e.g., the modules can directly be imported into one new ontology, and tested together, without any additional modeling, while in other cases, the integration means to add some "glue" to resolve conflicts and make sure that the requirements of the stories treated so far can be covered all together. However, the use of CPs facilitates the integration since it makes explicit the modeling choices made, assures that the development team has a shared vocabulary for talking about modeling choices, and in some cases even makes the integration semi-automatic, i.e., if the same CP is imported in several modules they are inherently aligned.

While the integration pair starts their task, our design pair can now go back to the list of remaining user stories, and select a new one, to start another development iteration. This process is continued until no more stories are to be covered.

Step 7 – Releasing new version of ontology network. After each new module has been integrated into the resulting ontology (i.e., ontology network), a new release is created, letting the customer and other parties, e.g., software developers, review and test the ontology at all stages of development.

3.4.4 Tool Support

In this section, we briefly present the ODP portal[16] and the eXtreme Design Tools (XD Tools), two resources that support XD. The ODP portal is a semantic wiki dedicated to best practices of ontology design for the semantic web, with particular focus on ODPs. The ODP portal supports the life cycle of ODPs, i.e., from their proposal to their evaluation and possible certification. CP wiki pages can be created automatically in the wiki by providing, as input, the CP OWL file properly annotated. Currently, the ODP portal supports the life cycle of *Content ODPs, Re-engineering ODPs, Alignment ODPs, Logical ODPs, Architectural ODPs,* and *Lexico-syntactic ODPs*. The ODP portal is associated with a registry of CPs[17].

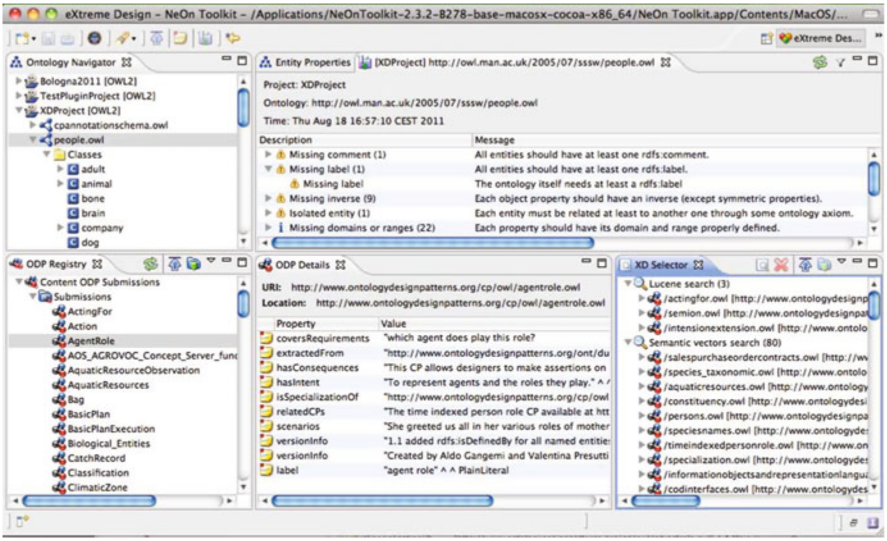
While the ODP portal is meant to give community support to XD, the XD Tools are meant to assist the execution of the XD methodology. XD Tools are a set of software components released as an Eclipse plugin, accessible through a perspective – eXtreme Design – compatible with Eclipse-based ontology design environments, such as the NeOn Toolkit. Currently, XD Tools are comprised of five main components that allow a user to browse a registry of CPs, search and import them into a local ontology project. Although specialization is possible through native NeOn Toolkit functionalities, XD Tools feature a wizard for specializing a CP, for usability reasons. As a special feature, a service for analyzing an ontology with respect to general modeling best practices is also included.

Figure 3.11 gives an overview of the XD Tools interface as it appears in the NeOn Toolkit. The *ODP Registry view* (bottom left of Fig. 3.11 – enlarged view in Fig. 3.12) exposes a tree-like view of a CP registry that can be browsed by a user. The default registry used is the ODP portal registry, but others can be added through customizing the plugin. When a CP is selected, the *ODP Details view*

[16] The ODP portal main page, http://www.ontologydesignpattern.org

[17] The ODP Portal pattern registry can be downloaded at: http://ontologydesignpatterns.org/schemas/registry.owl

(to the right of the registry view in both figures) shows a description of it, based on the annotations stored in the CP's OWL file. By right clicking on a CP, its OWL file can be downloaded through the "Get" command and put in a local ontology project. The *ODP Selector view* (bottom right of Fig. 3.11 – enlarged view in Fig. 3.13) provides a search service over the CP registry. By clicking on the "Search" icon (highlighted by a small circle in Fig. 3.13), a user can type a natural language query, e.g., a competency question, in a text field, and submit it to a set of search services that return ranked lists of CPs, from which the user can select the most appropriate one(s).


eXtreme Design - NeOn Toolkit - /Applications/NeOnToolkit-2.3.2-B278-base-macosx-cocoa-x86_64/NeOn Toolkit.app/Contents/MacOS/...
eXtreme Des...
Ontology Navigator
Entity Properties
[XDProject] http://owl.man.ac.uk/2005/07/sssw/people.owl
ODP Registry
ODP Details
XD Selector

Bologna2011 [OWL2]
TestPluginProject [OWL2]
XDProject [OWL2]
cpannotationschema.owl
people.owl
Classes
adult
animal
bone
brain
company
dog

Project: XDProject
Ontology: http://owl.man.ac.uk/2005/07/sssw/people.owl
Time: Thu Aug 18 16:57:10 CEST 2011

Description	Message
Missing comment (1)	All entities should have at least one rdfs:comment.
Missing label (1)	All entities should have at least one rdfs:label.
Missing label	The ontology itself needs at least a rdfs:label
Missing inverse (9)	Each object property should have an inverse (except symmetric properties).
Isolated entity (1)	Each entity must be related at least to another one through some ontology axiom.
Missing domains or ranges (22)	Each property should have its domain and range properly defined.

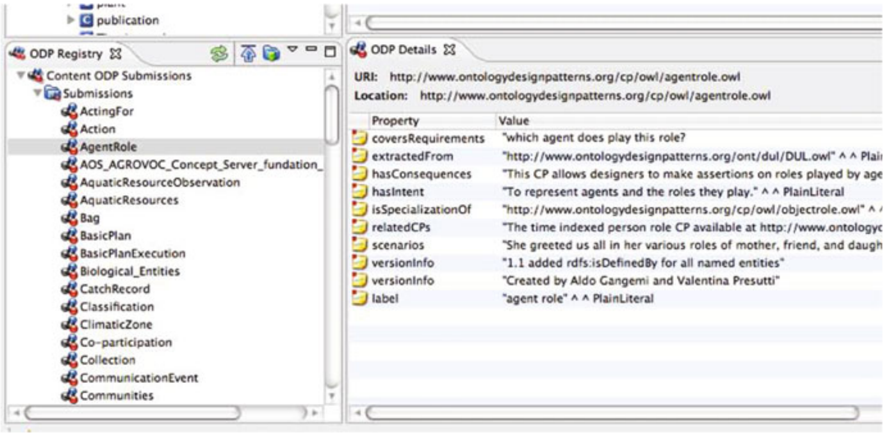
Content ODP Submissions
Submissions
ActingFor
Action
AgentRole
AOS_AGROVOC_Concept_Server_func
AquaticResourceObservation
AquaticResources
Bag
BasicPlan
BasicPlanExecution
Biological_Entities
CatchRecord
Classification
ClimaticZone

URI: http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl
Location: http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl

Property	Value
coversRequirements	"which agent does play this role?
extractedFrom	"http://www.ontologydesignpatterns.org/ont/du
hasConsequences	"This CP allows designers to make assertions on
hasIntent	"To represent agents and the roles they play." ^ /
isSpecializationOf	"http://www.ontologydesignpatterns.org/cp/owl
relatedCPs	"The time indexed person role CP available at htt
scenarios	"She greeted us all in her various roles of mother
versionInfo	"1.1 added rdfs:isDefinedBy for all named entitie:
versionInfo	"Created by Aldo Gangemi and Valentina Presutti
label	"agent role" ^ ^ PlainLiteral

Lucene search (3)
/actingfor.owl [http://www.ontologydesignp
/semion.owl [http://www.ontologydesignpa
/intensionextension.owl [http://www.ontolo
Semantic vectors search (80)
/salespurchaseordercontracts.owl [http://ww
/species_taxonomic.owl [http://www.ontolo
/aquaticresources.owl [http://www.ontology
/constituency.owl [http://www.ontologydesi
/persons.owl [http://www.ontologydesignpa
/speciesnames.owl [http://www.ontologyde:
/timeindexedpersonrole.owl [http://www.on
/specialization.owl [http://www.ontologydes
/informationobjectsandrepresentationlangua
/codinterfaces.owl [http://www.ontologydes

Fig. 3.11 Screenshot overview of XD Tools



publication

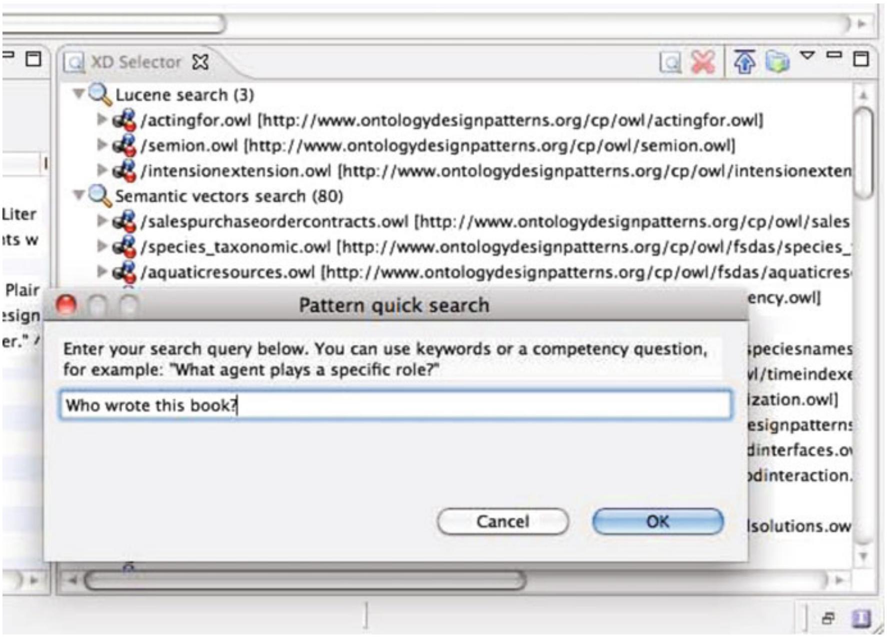
Content ODP Submissions
Submissions
ActingFor
Action
AgentRole
AOS_AGROVOC_Concept_Server_fundation_
AquaticResourceObservation
AquaticResources
Bag
BasicPlan
BasicPlanExecution
Biological_Entities
CatchRecord
Classification
ClimaticZone
Co-participation
Collection
CommunicationEvent
Communities

URI: http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl
Location: http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl

Property	Value
coversRequirements	"which agent does play this role?
extractedFrom	"http://www.ontologydesignpatterns.org/ont/dul/DUL.owl" ^ ^ Plai
hasConsequences	"This CP allows designers to make assertions on roles played by age
hasIntent	"To represent agents and the roles they play." ^ ^ PlainLiteral
isSpecializationOf	"http://www.ontologydesignpatterns.org/cp/owl/objectrole.owl" ^ /
relatedCPs	"The time indexed person role CP available at http://www.ontologyc
scenarios	"She greeted us all in her various roles of mother, friend, and daugh
versionInfo	"1.1 added rdfs:isDefinedBy for all named entities"
versionInfo	"Created by Aldo Gangemi and Valentina Presutti"
label	"agent role" ^ ^ PlainLiteral

Fig. 3.12 Screenshot depicting the ODP Registry view (*left*) and the ODP Details view (*right*)

The *XD Analyzer view* (top right of Fig. 3.11 – enlarged view in Fig. 3.14) can be run through a contextual menu on a selected ontology and shows a list of messages, each associated with a best practice criterion. A message indicates whether the ontology satisfies a certain criterion or not. The XD Analyzer has a pluggable architecture, allowing for easy extension of the set of heuristics that express "best practices." Three levels of messages can be produced: errors, warnings, and suggestions (i.e., proposals for improvement). An *error* is, for instance, a missing type, i.e., all instances should have an explicit class as its type (could be `owl:Thing`).



XD Selector

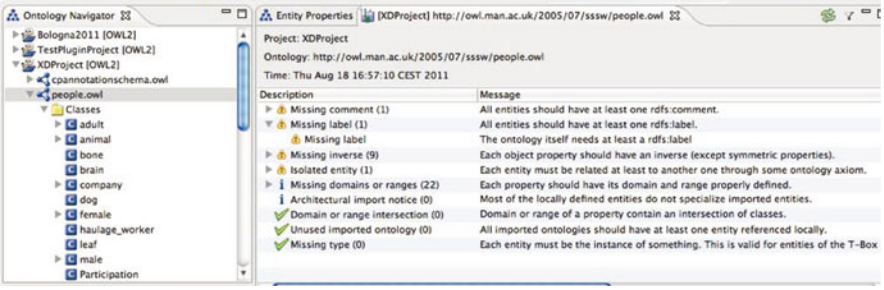
- Lucene search (3)
  - /actingfor.owl [http://www.ontologydesignpatterns.org/cp/owl/actingfor.owl]
  - /semion.owl [http://www.ontologydesignpatterns.org/cp/owl/semion.owl]
  - /intensionextension.owl [http://www.ontologydesignpatterns.org/cp/owl/intensionexten
- Semantic vectors search (80)
  - /salespurchaseordercontracts.owl [http://www.ontologydesignpatterns.org/cp/owl/sales
  - /species_taxonomic.owl [http://www.ontologydesignpatterns.org/cp/owl/fsdas/species_
  - /aquaticresources.owl [http://www.ontologydesignpatterns.org/cp/owl/fsdas/aquaticres

Pattern quick search

Enter your search query below. You can use keywords or a competency question, for example: "What agent plays a specific role?"

Who wrote this book?

Fig. 3.13 Screenshot depicting the ODP Selector view and its search query interface



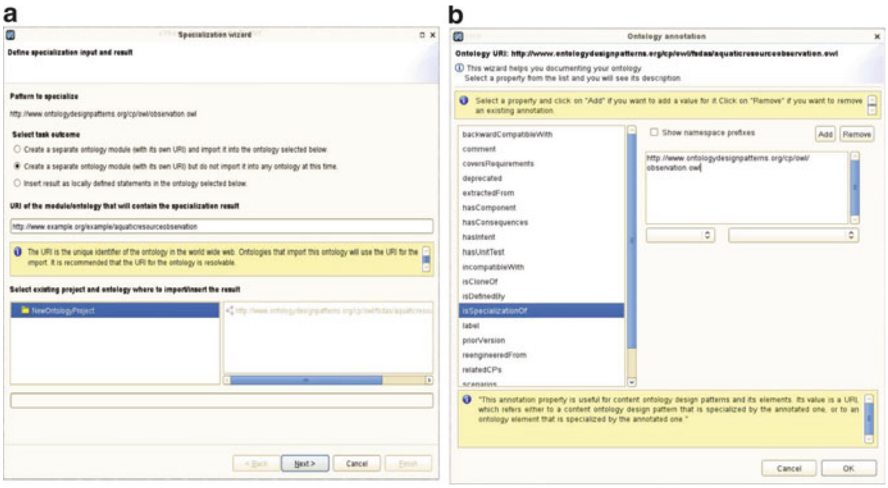
Project: XDProject

Ontology: http://owl.man.ac.uk/2005/07/sssw/people.owl

Time: Thu Aug 18 16:57:10 CEST 2011

Description	Message
Missing comment (1)	All entities should have at least one rdfs:comment.
Missing label (1)	All entities should have at least one rdfs:label.
Missing label	The ontology itself needs at least a rdfs:label
Missing inverse (9)	Each object property should have an inverse (except symmetric properties).
Isolated entity (1)	Each entity must be related at least to another one through some ontology axiom.
i Missing domains or ranges (22)	Each property should have its domain and range properly defined.
i Architectural import notice (0)	Most of the locally defined entities do not specialize imported entities.
✓ Domain or range intersection (0)	Domain or range of a property contain an intersection of classes.
✓ Unused imported ontology (0)	All imported ontologies should have at least one entity referenced locally.
✓ Missing type (0)	Each entity must be the instance of something. This is valid for entities of the T-Box

Fig. 3.14 Screenshot depicting the XD Analyzer, showing the results of the analysis in the list to the right. Yellow triangles indicate warnings, while "i" stands for suggestion. The number of occurrences is given within brackets



Define specialization input and result

Pattern to specialize

http://www.ontologydesignpatterns.org/cp/owl/observation.owl

Select task outcome

○ Create a separate ontology module (with its own URI) and import it into the ontology selected below.

● Create a separate ontology module (with its own URI) but do not import it into any ontology at this time.

○ Insert result as locally defined statements in the ontology selected below.

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

http://www.example.org/example/aquaticresourceobservation

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

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

NewOntologyProject

Ontology URI: http://www.ontologydesignpatterns.org/cp/owl/fsdas/aquaticresourceobservation.owl

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

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

□ Show namespace prefixes

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

http://www.ontologydesignpatterns.org/cp/owl/observation.owl

"This annotation property is useful for content ontology design patterns and its elements. Its value is a URI, which refers either to a content ontology design pattern that is specialized by the annotated one, or to an ontology element that is specialized by the annotated one."

Fig. 3.15 Part (**a**) shows the XD Specialization wizard and (**b**) the XD Annotation dialog

Examples of *warnings* are missing labels and comments, as well as proposals to create an inverse for each object property that has no inverse so far in the analyzed ontology. A *suggestion* could be a message to check the object properties that lack domain and range definitions, i.e., such definitions are not mandatory in a well-designed ontology, since they could be replaced by other axioms; however, if they are missing, it could also indicate that the developer has forgot to add them.

XD Tools also include a wizard for guiding users in the process of specializing a CP. Figure 3.15a shows the *Specialization wizard*. CP specialization, as the primary step of their reuse, can be challenging for an inexperienced user if it is done one element at a time, without guidance. From a user perspective, CP specialization has the following steps: (1) import the pattern into the working ontology, (2) define subclasses/sub-properties for each of the (most specific) pattern entities needed and (3) add any additional appropriate axioms. The specialization wizard provided by XD Tools guides the user through this process. Finally, XD Tools provide a so-called *Annotation dialog* – depicted in Fig. 3.15b – which supports annotation, i.e., documentation, of an ontology based on customizable annotation vocabularies.

In addition, XD Tools provide several help functions, such as inline info boxes, help sections in the Eclipse help center and "cheat sheets" describing the XD methodology for CP reuse.

3.5 Conclusion

In this chapter, we have presented ontology design patterns (ODPs), which are reusable modeling solutions that encode modeling best practices, by briefly discussing their different types and characteristics. ODPs are the main tool for performing pattern-based design of ontologies, which is an approach to ontology development that emphasizes reuse and promotes the development of a common "language" for sharing knowledge about ontology design best practices. ODPs are associated with a set of requirements that are explicitly expressed in order to favor their selection through a matching procedure. Content ODPs (CPs) have been the main focus of this chapter, which has shown through some examples how they can be used for building an ontology according to a set of elicited requirements. CPs are domain-dependent patterns, the requirements of which are expressed by means of competency questions, contextual statements, and reasoning requirements. In order to reuse CPs, we have defined a set of operations that include importing, specializing, and composing them to the aim of building a new ontology (or ontology network).

In the second part of the chapter, we have described an agile methodology for pattern-based ontology design named eXtreme Design (XD), an iterative and incremental process, which is characterized by a test-driven and collaborative development approach. The XD methodology is supported by a set of software components named XD Tools, which assist users in the process of pattern-based design.

The XD methodology has been tested in numerous ontology development projects, including user-based experiments conducted in controlled environments. The results of those experiments have been reported by Blomqvist and colleagues (2009a) and by Blomqvist and colleagues (2010a). The participants were, for instance, asked to assess how useful ODPs and XD were and how much overhead it added to their work processes, as well if XD felt like a natural way of working, i.e., if they were already working in a similar way before being introduced to the methodology. Overall, the methodology was received well, and the participants felt that it was a very natural way of working, without adding any unnecessary restrictions to the process. Nevertheless, the objective evaluation of their modeling results showed that the quality increased drastically, in particular with respect to a number of common mistakes, when introducing the methodology. This could be attributed to the testing focus of the methodology that enforces a rigorous evaluation of each solution before its release. So even though the participants felt that the methodology added nothing new, it actually helped them to structure their work and provide better and more rigorously tested ontologies.

References

Aguado de Cea G, Gómez-Pérez A, Montiel-Ponsoda E, Suárez-Figueroa MC (2009) Using linguistic patterns to enhance ontology development. In: Dietz J (ed) Proceedings of the international conference on knowledge engineering and ontology development (KEOD), Funchal, pp 206–213

Baker CF, Fillmore CJ, Lowe JB (1998) The Berkeley FrameNet project. In: Boitet C, Whitelock P (eds) Proceedings of the 36th annual meeting of the Association for Computational Linguistics and 17th international conference on computational linguistics, vol 1. Association for Computational Linguistics, Stroudsburg, PA, USA, pp 86–90

Basili V, Caldiera G, Rombach D (1994) The experience factory. In: Marciniak J (ed) Encyclopedia of software engineering. Wiley, New York, pp 469–476

Bizer C, Heath T, Berners-Lee T (2009) Linked data – the story so far. Int J Semant Web Inf Syst 5 (3):1–22

Blomqvist E, Gangemi A, Presutti V (2009a) Experiments on pattern-based ontology design. In: Proceeding of K-CAP 2009, Los Angeles. ACM, New York

Blomqvist E, Sandkuhl K, Scharffe F, Svatek V (2009b) Proceedings of the workshop on ontology patterns (WOP 2009), collocated with the 8th international semantic web conference (ISWC-2009), Washington, DC, USA, 25 Oct, 2009, vol 516. CEUR

Blomqvist E, Presutti V, Daga E, Gangemi A (2010a) Experimenting with eXtreme design. In: Proceedings of EKAW2010 – knowledge engineering and management by the masses, LNCS 6317. Springer, Berlin/Heidelberg/New York

Blomqvist E, Chaudhri V, Corcho O, Presutti V, Sandkuhl K (2010b) Proceedings of the 2nd international workshop on ontology patterns – WOP2010, vol 671. CEUR

Gamma E, Helm R, Johnson R, Vlissides J (1994) Design patterns: elements of reusable object-oriented software. Addison-Wesley, Reading

Gangemi A, Borgo S (2004) Core ontologies in ontology engineering 2004. (Un) Successful cases and best practices for ontology engineering: reusing well-founded ontologies for domain content specification. In: Proceedings of the EKAW*04 workshop on core ontologies in ontology engineering, Northamptonshire (UK), 8 Oct, 2004, vol 118. CEUR

Gangemi A, Presutti V (2009) Ontology design patterns. In: Staab S, Studer R (eds) Handbook on ontologies, 2nd edn. Springer, Berlin, pp 221–243

Gangemi A, Fisseha F, Keizer J, Lehmann J, Liang A, Pettman I, Sini M, Taconet M (2004) A core ontology of fishery and its use in the FOS project. In: EKAW 2004 workshop on core ontologies in ontology engineering, Northamptonshire. CEUR

Gangemi A, Sagri MT, Tiscornia D (2005) A constructive framework for legal ontologies. In: Law and the semantic web. Legal ontologies, methodologies, legal information retrieval, and applications. 3369. Springer, Berlin/Heidelberg/New York

Gruninger M, Fox MS (1994) The role of competency questions in enterprise eEngineering. In: IFIP WG5.7 workshop on benchmarking – theory and practice, Trondheim

Hammar K, Sandkuhl K (2010) The state of ontology pattern research: a systematic review of ISWC, ESWC and ASWC 2005–2009. In: Blomqvist E, Chaudhri VK, Corcho O, Presutti V, Sandkuhl K (eds) Proceedings of the 2nd International workshop on ontology patterns – WOP2010. Workshop at the 9th international semantic web conference (ISWC2010) – ISWC 2010 workshops, vol VIII. Shanghai, China, 8 Nov, 2010, vol 671. CEUR

Hay DC (2000) Data model patterns: conventions of thought. Dorset House Publishing, New York

Masolo C, Borgo S, Gangemi A, Guarino N, Oltramari A (2005) The wonderweb library of foundational ontologies. Wonderweb deliverable D18. Laboratory for applied ontology (ISTC-CNR)

Miles A, Bechhofer S (2009) SKOS simple knowledge organization system reference. W3C

Niles I, Pease A (2001) Towards a standard upper ontology. In: Welty C, Smith B (eds) 2nd international conference on formal ontology in information systems (FOIS-2001), Ogunquit

Noy N, Rector A (2004) Defining N-ary relations on the semantic web: use with individuals. W3C

Presutti V, Daga E, Gangemi A, Blomqvist E (2009) eXtreme design with content ontology design patterns. In: Blomqvist E, Sandkuhl K, Scharffe F, Svatek V (eds) Proceedings of the workshop on ontology patterns (WOP 2009), collocated with the 8th international semantic web conference (ISWC-2009), Washington, DC, USA, 25 Oct 2009, vol 516. CEUR

Rector A, Stevens R (2008) Barriers to the use of OWL in knowledge driven applications. In: Dolbear C, Ruttenberg A, Sattler U (eds) Proceedings of the fifth OWLED workshop on OWL: experiences and directions collocated with the 7th international semantic web conference (ISWC-2008) Karlsruhe, Germany, 26–27 Oct 2008, vol 432. CEUR

Scharffe F, Fensel D (2008) Correspondence patterns for ontology alignment. In: Gangemi A, Euzenat J (eds) Proceedings of the 16th international conference, EKAW 2008, Acitrezza, Italy. 5268. Springer, Berlin/Heidelberg/New York, pp 83–92

Shore J, Warden S (2007) The art of agile development. O'Reilly, Farnham

Svátek V, Sváb-Zamazal O, Presutti V (2009) Ontology naming pattern sauce for (human and computer) gourmets. In: Workshop on ontology patterns at ISWC'09, Washington DC, 2009. 516. CEUR

Vrandečić D, Gangemi A (2006) Unit tests for ontologies. In: Proceedings of the 1st international workshop on ontology content and evaluation in enterprise. Springer, Berlin/Heidelberg/New York

Vrandečić D, Sure Y (2007) How to design better ontology metrics. In: May W, Kifer M (eds) 4th European semantic web conference (ESWC'07). Springer, Berlin/Heidelberg/New York

Chapter 4
The NeOn Ontology Models

Alessandro Adamou, Raúl Palma, Peter Haase, Elena Montiel-Ponsoda, Guadalupe Aguado de Cea, Asunción Gómez-Pérez, Wim Peters, and Aldo Gangemi

Abstract Interoperability on multiple levels, concerning both the ontologies themselves and their engineering activities, is a key requirement for ontology networks to be efficient, with minimal redundancy and high reuse. This requirement has a strict binding for software tools that can support some interoperability levels, yet they can be hindered by a lack of shared models and vocabularies describing the resources to be handled, as well as the ways of handling them. Here, three examples of metalevel vocabularies are proposed, each covering at least one peculiar

A. Adamou (✉)
Semantic Technologies Lab, Institute of Cognitive Sciences, and Technologies (National Research Council – CNR), Via Nomentana 56, 00161 Rome, Italy

Department of Computer Science, Alma Mater Studiorum Universitá di Bologna, Mura Anteo Zamboni 7, 40126 Bologna, Italy
e-mail: alessandro.adamou@istc.cnr.it; adamou@cs.unibo.it

R. Palma
Poznan Supercomputing and Networking Center, ul. Dabrowskiego 79a, 60-529 Poznan, Poland
e-mail: rpalma@man.poznan.pl

P. Haase
fluid Operations AG, Altrottstr. 31, 69190 Walldorf, Germany
e-mail: peter.haase@fluidops.com

E. Montiel-Ponsoda • G. Aguado de Cea • A. Gómez-Pérez
Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo sn, 28660 Boadilla del Monte, Madrid, Spain
e-mail: emontiel@fi.upm.es; lupe@fi.upm.es; asun@fi.upm.es

W. Peters
University of Sheffield, Sheffield, UK
e-mail: w.peters@dcs.shef.ac.uk

A. Gangemi
Semantic Technologies Lab, Institute of Cognitive Sciences, and Technologies (National Research Council – CNR), Via Nomentana 56, 00161 Rome, Italy
e-mail: aldo.gangemi@cnr.it

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_4, © Springer-Verlag Berlin Heidelberg 2012

interoperability aspect: OMV for modeling the artifacts themselves, LIR for managing a multilingual layer on top of them, and C-ODO Light for modeling collaboration-supportive life cycle management tasks and processes. All of these models lend themselves to handling by dedicated software tools and are all being employed within NeOn products.

4.1 Introduction

Authoring ontologies and modeling domains of interest are only part of an ontology life cycle management process. If these activities are carried out in a monolithic fashion, from scratch and without reusing readily available knowledge models, this may lead to costly "reinventions of the wheel" and contradicts the Semantic Web philosophy of an open knowledge world. On the other hand, even when the intention and sentiment to follow this philosophy are present, they might not be encouraged by appropriate tool support. This, in turn, depends on the availability of formal models of processes and artifacts in ontology design, i.e., their *metalevel*. This model may sometimes be implicitly hardwired in the software itself, but if it is not, then it may be useful to share and exploit it for the sake of interoperability, be it conceptual, linguistic, functional, or social.

This chapter focuses on three contributions to the practice of ontology design by metalevel handling. Each contribution, presented itself as an ontology network, covers a specific design perspective, i.e., reuse (OMV), localization (LIR), and collaborative engineering (C-ODO Light). By the end of the chapter, the reader will have a practical insight as to how a model of the ontology metalevel can be employed to build effective software tools to automate engineering tasks.

4.2 Ontology Metadata Vocabulary (OMV)

Ontologies have undergone an enormous development and have been applied in many domains within the last years, especially in the context of the Semantic Web. Currently, however, efficient knowledge sharing and reuse, a prerequisite for the realization of the Semantic Web vision, is a difficult task. It is hard to find and share existing ontologies because of the lack of standards for documenting and annotating ontologies with metadata information. Without ontology-specific metadata, developers are not able to reuse existing ontologies, which leads to interoperability problems, as well as duplicate efforts. In order to provide a basis for an effective access and exchange of ontologies across the web, it is necessary to agree on a standard for ontology metadata. This standard then provides a common set of terms and definitions describing ontologies and is called *metadata vocabulary*.

Limitations. The need for a metadata vocabulary for describing ontologies has been acknowledged in the past by previous efforts (e.g., Dublin Core 1998, Reference Ontology 2000, Ontology Metaontology (OMO) 2003, and DogmaModeler Ontology 2005). However, at the moment, most of the current ontologies exist in pure form without any additional information, e.g., domain of interest, authorship information, and statistic information (Ungrangsi and Simperl 2008). This is due in part to the *lack of standards or community-accepted vocabularies* for documenting and annotating ontologies with metadata information. Moreover, most of the previous efforts carried out on this issue provide only a *list of property-value pairs* for describing ontologies (e.g., Arpírez et al. 2000; Jarrar 2005), limiting the processing capabilities and the related relevant information that can be described. Similarly, ontology metadata are in many of the existing systems and repositories (e.g., DAML Ontology Library[1], SchemaWeb Directory[2], and SWOOGLE[3]), not based on agreed standards, which makes them difficult to integrate or reuse. Finally, *general-purpose standards*, such as Dublin Core, *are not appropriate* for capturing information about ontologies because of the differences between arbitrary information sources and ontologies. For instance, aspects related to the application scenario, scope, purpose, or evaluation results are essential when describing ontologies. Additionally, besides structural and technical information, ontologies have to be described in terms of descriptive metadata, such as provenance information, ontology categorizations, underlying methodologies, or knowledge representation paradigms that are specific for ontologies.

Thereupon, in this chapter we describe our contribution to the alleviation of this situation: the ontology metadata standard OMV (Ontology Metadata Vocabulary), which specifies reusability-enhancing ontology features for human- and machine-processing purposes. It allows to clarify the relations between the available ontologies so that they are easy to search, to characterize, and to maintain. Moreover, it provides the means for making explicit the virtual and implicit network of ontologies.

Ontology Metadata Requirements. As a result of a systematic survey of the state of the art in the area of ontology reuse, we have elaborated an inventory of requirements for the metadata model. Besides analytical activities, we conducted extensive literature research focused on theoretical methods (Pinto and Martins 2001; Gangemi et al. 1999; Lozano-Tello and Gómez-Pérez 2004) and also on case studies on reusing existing ontologies (Uschold et al. 1998; Russ et al. 1999; Paslaru Bontas et al. 2005). Our aim was to identify the real-world needs of the community with respect to a descriptive metadata format for ontologies. Further on, the requirement analysis phase was complemented by a comparative study of existing (ontology-independent) metadata models and tools such as ontology repositories and libraries that (implicitly) make use of some form of ontology metadata.

[1] http://www.daml.org/ontologies/

[2] http://www.schemaweb.info/

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

Several aspects to be considered in ontology metadata representation are definitely similar to those of other more general metadata standards such as Dublin Core. Differences arise, however, if we consider the semantic nature of ontologies, which are much more than plain web information sources. The main requirements identified in this process are the following:

Accessibility. Metadata should be accessible and processable for both humans and machines. Whereas the human-driven aspects are ensured by the usage of natural language concept names, the machine-readability requirement can be implemented by the usage of web-compatible representation languages (such as XML or Semantic Web languages, see below). Furthermore, having metadata in processable format will facilitate the implementation of tools that use or manage ontology-related metadata (e.g., ontology changes).

Usability. A metadata model should (1) reflect the needs of the majority of ontology users, as reported by existing case studies in ontology reuse, but at the same time (2) allow proprietary extensions and refinements in particular application scenarios (e. g., ontology change management). From a content perspective, usability can be maximized by taking into account multiple metadata types, which correspond to specific viewpoints on the ontological resources and are applied in various application tasks. Despite the broad understanding of the metadata concept and the use cases associated to each definition, several key aspects of metadata information have already been established across computer science fields (NISO 2004):

- *Structural metadata* relate to statistical measures on the graph structure underlying an ontology. In particular, we mention the number of specific ontological primitives (e.g., number of classes and individuals). The availability of structural metadata influences the usability of an ontology in concrete application scenarios, because size and structure parameters constrain the type of tools and methods that are applied to aiding the reuse process. For instance, as it has been analyzed in the past (e.g., Gardiner et al. 2006), most ontology reasoners have still scalability issues when dealing with large ontologies. Furthermore, structural metadata provide core information to identify when an ontology changes (e.g., a different number of classes or individuals).
- *Descriptive metadata* relate to the domain modeled in the ontology in the form of keywords, topic classifications, textual descriptions of the ontology contents, etc. This type of metadata plays a crucial role in the selection of appropriate reuse candidates, a process that includes requirements with respect to the domain of the ontologies to be reused. Moreover, descriptive metadata are highly useful when identifying ontology changes from a high-level point of view (e.g., the domain has been specialized and the description has been updated).
- *Administrative metadata* provide information to help manage an ontology, such as when and how it was created, rights management, file format, and other technical information. Obviously, information like the date of modification of the ontology is also useful to identify when an ontology has changed.

Interoperability. Similar to the ontology it describes, metadata information should be available in a form that facilitates metadata exchange among applications. While the syntactical aspects of interoperability are covered by the usage of standard representation languages (see ‘Accessibility’), the semantic interoperability among machines handling ontology metadata information can be ensured by means of a formal and explicit representation of the meaning of the metadata entities (by conceptualizing the metadata vocabulary itself as an ontology).

4.2.1 OMV Overview

This section presents the ontology metadata vocabulary (OMV); the first part provides an overview of the core design principles applied to the development of the OMV metadata model; then, we describe in detail the core of such a model; next, we present implementation and practical aspects; finally, we provide an introduction to the OMV extensions.

4.2.1.1 Core and Extensions

Following the usability constraints identified during the requirements analysis, we decided to design the OMV schema modularly, distinguishing between the OMV core and various OMV extensions. The former captures information that is expected to be relevant to the majority of ontology reuse settings. However, in order to allow ontology developers and users to specify task-or application-specific ontology-related information, we allowed for the development of OMV extension modules, which are separated from the core schema while remaining compatible to it. That is, the terms are supposed to mean the same thing in the core and the extensions. Essentially, extensions reuse the core knowledge and provide specialized information for different ontology aspects.

4.2.1.2 Metadata Organization and Categorization

In the following, we present the organization and categorization of metadata (entities) in two dimensions, which provide a structured overview of the OMV ontology:

Property Appropriation. We organize metadata entities according to the impact on the prospected reusability of the described ontological content as presented in the following list:

- Required – mandatory metadata elements. Any missing entry in this category leads to an incomplete description of the ontology.
- Optional – important metadata facts, but not strongly required.

- Extensional – specialized metadata entities, which are not considered to be part of the core metadata schema.

Property Categorization. Orthogonal to the previous classification, we organize the metadata elements according to the type and purpose of the information contained as follows:

- General – elements providing general information about the ontology.
- Availability – information about the location of the ontology (e.g., its URI or the URL where the ontology is published on the web).
- Applicability – information about the intended usage or scope of the ontology.
- Format – information about the physical representation of the resource. In terms of ontologies, these elements include information about the representation language(s) in which the ontology is formalized.
- Provenance – information about the organizations contributing to the creation of the ontology.
- Relationship – information about relationships to other resources. This category includes versioning, as well as conceptual relationships such as extensions, generalization/specialization, and imports.
- Statistics – various metrics on the underlying graph topology of an ontology (e.g., number of classes).
- Other – information not covered in the categories listed above.

Note that the classification dimensions introduced above (appropriation and categorization) are intended to be considered when implementing several metadata support facilities. The first dimension is relevant for a metadata creation service, since it ensures a minimal set of useful metadata entries for each of the described resources. The second can be used in various settings, mainly to reduce the user-perceived complexity of the metadata schema, whose elements can be structured according to the corresponding categories.

4.2.1.3 OMV Core Metadata Entities

The main classes and properties of the OMV ontology are illustrated in Fig. 4.1[4].

Besides the main class Ontology, the metadata model contains elements describing various aspects related to the creation, management, and usage of an ontology. We will briefly discuss these in the following text. In a typical ontology engineering process, person(s) or organization(s) develop ontologies. We group these two classes under the generic class Party by a subclass-of relation. A Party can have several locations by referring to a Location individual and can create and contribute

[4] Please notice that not all classes and properties are included. The ontology is available for download in several ontology formats at http://omv.ontoware.org/

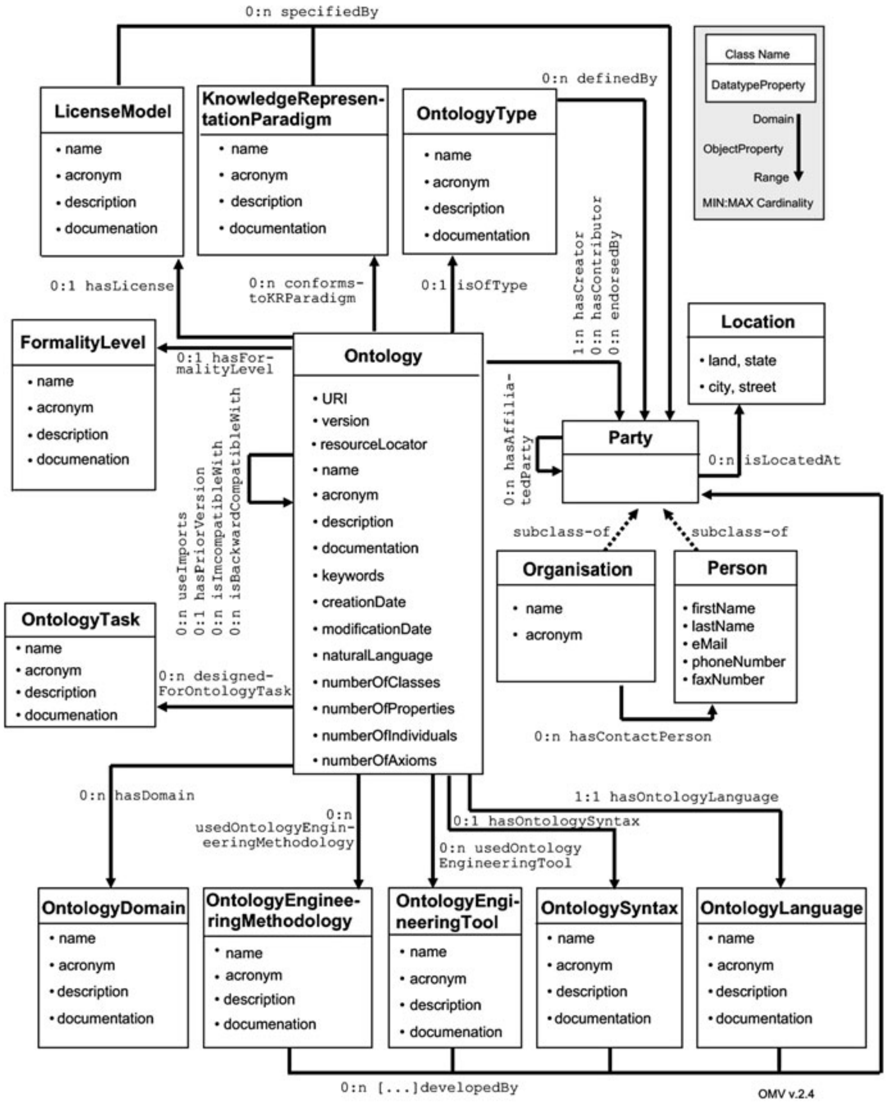


Fig. 4.1 OMV core overview

to ontological resources, i.e., Ontology class. Review details and further information can be captured in an extensional OMV module. Further on we provide information about the engineering process the ontology originally resulted from in terms of the classes `OntologyEngineeringMethodology`, `OntologyEngineeringTool`, and the attributes `version`, `status`, `creationDate`, and `modificationDate`. Again these can be elaborated as an extension of the core metadata schema. The usage history of the ontology is modeled by classes such as the OntologyTask and LicenceModel. The scheme also contains a representation of the most significant intrinsic features of an ontology. Details on ontology languages are representable with the help of the classes OntologySyntax, OntologyLanguage, and KnowledgeRepresentationParadigm. Ontologies might be categorized along a multitude of dimensions. One of the most popular classifications differentiates among application, domain, core, task, and upper-level ontologies. A further classification relies on their level of formality and types of Knowledge Representation (KR) primitives supported, introducing catalogs, glossaries, thesauri, taxonomies, frames, etc., as types of ontologies. The former categories can be modeled as individuals of the class OntologyType, while generic formality levels are introduced with the help of the class FormalityLevel. The domain the ontology describes is represented by the class OntologyDomain that references a predefined topic hierarchy such as the DMOZ hierarchy. Further content information can be provided as values of the attributes description, keywords, and documentation. Moreover, the metadata schema provides information about the imported ontologies (useImports) and versioning relations (hasPriorVersion, isBackwardCompatibleWith, and isIncompatibleWith) – analogously to the OWL ontology properties. Finally, OMV gives an overview of the graph topology of an Ontology with the help of several graph-related metrics represented as integer values of the attributes numberOfClasses, numberOfProperties, numberOfAxioms, and numberOfIndividuals.

4.2.1.4 Ontological Representation

Following the accessibility and interoperability requirements, as well as the nature of the metadata, which are intended to describe ontologies, the conceptual model designed in the previous steps was implemented in OWL2[5]. With OWL being established as the standard to represent ontologies, it was only logical to opt for representing ontology metadata using the same language. As a consequence, the same tooling for processing the ontologies can also be used for processing the ontology metadata.

Additionally, a metadata element is modeled either by means of classes and individuals or by means of valued properties. The former alternative, represented using additional classes linked by object properties, was chosen to model those metadata elements representing entities that can be referred to. The latter alternative, represented using datatype properties, was chosen to model metadata elements with value/content that can be easily mapped to conventional data types (numerical, literal, list values).

[5] In the remainder of this chapter, when OWL appears without any version information, it refers to OWL1. As opposed, when referring to OWL2, we explicitly note it.

Finally, OMV implements the appropriate properties of metadata entities by different means: The required and optional metadata entities are implemented in OMV core with the appropriate cardinality restrictions, while the extensional metadata entities are implemented in the different OMV extensions.

4.2.1.5 OMV Extensions

The OMV core metadata is intended to evolve toward a commonly agreed schema for Semantic Web ontologies. In contrast to this ambitious goal, we are aware that for specific domains, tasks, or communities, extensions in any direction might be required. These extensions should be compatible to the OMV core, but at the same time, they should fulfill the requirements of a domain-, task-, or community-driven setting.

The character of an OMV extension is a metadata ontology itself that imports the OMV core ontology. There are no restricting modeling guidelines to be met. However, developers are encouraged to follow the design principles described above (see Sects. 4.2.1.2 and 4.2.1.3), as well as to follow a basic set of guidelines for naming ontology terms (Palma et al. 2008).

Some of the existing OMV extensions were developed in collaboration with different institutions. The available extensions[6] are: the generic change ontology, which models changes to an ontology (Palma 2009); the lexOMV extension (Montiel-Ponsoda et al. 2007) that models the linguistic or multilingual data contained in the ontology; the modules extension that represents the description of ontology modules (d'Aquin et al. 2008); the peer extension that captures information of peers sharing metadata about ontologies and related entities (e.g., mappings and modules) (Wang et al. 2007); and the mapping extension that describes mappings between heterogeneous ontologies.

4.2.2 Uses and Benefits

OMV plays an important role in the ontology reuse task by facilitating the discovery and exchange of ontologies, fostering the widespread dissemination of ontology-driven technologies and the development of full-fledged ontology repositories and registries on the web. Furthermore, applications that work with the creation or (re)use of ontologies can benefit from having a standard schema for ontology metadata. By using the same vocabulary to describe ontology metadata, applications can exchange this information easily.

[6] OMV extensions are also available at http://omv.ontoware.org

Several applications are already using OMV to describe ontology metadata. In this section, we present a selection of these applications that use OMV at various stages of the ontology development life cycle. First, the NeOn Toolkit[7] (c.f. Part III of this book) includes a set of OMV-related plugins that either use OMV or provide access to OMV-enabled registries (e.g., Oyster, Centrasite). Oyster[8] is an open-source ontology registry that uses the metadata for retrieval and selection tasks and can also export OMV data for other applications. Similarly, the commercial registry Centrasite[9] provides an OMV-specialized web service to support the management of ontology metadata. Also, BioPortal[10] and Cupboard[11] are two ontology repositories that use OMV for the description of ontologies. The Semantic Web gateway Watson[12] can generate OMV annotations for the ontologies discovered. Finally, applications such as the Protege MetaAnalysis plugin[13] allow to calculate various metadata for ontologies and facilitate the export of that metadata to the OMV.

Oyster (Palma and Haase 2005) is a distributed registry that exploits Semantic Web techniques in order to provide a solution for exchanging and reusing ontologies and related entities (e.g., ontology developers, ontology mappings, ontology changes, etc.). To achieve this goal, Oyster uses OMV to describe ontologies and related entities.

Moreover, Oyster uses ontologies extensively to provide its main metadata management functions (registry metadata, formulating queries, routing queries, and processing answers). The ontology metadata entries are aligned and formally represented according to two ontologies: (1) the OMV that describes the properties of the ontology and (2) a topic hierarchy to define the domain of the ontology (c.f. Sect. 4.3).

NeOn Applications. The NeOn Toolkit includes different OMV-related plugins. The Oyster-API plugin enables programmatic access to all Oyster registry functionalities within any other NeOn Toolkit plugin. This plugin can either use a local or remote Oyster instance. Similarly, the Oyster-GUI plugin provides a graphical user interface to interact with Oyster servers and other OMV-enabled servers (e.g., Centrasite) implementing the OMV-based web service. This plugin allows submitting, updating, and removing instances of OMV core classes, submitting queries to search ontologies based on different criteria and importing from the Internet ontologies matching the search criteria. Furthermore, the change-capturing plugin implements methods and strategies for the capturing and synchronization of ontology changes that are formally represented as instances of the change ontology (an OMV

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

[8] http://oyster2.ontoware.org

[9] http://www.infoq.com/zones/centrasite/

[10] http://bioportal.bioontology.org

[11] http://cupboard.open.ac.uk:8081/cupboard

[12] http://watson.kmi.open.ac.uk/

[13] http://protegewiki.stanford.edu/wiki/MetaAnalysis

extension). This OMV extension is also used by several other plugins, such as Cicero plugin (to enable discussions on changes), Evolva plugin (to represent the changes proposed by the plugin based on background knowledge) and GATE Web service plugin (to represent the changes generated from textual sources).

Additionally, the Cupboard system produced in NeOn for ontology publishing, sharing, and reuse also relies on OMV to implement some of its features. Besides letting users add their ontologies in a personal space – hosting, indexing, linking, and exposing them through APIs and SPARQL – Cupboard is designed to be a community tool. It helps ontology users and practitioners (including ontology developers) in finding and reusing ontologies, through the use of rich ontology metadata (thanks to Oyster and OMV) and advanced ontology review mechanisms.

Finally, the latest update produced in NeOn of the collaborative ontology design ontology (C-ODO), called *codolight* (c.f. Sect. 4.4), has been aligned with OMV. Compared to the original C-ODO ontology design metamodel, *codolight* is now linked to requirements and application tasks, has been used for tool descriptions, is aligned to external vocabularies, is lighter in complexity, and improves association between the social and software layers of ontology design aspects. From a design viewpoint, the metadata provided by OMV have a semantics that is potentially compatible to that of other metamodels, and this alignment helps with metadata interoperability.

Protege Plugin. The Protege MetaAnalysis plugin calculates various metadata for ontologies and facilitates the export of those metadata to the OMV. The plugin is a tab widget consisting of four panels: the numbers panel, the design panel, the OWL panel, and the extras panel. The plugin computes metadata for a given ontology and displays them in these panels. The ontology metadata can be exported to an extension of the OMV. If the ontology already exists in OMV, the metadata for that ontology are updated. Otherwise, a new instance of the ontology is created in OMV and populated with the computed metadata.

BioPortal. While Oyster is a distributed ontology repository, BioPortal is a centralized repository of biomedical ontologies, where authors submit their ontologies. As part of the submission process, authors also fill in the form to describe their metadata. In the future, it is planned to add the capability for the authors simply to point to the location of an OWL file that has the OMV individuals and to have BioPortal import the information from that file.

BioPortal uses the ontology metadata in ontology search and navigation. Users can specify, e.g., whether they want their search term to appear only in concept definitions or in metadata as well. BioPortal will also use OMV extensions. For example, one of the functions of BioPortal is to be a repository of mappings between concepts in biomedical ontologies. Each mapping comes with its own set of metadata (e.g., the mapping author, the algorithms used, the application context in which the mapping is valid, etc.) (Fridman Noy et al. 2008). It is planned to represent the mapping metadata as an OMV extension.

Another feature of BioPortal is the use of peer reviews for ontology evaluation. Ontology users can rate BioPortal ontologies along different dimensions, such as coverage and degree of formality, based on their experience with the ontology in their own applications (Fridman Noy et al. 2005). The evaluation extension will also be an OMV extension.

The *Watson Semantic Web gateway* contains a repository of ontologies and provides export of their metadata in the OMV format. When Watson users search for ontologies, they can click on an ontology URI from the search results, and then on the overview page for that, click on "Get OMV" for the metadata export.

OMEGA is an algorithm that addresses the problem of populating metadata elements (Ungrangsi and Simperl 2008). It generates automatically metadata about arbitrary ontologies on the web and is available as a web application and a REST web service. It takes an ontology as input and automatically populates certain metadata information such as domain, level of formality, and statistics, using an ontology metadata schema, which is part of the OMV standard.

4.3 Linguistic Information Repository (LIR)

The symbiosis between ontologies and natural language has proven more and more relevant in the light of the growing interest and use of Semantic Web technologies. Ontologies that are well-documented in a natural language not only provide humans with a better understanding of the world model they represent, but also a better exploitation by the systems that may use them. This "grounding in natural language" is believed to provide improvements in tasks such as ontology-based information extraction, ontology learning, and population from text or ontology verbalization (Buitelaar et al. 2009).

Nowadays, there is a growing demand for ontology-based applications that need to interact with information in different natural languages, i.e., with multilingual information. This is the case of numerous international organizations currently introducing semantic technologies in their information systems, such as the Food and Agriculture Organization or the World Health Organization, to mention just a few. Such organizations have to manage information and resources available in more than a dozen of different natural languages and have to customize the information they produce to a similar number of linguistic communities.

For all these reasons, solutions have to be provided to model multiple natural language descriptions in ontologies. Such an undertaking needs to consider several requirements imposed by the characteristics of the domain of knowledge modeled in the ontology and by the type of linguistic descriptions that are required by the final application.

Requirements. Although the number of multilingual ontologies is still quite small compared with the total amount of ontologies available in the web[14], we have conducted a survey of the state of the art of modeling options to represent multilingual information in ontologies (Montiel-Ponsoda et al. 2010). This survey

[14] The Semantic Web search engine Watson provides data about the language of ontology labels that shows that around 80% of ontologies have literals only in English (http://watson.kmi.open.ac.uk/blog/2007/11/20/1195580640000.html)

has revealed the existence of three modeling options, which are briefly explained in the following:

- Including multilingual labels in the ontology model
- Combining the ontology model with a mapping model between different natural languages or a common interlingua
- Associating the ontology model with an external linguistic model

The first modeling option relies on the RDF(S) and OWL properties `rdfs:label` and `rdfs:comment` to associate word forms and descriptions to ontology elements. The main disadvantage of this option is that it is not possible to define any relation among the linguistic annotations, so that the linguistic information is restricted and the model is difficult to scale. The second option assumes the existence of several ontologies in the same domain with labels expressed in different natural languages, which are mapped to each other in a pairwise fashion, or through a common conceptualization or interlingua. This option has been considered in projects such as *EuroWordNet* (Vossen 1998). The risk of this option is that a lot of effort has to be put in developing one conceptualization per language and also in establishing mappings or links among conceptualizations. Finally, the third modeling option allows the association of an external model of linguistic descriptions to the ontology. The main advantages of this modeling option have to do with the capability of the linguistic model of evolving into a complex model of linguistic descriptions that can be accommodated to account for the needs of the final applications. Models that follow this approach include *LingInfo* (Buitelaar et al. 2006), *LexOnto* (Cimiano et al. 2007) or *LexInfo* (Buitelaar et al. 2009).

Whereas some models have been explicitly designed to enrich ontologies with linguistic information, such as the ones mentioned above, they mainly focus on morphosyntactic descriptions of ontological entities and have not handled multilingualism issues, as the ones that arise when aiming at reusing the same ontology in different linguistic and cultural settings. This is particularly relevant in the case of ontologies that represent categorizations of reality that are not completely valid for all the cultures and languages involved. In this context, we have to consider the possibility of providing relations among the linguistic descriptions in different languages associated to the same ontology elements.

Finally, we refer to the need for encoding the linguistic descriptions captured in the linguistic model according to standard models in order to guarantee interoperability, reuse, and commitment to best practices. The potential integration of terminological and lexical knowledge bases into our model requires interoperability with existing and proposed standards. In this sense, we have analyzed some standardization initiatives that have been developed in order to capture linguistic information that can be reused for various purposes. As the most important initiatives, we mention a number of standards from the International Organization for Standardization (ISO) and the World Wide Web Consortium (W3C) that capture terminological and lexical information. We are referring to Terminological Markup Framework (TMF) (ISO 2003), the Lexical Markup Framework (LMF) (ISO 2006), and Simple Knowledge Organization Systems (SKOS) (Miles et al.

2005). After the analysis of the state of the art and considering the needs of a model that aims at providing ontologies with multilingual information, we identified the following set of requirements:

Independence: the possibility for providing independent and complex models of linguistic information that can be self-contained and from which information can be inferred. The independence between the ontology and the linguistic model guarantees the full development of both without one restricting the other. In particular, in the case of the linguistic model, this allows the existence of a complex model that contains as much linguistic information as required by the final application and, additionally, in different languages.

Localization: the capability for providing a subset of linguistic descriptions to account for the linguistic realization of an ontology in different natural languages and representing term variants within one language and cultural specificities among different languages.

Interoperability: the flexibility of interoperating with existing standards for the representation of lexical and terminological information. By interoperating with standard models, there also exists the possibility for the model of interchanging knowledge with the standards and being extended with further linguistic description elements, if so required by the final application.

Accessibility: the fact of being implemented in a syntax or representation language that can provide tool support available to manage it, as well as access to external resources from which information can be obtained to semi-automatically support the model.

4.3.1 LIR Overview

This section presents the *Linguistic Information Repository* or LIR, a model that has been created with the twofold purpose of fulfilling the needs of portability and association of multilingual information to domain ontologies, on the one hand, and adapting ontologies to the needs of the languages involved in the localization activity, on the other.

The LIR has been implemented as an ontology in OWL. Its main purpose is not to provide a model for a lexicon of a language but to cover a subset of linguistic description elements that account for the linguistic realization of a domain ontology in different natural languages. A complete description of the current version of the LIR can be found in (Montiel-Ponsoda et al. 2008; Montiel-Ponsoda 2011).

The lexical and terminological information captured in the LIR is organized around the `LexicalEntry` class. Lexical entry is considered a union of word form (`Lexicalization`) and meaning (`Sense`). This ground structure has been inspired by the Lexical Markup Framework (LMF). The compliance with this standard is important for two main reasons: (a) Links to lexicons modeled according to this standard can be established, and (b) the LIR can be flexibly

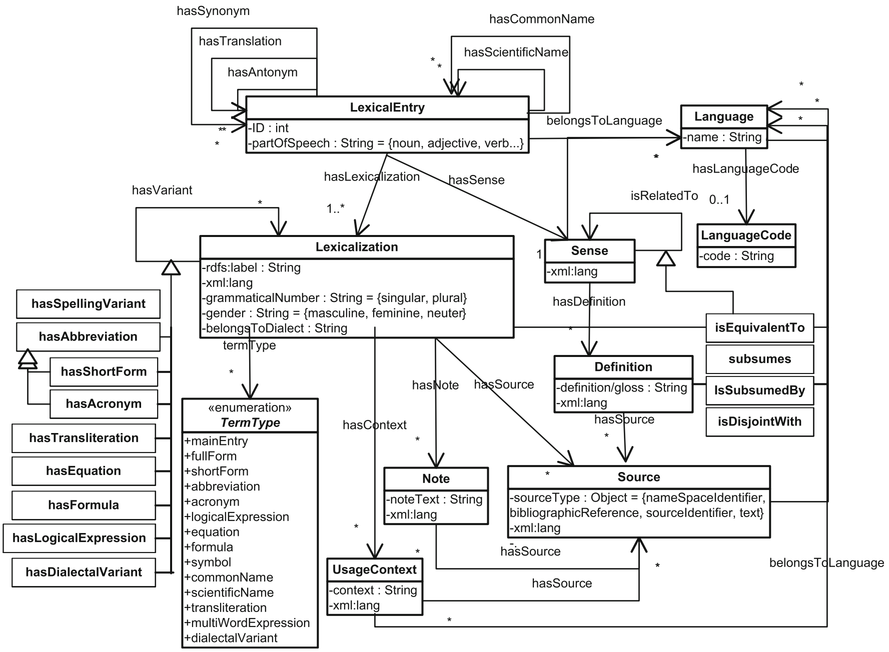


Fig. 4.2 Diagram of LIR ancillary classes

extended with modular extensions of the LMF (or standard-compliant) modeling specific linguistic aspects, such as deep morphology or syntax, not dealt by LIR in its present stage. For more details on the interoperability of the LIR with further standards see (Peters et al. 2010).

The rest of the classes that make up the LIR are `Language`, `Definition`, `Source`, `Note`, and `UsageContext` (see Fig. 4.2). These can be linked to the `Lexicalization` and `Sense` classes. Each lexicalization is associated to one sense. The sense class represents the meaning of the ontology concept in a given language. It has been modeled as an empty class because its purpose is to point to other resources in which that sense is captured. The meaning of the concept in a certain language (which may not completely overlap with the formal description of the concept in the ontology) is "materialized" in the definition class, i.e., is expressed in natural language. The `UsageContext` gives us information about how a word behaves syntactically in a certain language by means of examples. Source information can be attached to any class in the model (`Lexicalization`, `Definition`, etc.), and, finally, the `Note` class has been meant to include any information about language specificities, connotations, style, register, etc., and can be related to any class. By determining the `Language` of a lexical entry, we can ask the system to display only the linguistic information associated to the ontology belonging to a given language.

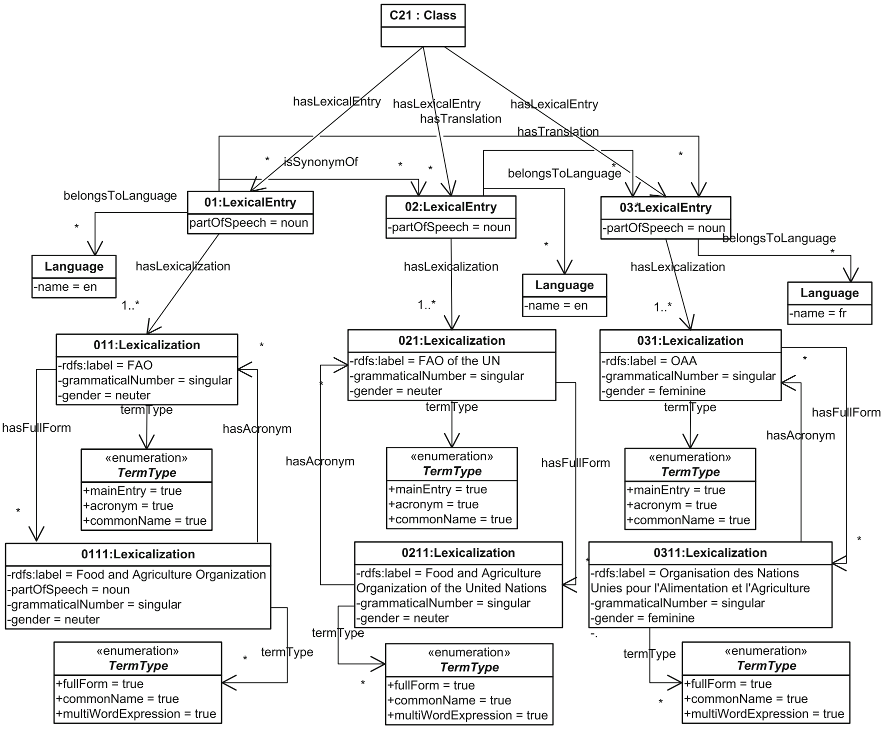


Fig. 4.3 LIR example usage within a single language

4.3.2 Uses and Benefits

The main benefit of the LIR model is that it provides a very granular specification of relationships between elements of an ontology. In particular, it identifies well-defined relationships among the linguistic descriptions used to represent ontological concepts, specifically:

- Well-defined relations within lexicalizations in one language
- Well-defined relations within lexicalizations across languages

Both cases are illustrated in the following. The example in Fig. 4.3 concerns the establishment of relations among term variants belonging to the same language. Specifically, this case exemplifies the use of various acronyms and full forms attached to one and the same concept. Three lexical entries (01:LexicalEntry, 02: LexicalEntry, and 03:LexicalEntry) are associated with the same concept (C21: Class), which means that they are terms that identify one and the same concept. Two lexical entries (01:LexicalEntry and 02:LexicalEntry) belong to English, whereas the third lexical entry (03:LexicalEntry) belongs to French. The two English lexical entries are considered synonyms, and both are translations of the

French lexical entry. Each lexical entry contains two lexicalizations. For example, 01:LexicalEntry includes 011:Lexicalization and 0111:Lexicalization, whose labels are FAO and Food and Agriculture Organization, respectively. FAO is the acronym for Food and Agriculture Organization, and, moreover, it is considered the main entry. FAO of the UN and Food and Agriculture Organization of the United Nations are deemed synonyms of FAO and Food and Agriculture Organization. Both lexical entries (01:LexicalEntry and 02:LexicalEntry) are translations of OAA and Organisation des Nations Unies pour l'Alimentation et l'Agriculture in the French language. Thanks to LIR it is possible to retrieve synonyms within the same language associated with the same concept and distinguish different term types such as acronyms and full forms.

The second example highlights the possibility given by the LIR model to represent scientific names and use them across languages (scientific names are in Latin and are internationally accepted over scientific communities). Variants in the same language (e.g., Buffaloes (syncerus)) can therefore be connected to the same scientific term, such as the English and Japanese translations. We have illustrated in Fig. 4.4 how the concept buffaloes (C133:Class) has four lexical entries associated (01:LexicalEntry, 02:LexicalEntry, 03:LexicalEntry, and 04:LexicalEntry). Two of them belong to the English language and contain synonymous lexicalizations (011: Lexicalization and 021:Lexicalization).

Then, we have a lexicalization in Latin that represents the scientific name, and it is accordingly related with the rest of lexical entries by means of the object property `hasScientificName`. Finally, 04:LexicalEntry belongs to the Japanese language, which is also the common denomination in Japanese of the *Syncerus caffer* scientific name and, at the same time, the translation of the two lexicalizations in English.

To conclude, we refer to the *LabelTranslator* NeOn plugin, a translation-supporting tool (Espinoza et al. 2008) that provides semi-automatically translations

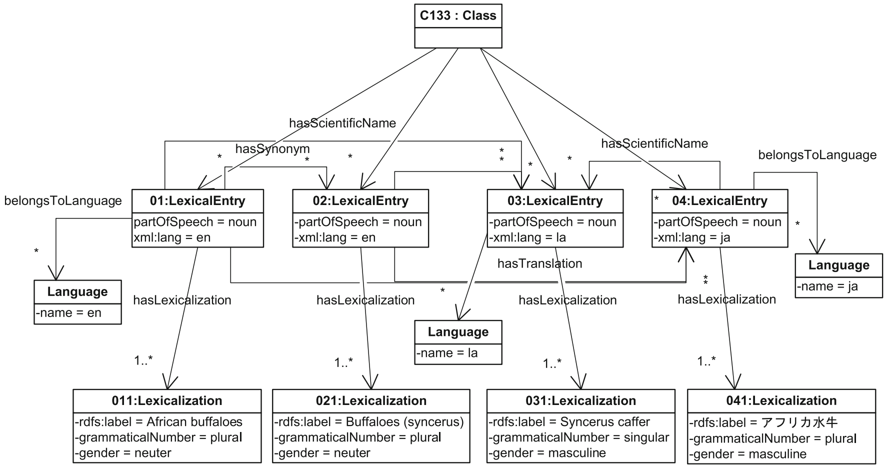


Fig. 4.4 LIR example of cross-language usage

for ontology lexicalizations. Currently, the languages supported by the plugin are Spanish, English, and German. Once translations are obtained for the labels of the original ontology, they are stored in the LIR. However, if the system does not support the language combination in which we are interested, we can still use this system to take advantage of the LIR application programming interface or API implemented in the NeOn Toolkit. In this sense, the needed linguistic information can be introduced manually.

4.4 Collaborative Ontology Design Ontology (C-ODO) Light

Authoring and maintaining Semantic Web ontologies is generally not an individual, monolithic activity but is intrinsically grounded on social and collaborative processes, more so when ontologies are configured in a networked architecture. Continuous interaction between knowledge engineers and domain experts is key and so is that with resource providers whenever reuse or re-engineering enters the life cycle.

However, without dedicated tool support, collaboration may occur across general-purpose software tools and communication channels, in which case a manual effort is required to coordinate and bring the outcome of these activities together. Thus, on one hand, tool support to ontology engineering activities (e.g., *reusing* existing ontologies and design patterns; *re-engineering* thesauri, lexica, and database schemas; *validating* the outcome) is required. On the other hand, tools are often unable to support these activities in a *collaborative* setting, e.g., aiding the discussion and consensus-based assessment of an ontology element and the rationale behind it. Among other reasons, this can also be ascribed to an inadequate requirement analysis describing the actual processes and data involved therein, and the lack of a conceptual framework that formally expresses these notions so that they can be unified and reasoned upon.

4.4.1 C-ODO Light Overview

C-ODO Light (aka *codolight*) is one such formal knowledge framework. It is a pattern-based OWL-DL ontology network that provides a metamodel for describing collaborative ontology projects (Gangemi et al. 2007). C-ODO Light was designed so as to take into account requirements deriving from the experience with formal models for describing ontology projects and tools, as well as existing controlled vocabularies.

In particular, the network displays the following features:

1. The ability to formalize ontology design tool descriptions in terms of input/output data (knowledge types), functionalities, interface objects, and interaction patterns

2. Smooth integration between human-oriented and tool-oriented descriptions of ontology design aspects
3. Alignment to existing vocabularies such as DOAP, OMV, etc.
4. Light axiomatization, e.g., no use of anonymous classes in restrictions
5. Modular development by pattern-based design (cf. Chap. 3), in compliance with the `ontologydesignpatterns.org` practices

Additionally, the *codolight* core is extended to support specific ontology application tasks, such as:

1. Browsing semantic data about ontology projects, tools, data, repositories, solutions, discussion, evaluation, etc.
2. Searching and selecting design components based on design aspects, knowledge types, individual needs, user profiles, etc.
3. Creating design configuration interfaces that aid or automate task 2
4. Help collecting ontology requirements, design functionalities, and ontology application tasks for an ontology project
5. Providing a shared network of vocabularies to create/query/reason on annotations and data related to ontology projects, including integration between annotations of heterogeneous provenance, such as those coming from collaborative discussions and change

It is here anticipated that part of these tasks are implemented within NeOn in the form of the *Kali-ma* tool, to be described in Chap. 15.

4.4.2 Structure

The C-ODO Light network of ontologies is organized as a layered architecture, where these layers are connected with different types of bindings. Besides the two bottom layers that define the main structure, there are three additional layers that bridge it with existing applications, vocabularies, and functionalities:

Pattern layer: It contains reusable content ontology design patterns (Content ODP) (Presutti and Gangemi 2008) that include, e.g. *sequence*, *partof*, *situation*, *collectionentity,* and so on. The patterns are reused in the design of the ontologies constituting the core architecture of *codolight*.

Core codolight layer: It contains the nine modules of the *codolight* core network of ontologies, centered around the *codkernel* module in the center, along with modules *coddata*, *codprojects*, *codworkflows*, *codarg*, *codsolutions*, *codtools*, *codinterfaces*, and *codinteraction* importing *codkernel*.

Plugin layer: It consists of the modules containing the descriptions of the NeOn Toolkit plugins related to ontology design, formalized in OWL by reusing the codolight vocabulary and some of the alignment modules.

Categorization layer: It consists of the modules containing the definition of the design aspects according to which tools, knowledge types, and functionalities are organized, as well as the closure of inferences derived by the application of reasoners to the previous layers.

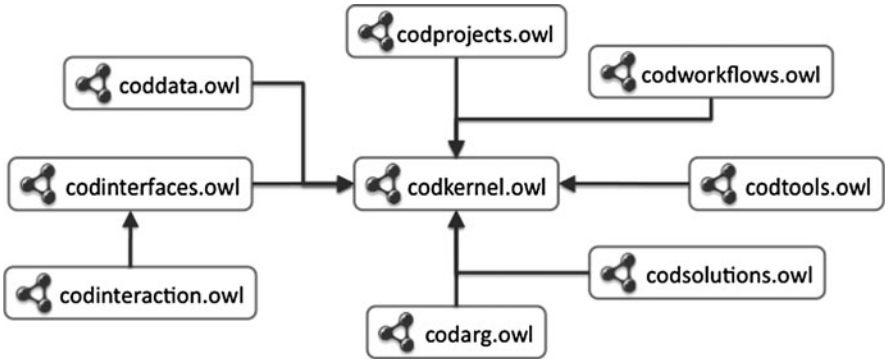


Fig. 4.5 Core C-ODO Light corolla architecture

Alignment layer: It consists of the modules containing mapping axioms between *codolight* and related vocabularies, currently: OMV, DOAP, FOAF, NeOn Access Rights model, NeOn OWL metamodel, NeOn OWL2 metamodel, and the Software Ontology Model.

Each layer is in its turn an ontology network with its own architecture. In particular, the core *codolight* layer network encodes the main aspects of ontology design by following an architectural ontology design pattern called *corolla*. The floral metaphor for the corolla pattern, shown in Fig. 4.5, suggests an overall shape for the network composed of a *kernel* module, which includes the definition of core concepts of the domain of interest, and a set of *petal* modules, each defining a specific aspect of the same domain.

The corolla pattern minimizes dependencies and enforces loose coupling between the modules of an ontology network: In the *codolight* core example, all modules, but *codinteraction* (described below), directly depend on the kernel module exclusively. Also, the modules are built so that their structure suggests an organization of the network, by which different aspects of the domain of interest are represented by each petal module. The criterion by which an ontology network can be broken apart into a corolla can be: *What are the main aspects of the domain described by the ontology network?*

The kernel module defines core concepts, shared by all aspects. As shown in Fig. 4.6, axiomatization is minimal at the kernel level, i.e., although the basic classes of collaborative ontology design are defined, only a minimum set of new properties, or restrictions holding between them, is asserted. It is up to petal modules to refine the axiomatization of at least one of the core concepts each, thus adding details for at least one aspect.

All classes defined in the kernel module are specializations of classes from the pattern layer (such as `DesignFunctionality` subsuming `Task` from the `taskrole` pattern[15] or `KnowledgeResource`, `FormalExpression`,

[15] http://www.ontologydesignpatterns.org/cp/owl/taskrole.owl

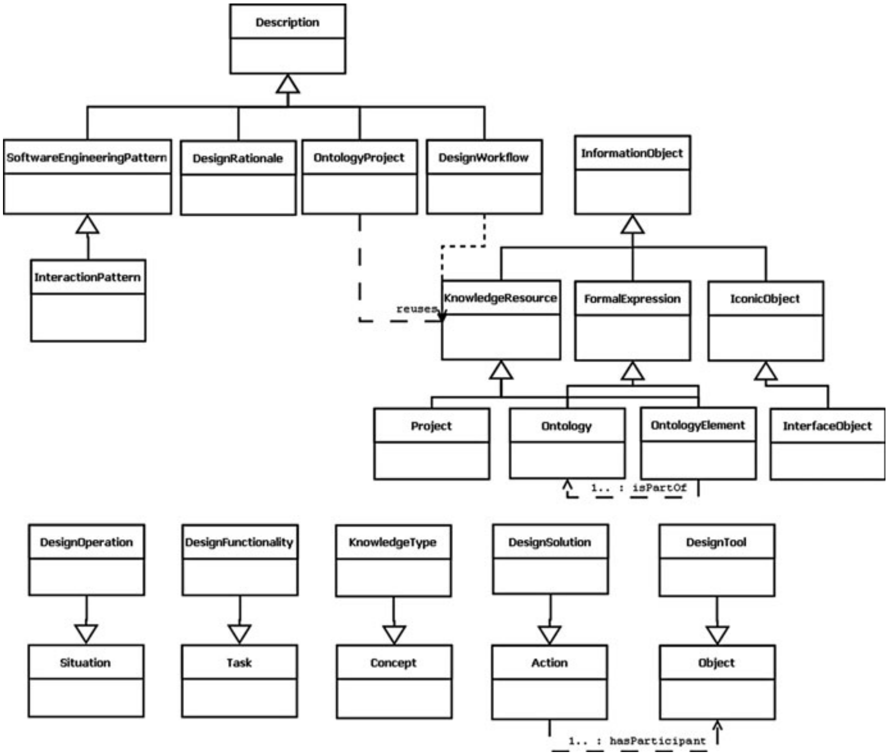


Fig. 4.6 C-ODO Light kernel class diagram

`IconicObject` all subsuming `InformationObject` from the `intensionextension` pattern[16]), thus inheriting the structure defined by equality over shared classes.

The following petal modules are defined for *codolight*:

Data (*coddata*)*:* contains the main notions that classify the data managed when designing an ontology: ontologies, ontology elements, Knowledge Organization Systems (KOS), KOS elements, rules, modules, encoding syntaxes, and more. For each class of knowledge resources, a `knowledge-type` instance is provided.

Projects (*codprojects*): contains the minimal vocabulary for representing ontology design projects and their executions. An ontology project is here taken as a social entity, whose computational counterpart (e.g., a project created in the NeOn Toolkit) is a software entity that collects resources and descriptions related to an ontology project.

Workflows (*codworkflows*): contains classes and properties to represent workflows from within ontology projects: collaborative workflows, accountable agents, need for an agent or a design functionality, etc.

[16] http://www.ontologydesignpatterns.org/cp/owl/intensionextension.owl

Argumentation (*codarg*): contains the basic classes and properties to represent argumentation concepts: arguments, threads, ideas, positions, rationales, etc.

Solutions (*codsolutions*): contains classes and properties to represent ontology design solutions: competency questions, ontology design patterns, ontology requirements, unit tests, etc.

Tools (*codtools*): contains classes and properties to represent ontology design tools: tools, pieces of code, code entities, computational tasks, input and output data relations, etc.

Interfaces (*codinterfaces*): contains classes and properties that represent some typical user interface entities, such as interface objects, panes, and windows.

Interaction (*codinteraction*): contains classes and properties that represent some typical entities related to human-computer and human-ontology interaction, e.g., user types, computational tasks, and workflows. These can in turn be combined so as to construct interaction pattern models.

One advantage of employing this aspect-oriented architecture is *selective extensibility*. A software application intended to exploit only a given subsystem of ontology life cycle management, such as reasoning on the usage of interaction patterns and user interface widgets, can import only the *codinterfaces* and *codinteraction* modules. Possibly, they can also extend these modules with ontologies that model further additional interaction patterns or GUI elements originally not intended for the application domain at hand.

4.4.3 Alignments

This section provides an insight on some alignments that hold between *codolight* and other vocabularies that are widely used on the Semantic Web or are introduced as part of the methodology described by this book (c.f. Chap. 2), such as OMV that is described in this very chapter.

OWL[17]. The alignments between *codolight* and the OWL language constructs consider three different vocabularies:

- The original RDF, RDFS, and OWL vocabularies from the W3C
- The OWL1 metamodel designed for NeOn
- The OWL2 metamodel also designed for NeOn

The reason why so many different vocabularies represent entities from a same language is mainly due to the pragmatic evolution of semantic technologies. The original vocabularies by W3C are not extremely detailed in distinguishing the constructs available in OWL (and RDF, RDFS); e.g., it is difficult to describe existential restrictions explicitly, because these are just instances of `owl:Restriction`. On the other hand, W3C vocabularies are implemented in all APIs and tools for ontology engineering, so in order to maximize interoperability,

[17] http://www.ontologydesignpatterns.org/cpont/codo/owl22codo.owl

an ontology design vocabulary like *codolight* must be aligned to the main data vocabularies. The OWL metamodels developed in NeOn try to overcome the referential coarseness of OWL constructs, e.g., by providing the class `owlodm1:ExistentialRestriction`. On the other hand, these metamodels are not intended to be a replacement for the W3C OWL data model.

Ontology Metadata Vocabulary (OMV)[18]. The OMV described in this chapter is a vocabulary for annotating ontologies with time, authors, tools, languages, etc., and it is used to provide support for ontology registries. However, from a design viewpoint, the semantics of OMV metadata are potentially compatible with those of other metamodels, so this alignment aids metadata interoperability. Only subsumption alignments occur, i.e., OMV classes and properties are subclasses or subproperties of those from the *codolight* pattern and core layers. Aligned OMV classes of interest include `omv:OntologySyntax`, `omv:FormalityLevel`, `omv:OntologyEngineeringTool`, and `omv:OntologyTask`. Aligned OMV properties of interest include `omv:hasContributor`, `omv:hasCreator`, `omv:useImports`, and `omv:isIncompatibleWith`.

Description of a Project (DOAP)[19]. DOAP is a vocabulary for creating profiles of software projects[20] with time, authors, FOAF (Friend of a Friend) vocabulary profiles, etc. The `doap:Project` notion addressed here is computational and not social; therefore, it has been aligned as equivalent to `codkernel:Project`. Subsumption mappings occur between `doap:Repository` and `collectionentity:Collection`, and between `doap:Version` and `coddata:Annotation`. Direct mappings occur between FOAF and the *codolight* pattern layer, either by equivalence (`foaf:Agent` and `agentrole:Agent`) or by subsumption (`foaf:topic` and `topic:hasTopic`; `foaf:Document` and `intensionextension:InformationObject`; `foaf:member` and `collectionentity:hasMember`).

NeOn Access Rights Model[21]. This ontology representing access control policies and related entities uses three different vocabularies, i.e., *accessRights*[22], *ar-entities*[23], and *ar-agents*[24]. Entities from these vocabularies, such as `Action`, `Agent,` and `Content`, map to *codolight* via subsumption alignments. The `Right` class, being a conceptualization of its holder in the access policies context, subsumes the `Description` class from the `description` design pattern.

Software Ontology Model (SOM)[25]. The SOM is designed to represent entities in the object-oriented programming model (Tappolet et al. 2010). Given the class

[18] http://www.ontologydesignpatterns.org/cpont/codo/omv2codo.owl

[19] http://www.ontologydesignpatterns.org/cpont/codo/doap2codo.owl

[20] http://trac.usefulinc.com/doap

[21] http://www.ontologydesignpatterns.org/cpont/codo/accessrights2codo.owl

[22] http://www.uni-koblenz.de/~bercovici/owl/2008/7/accessRight.owl

[23] http://www.uni-koblenz.de/~bercovici/owl/2008/7/entity.owl

[24] http://www.uni-koblenz.de/~schwagereit/owl/agents.owl

[25] http://www.ontologydesignpatterns.org/cpont/codo/som2codo.owl

subtree for the `som:Entity` class, which includes functions, methods, classes, and packages, the parent class aligns to `codtools:CodeEntity` by equivalence.

4.5 Conclusions

A methodology for managing ontology networks is best designed if formal models of the resources and processes involved come along with it. To that end, the NeOn Methodology proposes three stand-alone models, i.e., the OMV, LIR, and C-ODO Light ontologies, that can nonetheless be interconnected in order to represent and reason on the structural, linguistic, and engineering aspects of ontology life cycle. Ontology alignments are provided across these models to ensure logic interoperability, without hampering their stand-alone usage possibilities. These ontologies also serve as a back end for software applications provided as plugins for the NeOn Toolkit (see Chaps. 13, 14, 15), which treat each ontology separately to serve dedicated phases and processes in ontology management.

References

Arpírez J, Gómez-Pérez A, Lozano-Tello A, Pinto HS (2000) Reference ontology and (ONTO) 2 agent: the ontology yellow pages. Knowl Inf Syst 2:387–412

Buitelaar P, Declerck T, Frank A, Racioppa S, Kiesel M, Sintek M, Engel R, Romanelli M, Sonntag D, Loos B, Micelli V, Porzel R, Cimiano P (2006) Linginfo: design and applications of a model for the integration of linguistic information in ontologies. In: Proceedings of the OntoLex 2006 workshop: interfacing ontologies and lexical resources for semantic web technologies, Genoa

Buitelaar P, Cimiano P, Haase P, Sintek M (2009) Towards linguistically grounded ontologies. In: Proceedings of the 6th annual European semantic web conference (ESWC2009), Heraklion, pp 111–125

Cimiano P, Haase P, Herold M, Mantel M, Buitelaar P (2007) LexOnto: a model for ontology lexicons for ontology-based nlp. In: Proceedings of the OntoLex07 workshop at the ISWC07, Busan

d'Aquin M, Haase P, Rudolph S, Euzenat J, Zimmermann A, Dzbor M, Iglesias M, Jacques Y, Caracciolo C, Buil-Aranda C, Gómez-Pérez J (2008) NeOn formalisms for modularization: syntax, semantics, algebra. Technical report D1.1.3, Open University

Espinoza M, Gómez-Pérez A, Mena E (2008) Enriching an ontology with multilingual information. In: Proceedings of the 5th annual of the European semantic web conference (ESWC 2008), Tenerife, pp 333–347

Fridman Noy N, Guha RV, Musen MA (2005) User ratings of ontologies: who will rate the raters? In: Proceedings of the AAAI 2005 spring symposium on knowledge collection from volunteer contributors, Stanford, CA, USA

Fridman Noy N, Griffith N, Musen MA (2008) Collecting community-based mappings in an ontology repository. In: Proceedings of 7th international semantic web conference´08. Springer, Karlsruhe

Gangemi A, Pisanelli DM, Steve G (1999) An overview of the ONIONS project: applying ontologies to the integration of medical terminologies. Data Knowl Eng 31(2):183–220

Gangemi A, Lehmann J, Presutti V, Nissim M, Catenacci C (2007) C-ODO: an OWL meta-model for collaborative ontology design. In: Fridman Noy N, Alani H, Stumme G, Mika P, Sure Y, Vrandecic D (eds) CKC, CEUR-WS.org

Gardiner T, Horrocks I, Tsarkov D (2006) Automated Benchmarking of Description Logic Reasoners. In Parsia B, Sattler U, Toman D (eds) Proc. of the Int. Workshop on Description Logics (DL'06), Windermere Lake District, UK. Volume 189 of CEUR., Lake District, UK 167–174

ISO 16642 (2003) Terminological markup framework in computer applications in terminology. Technical report, International Organization for Standardization (ISO). URL http://www.loria.fr/projets/TMF/

ISO 24613 (2006) Lexical markup framework in language resource management. Technical report, International Organization for Standardization (ISO). URL http://lirics.loria.fr/doc_pub/LMF%20rev9%2015March2006.pdf

Jarrar M (2005) Towards methodological principles for ontology engineering. PhD thesis, Vrije Universiteit Brussel, Brussels

Lozano-Tello A, Gómez-Pérez A (2004) ONTOMETRIC: a method to choose the appropriate ontology. J Database Manag 15(2)

Miles A, Matthews B, Beckett D, Brickley D, Wilson M, Rogers N (2005) SKOS: a language to describe simple knowledge structures for the web. In: Proceedings of the XTech conference 2005, Amsterdam

Montiel-Ponsoda E (2011) Multilingualism in ontologies: multilingual lexico-syntactic patterns for ontology modeling and linguistic information repository for ontology localization. PhD thesis, Universidad Politécnica de Madrid, Madrid

Montiel-Ponsoda E, Aguado de Cea G, Suárez-Figueroa MC, Palma R, Peters W, Gómez-Pérez A (2007) LexOMV: an OMV extension to capture multilinguality. In: 6th international semantic web conference. In Workshop Ontolex07, Busan

Montiel-Ponsoda E, Peters W, Aguado de Cea G, Espinoza M, Gómez-Pérez A, Sini M (2008) Multilingual and localization support for ontologies. Technical report, D2.4.2 NeOn project deliverable

Montiel-Ponsoda E, Aguado de Cea G, Gómez-Pérez A, Peters W (2010) Enriching ontologies with multilingual information. J Nat Lang Eng 17(3):283–309

NISO (2004) Understanding metadata. NISO Press, National Information Standards Organization. Available at http://www.niso.org/publications/press/UnderstandingMetadata.pdf

Palma R (2009) Ontology metadata management in distributed environments. PhD thesis, Universidad Politécnica de Madrid

Palma R, Haase P (2005) Oyster – sharing and re-using ontologies in a peer-to-peer community. In: International semantic web conference, Galway, pp 1059–1062

Palma R, Hartmann J, Haase P (2008) OMV – ontology metadata vocabulary for the semantic web. Technical report, Universidad Politécnica de Madrid, University of Karlsruhe. Version 2.4. Available at http://omv.ontoware.org/

Paslaru Bontas E, Mochol M, Tolksdorf R (2005) Case studies on ontology reuse. In: Proceedings of the IKNOW05 international conference on knowledge management, Graz

Peters W, Gangemi A, Villazón-Terrazas B (2010) Modelling and re-engineering linguistic/terminological resources. Technical report, D2.4.4 NeOn project deliverable

Pinto HS, Martins JP (2001) A methodology for ontology integration. In: Proceedings of the international conference on knowledge capture K-CAP01, Victoria

Presutti V, Gangemi A (2008) Content ontology design patterns as practical building blocks for web ontologies. In: ER '08: proceedings of the 27th international conference on conceptual modeling. Springer, Berlin/Heidelberg, pp 128–141

Russ T, Valente A, Macgregor R (1999) Practical experiences in trading off ontology usability and reusability. In: Proceedings of the 12th workshop on knowledge acquisition, modeling and management (EKAW'99), Banff, pp 16–21

Tappolet J, Kiefer C, Bernstein A (2010) Semantic web enabled software analysis. J Web Semant 8(2–3):225–240

Ungrangsi R, Simperl E (2008) OMEGA: an automatic ontology metadata generation algorithm. In: 16th international conference on knowledge engineering, knowledge management and knowledge patterns. Springer, Berlin/Heidelberg/New York

Uschold M, Healy M, Williamson K, Clark P, Woods S (1998) Ontology reuse and application. In: Proceedings of the international conference on formal ontology and information systems FOIS98, Trento

Vossen P (1998) Introduction to EuroWordNet. In Ide N, Greenstein D, Vossen P (eds) Special issue on EuroWordNet, vol 32(2–3), pp 73–89

Wang Y, Haase P, Palma R (2007) D1.4.1: Prototypes for managing networked ontologies. Technical report D1.4.1, University of Karlsruhe; NeOn deliverable. URL http://www.neon-project.org/

Part II
Ontology Engineering Activities

Chapter 5
Ontology Requirements Specification

Mari Carmen Suárez-Figueroa and Asunción Gómez-Pérez

Abstract The goal of the ontology requirements specification activity is to state why the ontology is being built, what its intended uses are, who the end users are, and which requirements the ontology should fulfill. This chapter presents detailed methodological guidelines for specifying ontology requirements efficiently. These guidelines will help ontology engineers to capture ontology requirements and produce the ontology requirements specification document (ORSD). The ORSD will play a key role during the ontology development process because it facilitates, among other activities, (1) the search and reuse of existing knowledge resources with the aim of reengineering them into ontologies, (2) the search and reuse of ontological resources (ontologies, ontology modules, ontology statements as well as ontology design patterns), and (3) the verification of the ontology along the ontology development.

5.1 Introduction

One of the key processes in software development is software specification (Sommerville 2007), whose aim is to understand and define what functionalities are required from the software product. It has been proved that a detailed software requirements document provides several benefits (IEEE 1993), such as (a) the establishment of the basis for agreement between customers and suppliers on what the software product is supposed to do, (b) the reduction of the development effort, (c) the provision of a basis for estimating costs and schedules, and (d) the offer of a baseline for validation and verification.

M.C. Suárez-Figueroa (✉) • A. Gómez-Pérez
Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid,
Campus de Montegancedo sn., 28660 Boadilla del Monte, Madrid, Spain
e-mail: mcsuarez@fi.upm.es; asun@fi.upm.es

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_5, © Springer-Verlag Berlin Heidelberg 2012

When a software application based on ontologies is being developed, ontology requirements should be identified in addition to the application requirements. Our experience in building ontology-based applications, in domains as diverse as satellite data processing[1], finding funding programs[2], fishery stocks[3], user context[4], and e-employment[5], has shown that more critical than capturing software requirements was the efficient and precise identification of the knowledge that the ontology should contain. Up to now, application developers already have precise methodologies (Sommerville 2007; IEEE 1993; Wiegers 2003) that help them to define application requirements. However, the guidelines included in current methodologies for building ontologies are not enough for defining ontology requirements.

For this reason, this chapter presents detailed methodological guidelines for specifying ontology requirements as part of the NeOn Methodology (see Chap. 2). Such methodological guidelines are based on the use of the so-called competency questions (CQs) (Grüninger and Fox 1995) and are inspired by how methodologies for building ontologies propose to perform the ontology requirements specification activity (Staab et al. 2001; Uschold 1996; Noy and McGuinness 2001). These guidelines are also inspired by available practices and previous experiences in different national and European funded projects. These methodological guidelines help to capture knowledge from users and to produce the ontology requirements specification document that will be used by ontology developers to develop an ontology that will fulfill the requirements identified.

5.2 Methodological Guidelines for Ontology Requirements Specification

The ontology requirements specification activity has a main goal to state why the ontology is being built, which its intended uses are, who the end users are, and what specific requirements the ontology should fulfill are. For specifying the specific ontology requirements, the competency questions technique proposed in (Grüninger and Fox 1995) is used. Before identifying the set of competency questions, the purpose and scope of the ontology should be identified, as well as its level of formality, and its intended uses and end users.

[1] http://www.ontogrid.net

[2] http://esperonto.net/fundfinder

[3] http://www.neon-project.org/nw/Ontology-driven_fish_stock_depletion_assessment_system

[4] http://www.isoco.com/ontologies/mio/index.html

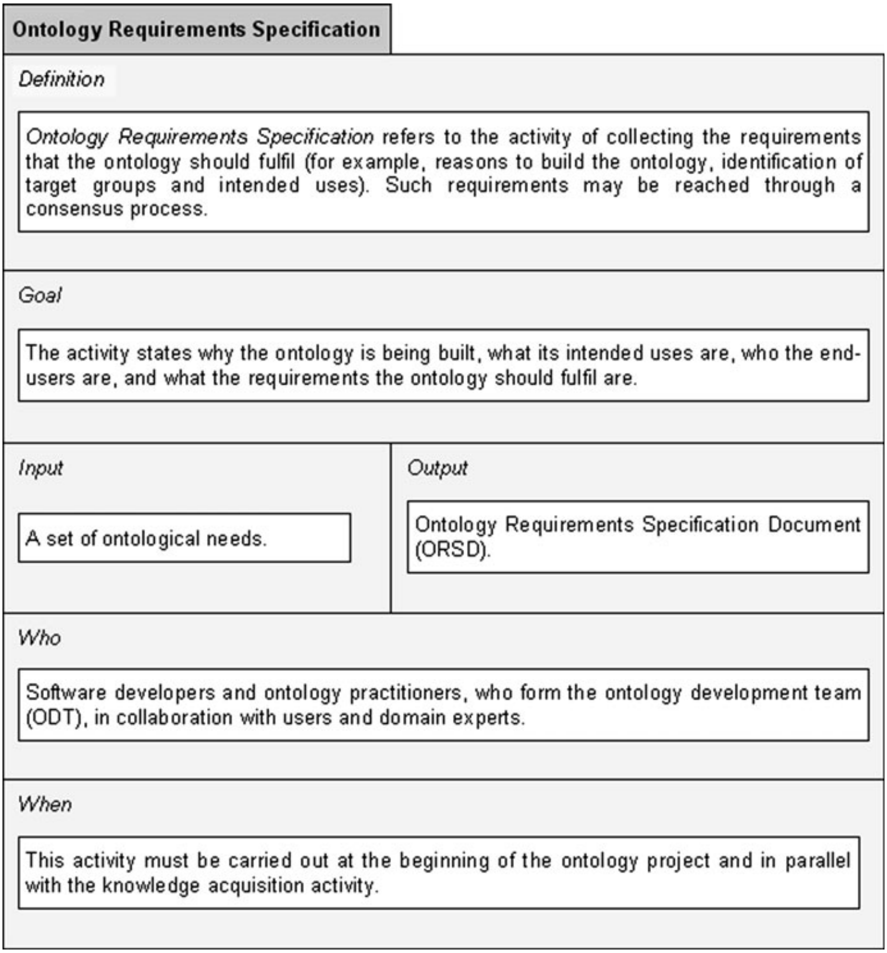
[5] http://www.seemp.org

The NeOn Methodology framework for building ontology networks proposes the *filling card*, for the ontology requirements specification activity, as it is shown in Fig. 5.1. The card includes the definition, goal, inputs, and outputs, who carries out the activity and when the activity should be performed.

The tasks for carrying out the ontology requirements specification activity can be seen in Fig. 5.2. The result of this activity is the ontology requirements specification document that should be written following the ORSD template shown in Table 5.1.

The tasks for carrying out the ontology requirements specification activity are explained in detail next.

Task 1. Identifying the purpose, scope, and implementation language. The objective is to determine the main goal of the ontology, its coverage and foreseeable



Ontology Requirements Specification

Definition

Ontology Requirements Specification refers to the activity of collecting the requirements that the ontology should fulfil (for example, reasons to build the ontology, identification of target groups and intended uses). Such requirements may be reached through a consensus process.

Goal

The activity states why the ontology is being built, what its intended uses are, who the end-users are, and what the requirements the ontology should fulfil are.

Input

A set of ontological needs.

Output

Ontology Requirements Specification Document (ORSD).

Who

Software developers and ontology practitioners, who form the ontology development team (ODT), in collaboration with users and domain experts.

When

This activity must be carried out at the beginning of the ontology project and in parallel with the knowledge acquisition activity.

Fig. 5.1 Ontology requirements specification *filling card*

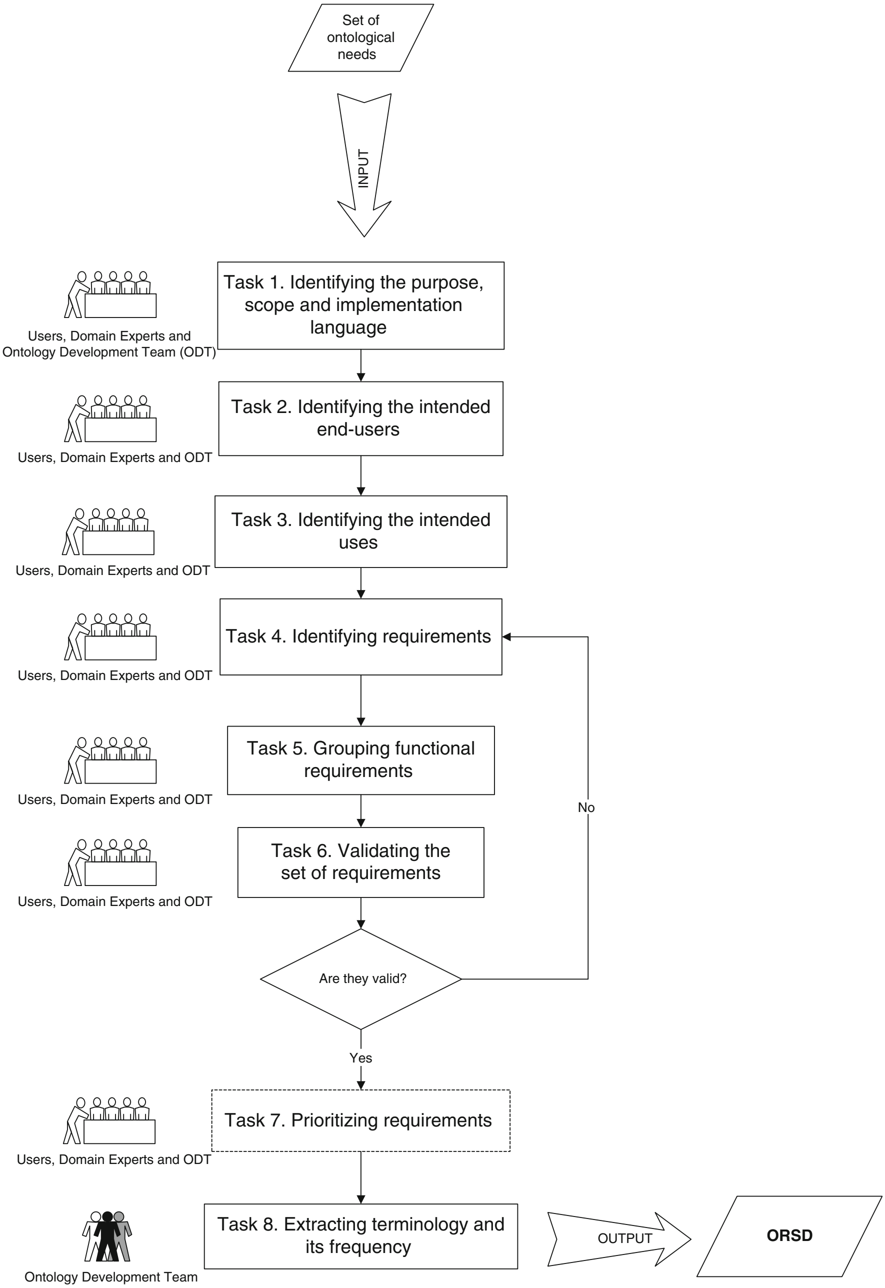

Set of ontological needs
INPUT
Task 1. Identifying the purpose, scope and implementation language
Users, Domain Experts and Ontology Development Team (ODT)
Task 2. Identifying the intended end-users
Users, Domain Experts and ODT
Task 3. Identifying the intended uses
Users, Domain Experts and ODT
Task 4. Identifying requirements
Users, Domain Experts and ODT
Task 5. Grouping functional requirements
Users, Domain Experts and ODT
Task 6. Validating the set of requirements
Users, Domain Experts and ODT
Are they valid?
No
Yes
Task 7. Prioritizing requirements
Users, Domain Experts and ODT
Task 8. Extracting terminology and its frequency
Ontology Development Team
OUTPUT
ORSD

Fig. 5.2 Tasks for ontology requirements specification

Table 5.1 Template for the OSRD

Ontology Requirements Specification Document Template

1 *Purpose*

The general goal of the ontology. In other words, the main function or role that the ontology should have

2 *Scope*

The general coverage and the degree of detail that the ontology should have

3 *Implementation language*

The formal language that the ontology should have

4 *Intended end users*

The intended end users expected for the ontology

5 *Intended uses*

The intended uses expected for the ontology

6 *Ontology requirements*

(a) *Non-Functional requirements*

The general requirements or aspects that the ontology should fulfill, including optionally priorities for each requirement

(b) *Functional requirements: Groups of competency questions*

The content specific requirements that the ontology should fulfill, in the form of groups of competency questions and their answers, including optionally priorities for each group and for each competency question

7 *Pre-glossary of terms*

(a) *Terms from competency questions*

The list of terms included in the competency questions and their frequencies

(b) *Terms from answers*

The list of terms included in the answers and their frequencies

(c) *Objects*

The list of objects included in the competency questions and in their answers

granularity, and its implementation language (e.g., OWL, RDFS[6], WSML[7]). The ontology development team holds a set of interviews with possible users and domain experts in order to carry out this task, taking as input a set of ontological needs, that is, the necessity of having the knowledge represented in the form of an ontology. The users and domain experts are crucial to identify the purpose and scope of the ontology; on the other hand, the ontology developers should decide the formal language to be used for implementing the ontology.

The task output is included in slots 1–3 of the template shown in Table 5.1.

Task 2. Identifying the intended end users. The goal of this task is to establish who the intended main end users of the ontology are. The ontology development team holds a set of interviews with the users and domain experts to carry out this task, taking as input a set of ontological needs.

[6] http://www.w3.org/TR/rdf-schema/

[7] http://www.wsmo.org/wsml/wsml-syntax

The task output is a list containing the intended end users of the ontology to be built; the list is included in slot 4 of the template shown in Table 5.1.

Task 3. Identifying the intended uses. The development of an ontology is mainly motivated by scenarios related to the application that will use the ontology. The goal of this task is to obtain the intended uses and use scenarios of the ontology. The ontology development team holds a set of interviews with the users and domain experts in order to carry out this task, taking as input a set of ontological needs; the purpose here is to obtain the uses of the ontology within the application, and to have a general idea of the application requirements, in terms of knowledge to be represented.

The task output is a list of intended uses in the form of scenarios, which is included in slot 5 of the template shown in Table 5.1. Such scenarios describe a set of general ontology requirements that the ontology should satisfy after being formally implemented. The scenarios should be described in natural language; they can be expressed in UML as use cases.

Task 4. Identifying requirements. The goal of this task is to acquire the set of requirements that the ontology should satisfy. Ontology requirements, similar to software requirements[8], can be divided into the following two types:

- *Non-functional ontology requirements* refer to the characteristics, qualities, or general aspects not related to the content that the ontology should represent. Examples of non-functional requirements are (a) whether the terminology to be used in the ontology must be taken from standards, (b) whether the ontology must be multilingual, or (c) whether the ontology should be written following a specific naming convention.
- *Functional ontology requirements*, which can be seen as content specific requirements, refer to the particular knowledge to be represented by the ontology and the particular terminology to be included in the ontology. In the SEEMP case study (Villazón-Terrazas et al. 2011), for example, the knowledge and the terminology are about curriculum vitae with candidate skills, education level, expertise, previous work experience, or about job offers with information on job location, salary, etc.

The ontology development team should interview the users and domain experts, taking as input a set of ontological needs, and they should obtain as result the initial set of ontology requirements (non-functional and functional) of the ontology to be built. To identify functional requirements, they use as main technique the writing of the requirements in natural language in the form of the so-called CQs. They can use

[8] In software engineering, functional requirements refer to the required behavior of the system, that is, the functionalities that the software system should have, while non-functional requirements refer to implicit expectations about how well the software system should work. That is, these requirements can be seen as aspects about the system or as “non-behavioral” requirements (Sommerville 2007).

mind map tools (Buzan 1974) and spreadsheet processors (such as MS Excel) for gathering the requirements. If people are geographically distributed, they can employ Wiki tools, such as Cicero[9].

Some strategies for identifying CQs are:

- Top-down: The team starts with complex questions that are decomposed in simpler ones.
- Bottom-up: The team starts with simple questions that are composed to create complex ones.
- Middle out: The team starts just writing down important questions that are composed and decomposed later on to form abstract and simple questions, respectively.

The output of this task is (1) a list of non-functional ontology requirements written in natural language, which is included in slot 6a of the template shown in Table 5.1, and (2) a list of functional ontology requirements in the form of CQs and their associated answers, which is the input of Task 5. This list of functional requirements will be grouped in Task 5 and then included in slot 6b of the template shown in Table 5.1.

Task 5. Grouping functional requirements. The goal of this task is to group into several categories the list of functional ontology requirements in the form of CQs and their associated answers obtained in Task 4. The users, the domain experts, and the ontology development team should classify the list of CQs written in natural language with a hybrid approach that not only combines preestablished categories such as time and date, units of measure, currencies, location, languages, etc., but also creates categories for those terms that appear with the highest frequencies in the list of CQs.

Techniques such as card sorting can be used when the grouping is done manually. In addition, mind map tools can help to display graphically and in groups the CQs; or Cicero if the grouping is done collaboratively.

The task output is the set of groups of functional requirements in the form of CQs and their associated answers, which is included in slot 6b of ORSD template shown in Table 5.1. The set of groups obtained in this task is used by the ontology development team to follow a modular approach during the ontology building.

Usually this task is carried out in parallel with Task 4.

To group CQs is useful because it permits to identify the essential parts to be covered by the ontology. The different groups of CQs will be used by the ontology development team for developing the ontology with a modularization approach. In addition, such groups can be used to organize the development in a collaborative fashion in which different teams are in charge of a set of CQs groups.

[9] http://cicero.uni-koblenz.de/wiki/index.php/Main_Page

Task 6. Validating the set of requirements. The aim here is to identify possible conflicts between ontology requirements, missing ontology requirements, and contradictions between them. Users, domain experts, and ontology developers must carry out this task taking as input the set of requirements identified in Task 4 (it includes both non-functional and functional requirements) to decide if each element of the set is valid or not.

The task output is the confirmation of the validity of the set of non-functional and functional ontology requirements.

The criteria that can be used in this validation task and that are mainly inspired by (IEEE 1993; Davis 1993) are the following:

- *Correctness*. A set of requirements is *correct* if, and only if, each requirement refers to some features of the ontology to be developed.
- *Completeness*. Inspired by (Wieringa 1996), a set of requirements can be considered *complete* if, and only if, users and domain experts review the requirements and confirm that they are not aware of additional requirements.
- *Consistency*. A set of requirements can be considered internally *consistent* if, and only if, no conflicts exist between them.
- *Verifiability*. A set of requirements is *verifiable* if, and only if, there is a finite process with a reasonable cost that tests whether the final ontology satisfies each requirement.
- *Understandability*. Each requirement must be *understandable* to end users and domain experts.
- *Unambiguity*. An ontology requirement is *unambiguous* if, and only if, it has only one meaning; that is, if it does not admit any doubt or misunderstanding.
- *Conciseness*. A set of requirements is *concise* if, and only if, each and every requirement is relevant and no duplicated or irrelevant requirements exist.
- *Realism*. A set of requirements is *realistic* if, and only if, each and every requirement meaning makes sense in the domain.
- *Modifiability*. A set of requirements is *modifiable* if, and only if, its structure and style allow changing issues in an easy, complete, and consistent way.
- *Traceability*. An ontology requirement is *traceable* if, and only if, its origin is known, and it can be referred to in other documents during the ontology development.

Task 7. Prioritizing requirements. The goal of this task is to give different levels of priority to the non-functional and functional ontology requirements identified. In the case of functional requirements, priorities should be given to the different groups of CQs and, within each group, to the different CQs; additionally, priorities could be given to each CQs independently of the groups. Users, domain experts, and the ontology development team should carry out this task, taking as input the requirements identified in Task 4 and the groups of CQs written in natural language obtained in Task 5. The task output is a set of priorities attached to each requirement, to each group of CQs, and to each CQ in a group. The output is included in the slots 6a and 6b of the template shown in Table 5.1.

Priorities will be used by the ontology development team for planning and scheduling the ontology development and for deciding which parts of the ontology are going to be developed first. This task is optional, but recommended. In fact, if no priorities are given to the groups of CQs, ontology developers will start modeling the ontology without any guidance regarding the functional requirements that should be implemented first; in this case, the waterfall ontology life cycle model should be selected during the scheduling of the ontology project. On the contrary, if different priorities have been assigned to functional ontology requirements, the iterative-incremental ontology life cycle model should be selected in the scheduling activity.

Task 8. Extracting terminology and its frequency. The goal of this task is to extract a pre-glossary of terms with their frequencies from the list of CQs and their answers identified in Task 4. The ontology development team carries out this task using terminology extraction techniques and tools supporting such techniques.

This pre-glossary of terms is divided in three different parts: terms from the CQs, terms from their answers, and terms identified as named entities.

- From the requirements in the form of CQs, the ontology development team should extract terminology (names, adjectives, and verbs) that will be formally represented in the ontology by means of concepts, attributes, relations, or instances (in the case of named entities).
- From the answers to the CQs, the ontology development team should extract terminology that could be represented in the ontology as concepts or as instances.
- From both CQs and corresponding answers, the ontology development team should extract named entities such as countries or currencies, which are objects in the universe of discourse.

The output is included in the slots 7a, 7b, and 7c of the template shown in Table 5.1, respectively.

The set of terms with higher appearance frequencies will be used later on for searching knowledge resources that could be potentially reused in the ontology development. The following heuristic can be applied: the set of more frequent terms is that requires more effort during the ontology development; for this reason, frequencies are important to know which knowledge resources allow to save more effort.

5.3 Ontology Requirements Specification in the Semantic Nomenclature Case Study

This section provides one example of how to use the guidelines proposed for the ontology requirements specification activity and what results are expected from any of the tasks detailed in the guidelines. The example shows an excerpt of the ORSD obtained after performing the ontology requirements specification activity following the methodological guidelines proposed in this chapter.

The example presented refers to the requirements specification of the ontology network developed within the Semantic Nomenclature case study (see Chap. 20). This requirements specification is not intended to be exhaustive; it just describes the most important points. A detailed and complete requirements specification is described in (Gómez-Pérez et al. 2007).

The main objectives of the Semantic Nomenclature case study were (a) helping in the systematization of creating, maintaining, and keeping up-to-date drug-related information, and (b) allowing an easy integration of new drug resources. In order to do that, the case study tackles the engineering of a pharmaceutical product ontology network implemented in OWL based on the nomenclature of products in the pharmaceutical sector in Spain. This ontology network represents the general aspects of the main terms and objects related to drugs, and it classifies these pharmaceutical terms according to the Anatomical Therapeutic Chemical (ATC)[10] classification.

Next we described the tasks followed for the ontology requirements specification activity within the Semantic Nomenclature case study based on the methodological guidelines proposed in this chapter.

Task 1. Identifying purpose, scope, and implementation language. The development of the Semantic Nomenclature ontology network is motivated by scenarios related to the end-user application that will use the ontology network. Such scenarios describe a set of the ontology requirements that the ontology should satisfy after being formally implemented. The motivating scenarios are described in (Gómez-Pérez et al. 2006). In summary, the purpose of building the ontology network within the Semantic Nomenclature case study is to provide a consensual knowledge model of the pharmaceutical domain and to solve the lack of communication between stakeholders in the pharmaceutical sector. The Semantic Nomenclature ontology network should provide a complete reference model about all the knowledge around the pharmaceutical products based on the main pharmaceutical classification and models used in the pharmaceutical sector. The implementation language selected is OWL.

Task 2. Identifying the intended end users. The analysis of the motivating scenarios described in (Gómez-Pérez et al. 2006) allowed ontology developers to identify the following intended end users of the ontology:

- User 1. Pharmacists who navigate across the ontology searching for drug information.
- User 2. GSCoP (General Spanish Council of Pharmacists) technicians who navigate across the ontology network and search for information or relations about a given concept (drug, active ingredient, etc.). GSCoP technicians also extract the latest information from different sources and update their database.

[10] http://www.whocc.no/atc_ddd_index/

- User 3. Spanish government analysts who study the situation of the pharmaceutical product information in the Spanish market or update the content.

Task 3. Identifying the intended uses. The analysis of the motivating scenarios described in (Gómez-Pérez et al. 2006) allowed ontology developers to identify the following main intended uses of the ontology:

- Use 1. To search for updated information about the characteristics of pharmaceutical products
- Use 2. To connect heterogeneous pharmaceutical models
- Use 3. To update pharmaceutical product information databases

Task 4. Identifying requirements. The *non-functional ontology requirement* identified was:

- NFR1. The ontology must support a multilingual scenario in the following languages: Spanish, Catalan, Basque, and Galician.

For specifying the *functional ontology requirements*, the competency question technique was used. In addition, the *bottom-up* approach for identifying them was used because it was the more direct way to work with the domain experts. Competency questions were stored in an Excel file and then rewritten in a mind map tool as appears in Fig. 5.3.

In total, 61 competency questions were identified; they are described in detail in (Gómez-Pérez et al. 2007). Examples of some competency questions are:

- CQ1. What is the drug commercial name? *Aspirina C (400/240MG 10 comprimidos efervescentes)*
- CQ2. What is the drug's main active ingredient (molecule)? *Acido acetilsalicilico*
- CQ3. What is its Spanish national code? *7127291*
- CQ4. What is the drug registration date? *01/09/1976*

Task 5. Grouping functional requirements. The 61 competency questions described in (Gómez-Pérez et al. 2007) were manually grouped into the following three groups with the domain experts' help:

- CQG1. Pharmaceutical product (29 competency questions)
- CQG2. Laboratory (4 competency questions)
- CQG3. Active ingredient (12 competency questions)

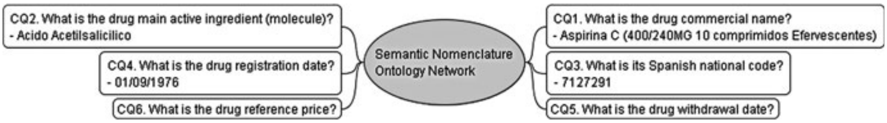

CQ2. What is the drug main active ingredient (molecule)?
- Acido Acetilsalicilico
CQ4. What is the drug registration date?
- 01/09/1976
CQ6. What is the drug reference price?
Semantic Nomenclature Ontology Network
CQ1. What is the drug commercial name?
- Aspirina C (400/240MG 10 comprimidos Efervescentes)
CQ3. What is its Spanish national code?
- 7127291
CQ5. What is the drug withdrawal date?

Fig. 5.3 Excerpt of competency questions

The criteria for grouping the competency questions were based on the identified uses, the identified users, and the domain experts' suggestions. Apart from the three aforementioned groups, ontology developers have created a new group, called Composite (CQG4), which includes the result of combining simple CQs to obtain more general and complex CQs.

Figure 5.4 shows the four groups with examples of CQs in the laboratory and pharmaceutical product groups.

Task 6. Validating the set of requirements. During the overall process, ontology developers received recommendations, suggestions, and advice from the domain experts, and they iterated several times until the final approval by the end users was achieved. Domain experts and ontology developers used the following criteria for validating the set of requirements (both non-functional and functional requirements):

- Correctness. Domain experts and ontology developers checked the correctness of each non-functional requirement and of each competency question, verifying that its formulation and answers were correct.
- Consistent. Domain experts also verified that the non-functional requirements and the competency questions did not have any possible inconsistency.

Task 7. Prioritizing requirements. Within the Semantic Nomenclature case study, ontology developers did not carry out this step. This means the first version of the ontology network must be able to represent the knowledge contained in all the competency questions and be able to cover all the non-functional requirements.

Task 8. Extracting terminology and its frequency. From the competency questions and their answers, ontology developers manually extracted the terminology that will be formally represented in the ontology network by means of concepts, attributes, and relations. In addition, ontology developers identified the terms and the objects in the universe of discourse. Examples of the terms identified are shown in Table 5.2.

After following these tasks, the output of the ontology requirements specification activity is the ontology requirements specification document. An excerpt of this document, which has been written for this chapter, is shown in Table 5.3.

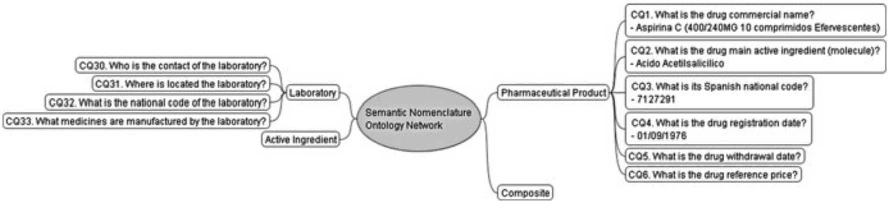

CQ30. Who is the contact of the laboratory?
CQ31. Where is located the laboratory?
CQ32. What is the national code of the laboratory?
CQ33. What medicines are manufactured by the laboratory?
Laboratory
Active Ingredient
Semantic Nomenclature Ontology Network
Pharmaceutical Product
Composite
CQ1. What is the drug commercial name?
- Aspirina C (400/240MG 10 comprimidos Efervescentes)
CQ2. What is the drug main active ingredient (molecule)?
- Acido Acetilsalicilico
CQ3. What is its Spanish national code?
- 7127291
CQ4. What is the drug registration date?
- 01/09/1976
CQ5. What is the drug withdrawal date?
CQ6. What is the drug reference price?

Fig. 5.4 Examples of competency questions in groups

Table 5.2 Examples of terminology

Terms from competency questions	
Drug	Dosage
Medicine	Date
Laboratory	Indication
Active ingredient	
Terms from answers	
Aspirina C	7127291
Ácido acetilsalicílico	01/01/1976
Objects	
Ibuprofeno	*Tetrazepam*
Butibufeno	*Procaina*
Penicilamina	*Ketamina*
Niflumico acid	*Clotiazapam*
Galamina	*Oxitriptan*

Table 5.3 Excerpt of Semantic Nomenclature ontology requirements specification document

Semantic Nomenclature ontology requirements specification

1 *Purpose*

The purpose of building the Semantic Nomenclature ontology network is to provide a reference model for the pharmaceutical domain. This model should be based on the main pharmaceutical classification and models used in the pharmaceutical sector

2 *Scope*

The ontology has to focus just on the Spanish and European pharmaceutical domain

3 *Implementation language*

The ontology has to be implemented in OWL

4 *Intended end users*

User 1. Pharmacist

User 2. GSCoP technicians

User 3. Spanish government analysts

5 *Intended uses*

Use 1. To search for updated information about the characteristics of pharmaceutical products

Use 2. To connect heterogeneous pharmaceutical models

Use 3. To update pharmaceutical product information databases

6 *Ontology requirements*

(a) *Non-Functional requirements*

NFR1. The ontology must support a multilingual scenario in the following languages: Spanish, Catalan, Basque, and Galician

(b) *Functional requirements: Groups of competency questions*

CQG1. Pharmaceutical product (29 CQs)

CQG2. Laboratory (4 CQs)

CQG3. Active ingredient (12 CQs)

CQG4. Composed ones (16 CQs)

7 *Pre-glossary of terms*

(a) *Terms from competency questions*

Drug (29) Medicine (15) Active ingredient (20) Laboratory (14)

(b) *Terms from answers*

Aspirina C *Ácido acetilsalicílico* 7127291 01/01/1976

(c) *Objects*

Ibuprofeno *Butibufeno* *Galamina* *Procaina*

5.4 Conclusions

One of the critical activities when developing ontologies is to identify their functional and non-functional requirements. In this chapter, the ontology requirements specification activity has been systematized by proposing detailed and prescriptive methodological guidelines for specifying ontology requirements, based on CQs, and by providing a template for writing the ontology requirement specification document (ORSD).

The ORSD will play a key role during the ontology development process because it facilitates different activities. In that sense, it will be shown in later chapters that the ontology requirements specification document (1) is a crucial input for the scheduling of ontology development projects (see Chap. 14) and (2) facilitates, among other activities, the search and reuse of non-ontological resources for reengineering them into ontologies (such as lexicons, glossaries, and dictionaries); the search and reuse of ontologies, ontology modules, ontology statements (e.g., using Watson), and ontology design patterns; and the verification of the ontology during the whole ontology development.

References

Buzan T (1974) Use your head. Ariel Books, British Broadcasting Corporation (BBC), London

Davis A (1993) Software requirements: objects, functions and states. Prentice Hall, Upper Saddle River

Gómez-Pérez JM, Daviaud C, Morera B, Benjamins R, Pariente Lobo T, Herrero Cárcel G, Tort G (2006) NeOn deliverable D8.1.1. Analysis of the pharma domain and requirements

Gómez-Pérez JM, Buil-Aranda C, Pariente Lobo T, Herrero Cárcel G, Baena A (2007) NeOn deliverable D8.3.1. Ontologies for the pharmaceutical case studies

Grüninger M, Fox MS (1995) Methodology for the design and evaluation of ontologies. In: Skuce D (ed) IJCAI95 workshop on basic ontological issues in knowledge sharing, Montreal, pp 6.1–6.10

IEEE (1993) IEEE Recommended practice for software requirements specifications. IEEE Std. 830

Noy NF, McGuinness DL (2001) Ontology development 101: a guide to creating your first ontology. Technical report KSL-01-05, Stanford Knowledge Systems Laboratory, Stanford

Sommerville I (2007) Software engineering, 8th edn. Addison-Wesley, London. ISBN 0-321-31379-8

Staab S, Schnurr HP, Studer R, Sure Y (2001) Knowledge processes and ontologies. IEEE Intell Syst 16(1):26–34

Uschold M (1996) Building ontologies: towards a unified methodology. In: Watson I (ed) 16th Annual conference of the British Computer Society Specialist Group on Expert Systems, Cambridge, UK

Villazón-Terrazas B, Ramírez J, Suárez-Figueroa MC, Gómez-Pérez A (2011) A network of ontology networks for building e-employment advanced systems. Expert Syst Appl 38(11): 13612–13624

Wiegers E (2003) Software requirements 2: practical techniques for gathering and managing requirements throughout the product development cycle, 2nd edn. Microsoft Press, Redmond. ISBN 0-7356-1879-8

Wieringa R (1996) Requirements engineering: frameworks for understanding. Wiley, New York

Chapter 6
Reusing and Re-engineering Non-ontological Resources for Building Ontologies

Boris Villazón-Terrazas and Asunción Gómez-Pérez

Abstract With the goal of speeding up the ontology development process, ontology developers are reusing as much as possible available ontological and non-ontological resources such as classification schemes, thesauri, lexicons, and folksonomies, that have already reached some consensus. The reuse of such non-ontological resources necessarily involves their re-engineering into ontologies. Based on this new trend, this chapter presents a general method for re-engineering non-ontological resources into ontologies, taking into account that non-ontological resources are highly heterogeneous in their data model and contents. The method is based on the so-called re-engineering patterns, which define a procedure that transforms the non-ontological resource components into ontology representational primitives. This chapter also presents the description of a software library that implements the transformations suggested by the patterns. Finally, the chapter depicts an evaluation of the method.

6.1 Introduction and Motivation

Research on ontology engineering methodologies has provided methods and techniques for developing ontologies from scratch. Well-recognized methodological approaches such as METHONTOLOGY (Gómez-Pérez et al. 2003), On-To-Knowledge (Schnurr et al. 2001), and DILIGENT (Pinto et al. 2004) issue guidelines that help researchers to develop ontologies. However, researchers face

B. Villazón-Terrazas (✉) • A. Gómez-Pérez
Ontology Engineering Group, Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo sn., 28660 Boadilla del Monte, Madrid, Spain
e-mail: bvillazon@fi.upm.es; asun@fi.upm.es

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_6, © Springer-Verlag Berlin Heidelberg 2012

an important limitation: no guidelines are provided for building ontologies by re-engineering some knowledge resources widely used within a particular community.

During the last decade, specific methods, techniques, and tools were proposed for building ontologies from available knowledge resources. First, ontology learning methods and tools were proposed to extract relevant concepts and relations from structured, semi-structured, and non-structured resources (Gómez-Pérez and Manzano-Macho 2004; Maedche and Staab 2001) in order to form a single ontology. One important constraint of these methods and tools is that they propose ad hoc solutions to transforming such resources, mainly texts, into ontologies. Hepp (2006), Hepp and de Brujin (2007), and Hepp (2007) stated that employing methods and techniques when transforming non-ontological resources to ontologies is key for the success of semantic technology for two main reasons: (1) if the use of semantic technologies for real-world data integration challenges is required, it is possible to refer to the original conceptual elements, and (2) for many domains, the existing category systems, XML schemas, and normative entity identifiers are the most efficient resources for engineering ontologies.

The literature presents a wide set of methods and tools for the ontologization of non-ontological resources. This ontologization of resources has led to the design of several specific methods, techniques, and tools (Hepp and de Brujin 2007; Hyvöonen et al. 2008; Gangemi et al. 2003; García and Celma 2005). These are mainly specific to a particular resource type, or to a particular resource implementation. Thus, every time ontology engineers are faced with a new resource type or implementation, they develop ad hoc solutions to transforming such resource into a single ontology.

The analysis of the ontologies developed by distinct research groups in different international and national projects have revealed that there are different alternative ways or possibilities to build ontologies by reusing and re-engineering the available knowledge resources used by a particular community. However, at this stage, we can state that all the projects perform an ad hoc transformation of the resources available for building ontologies.

Therefore, a new ontology development paradigm started approximately in 2007, whose emphasis was on the *reuse and possible subsequent reengineering of knowledge resources*, as opposed to custom-building new ontologies from scratch. However, in order to support and promote such reuse-based approach, new methods, techniques, and tools are needed.

The remainder of the chapter is organized as follows: Section 6.2 presents our categorization of non-ontological resources. Then, Sect. 6.3 describes the methodological guidelines for reusing non-ontological resources. Next, Sect. 6.4 provides the pattern-based method for re-engineering non-ontological resources into ontologies. Section 6.5 introduces the technological support for our re-engineering method. Then, Sect. 6.6 describes an example of the methodological guidelines presented here. Finally, Sect. 6.7 presents the conclusions and future work.

6.2 Types of Non-ontological Resources

The knowledge resources, reused in several projects for building ontologies, contain readily available a wealth of category definitions and reflect some degree of community consensus. In this chapter, we refer to *non-ontological resources (NOR)*[1]. Examples of NORs are classification schemes, thesauri, lexica, and folksonomies, among others. This type of resources encodes different types of knowledge and can be implemented in different ways.

Our analysis of the literature has revealed different ways of categorizing non-ontological resources. Thus, Maedche and Staab (2001) and Sabou et al. (2007) classify non-ontological resources into unstructured (e.g., free text), semi-structured (e.g., folksonomies), and structured (e.g., databases) resources, whereas Gangemi et al. (1998) distinguish catalogs of normalized terms, glossed catalogues, and taxonomies. Finally, Hodge (2000) proposes characteristics such as structure, complexity, relationships among terms, and historical functions for classifying them. However, an accepted and agreed-upon typology of non-ontological resources does not exist yet.

Therefore, one of the contributions of this chapter is the categorization of NORs, according to the following three features presented in Fig. 6.1: (1) type of NOR, which refers to the type of inner organization of the information; (2) data model, that is, the design data model used to represent the knowledge encoded by the resource; and (3) resource implementation.

According to the *type of NORs*, we classify them into:

- *Glossaries*: A glossary is an alphabetical list of terms or words found in or related to a specific topic or text. It may or may not include explanations, and its vocabulary may be monolingual, bilingual, or multilingual (Wright and Budin 1997). An example of glossary is the FAO Fisheries Glossary[2].
- *Lexicons*: In a restricted sense, a computational lexicon is considered as a list of words or lexemes hierarchically organized and normally accompanied by meaning and linguistic behavior information (Hirst 2004). A fine example is WordNet[3], the best known computational lexicon of English.
- *Classification schemes*: A classification scheme is the descriptive information of an arrangement or division of objects into groups according to the characteristics that the objects have in common (ISO/IEC FDIS 11179-1). A good example is the Fishery International Standard Statistical Classification of Aquatic Animals and Plants (ISSCAAP)[4].
- *Thesauri*: Thesauri are controlled vocabularies of terms in a particular domain with hierarchical, associative, and equivalence relations between terms.

[1] Along this chapter, we use either *NOR* or *non-ontological resource* without distinction

[2] http://www.fao.org/fi/glossary/default.asp

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

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

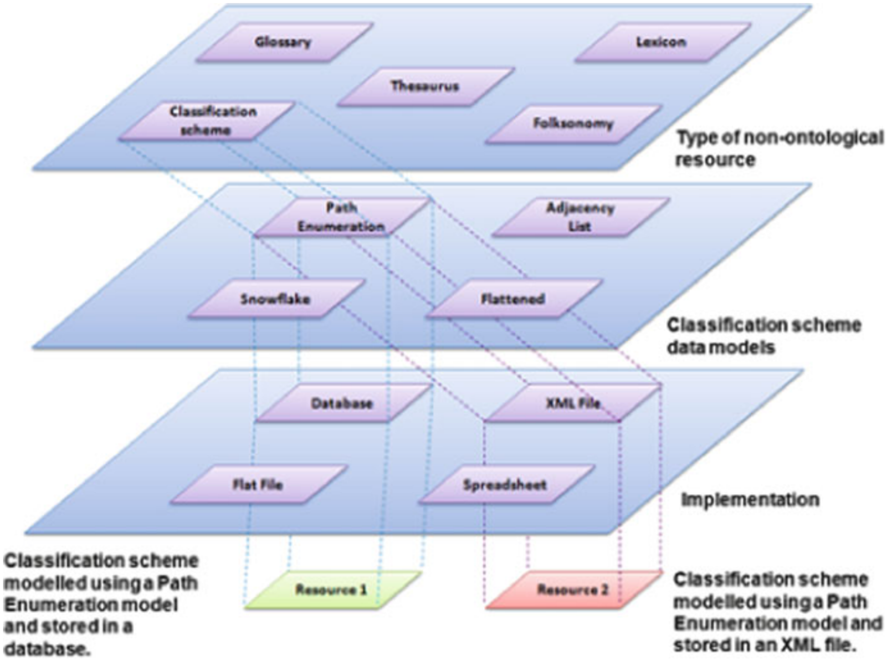

Glossary
Lexicon
Thesaurus
Classification scheme
Folksonomy
Type of non-ontological resource
Path Enumeration
Adjacency List
Snowflake
Flattened
Classification scheme data models
Database
XML File
Flat File
Spreadsheet
Implementation
Classification scheme modelled using a Path Enumeration model and stored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

Fig. 6.1 Non-ontological resource categorization

Thesauri are mainly used for indexing and retrieving articles in large databases (ISO 2788). An example of thesaurus is the AGROVOC[5] thesaurus.

- *Folksonomies*: Folksonomies are Web 2.0 systems that users employ to upload and annotate their content effortlessly and without requiring any expert knowledge[6]. This simplicity has made folksonomies widely successful, and this success, in its turn, has resulted in a massive amount of user-generated and user-annotated web content. The main advantage of folksonomies is the implicit knowledge they contain. When users tag resources with one or more tags, they assign these resources the meaning of the tag. Furthermore, the co-occurrence of tags implies a semantic correlation among them. An example of how folksonomies are used can be seen in the *del.icio.us*[7] web site.

The knowledge encoded by the resource can be represented in different ways, known as data models. A data model (Carkenord 2002) is an abstract model that describes how data is represented and accessed. There are three types: (1) the conceptual data model, which presents the primary entities and relationships of

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

[6] http://www.vanderwal.net/folksonomy.html

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

concern to a specific domain; (2) the logical data model, which depicts the logical entity types, the data attributes describing those entities, and the relationships between entities; and (3) the physical data model, which is related to a specific implementation of the resource. In this chapter, we will use the term data model when referring to the logical data model. With regard to the *data model*, there are different ways of representing the knowledge encoded by the resource. In this chapter, we only focus in data models for classification schemes, thesauri, and lexica. The data models are described in detail in Villazón-Terrazas et al. (2010).

Next we present several *data models for classification schemes*, shown in Fig. 6.2.

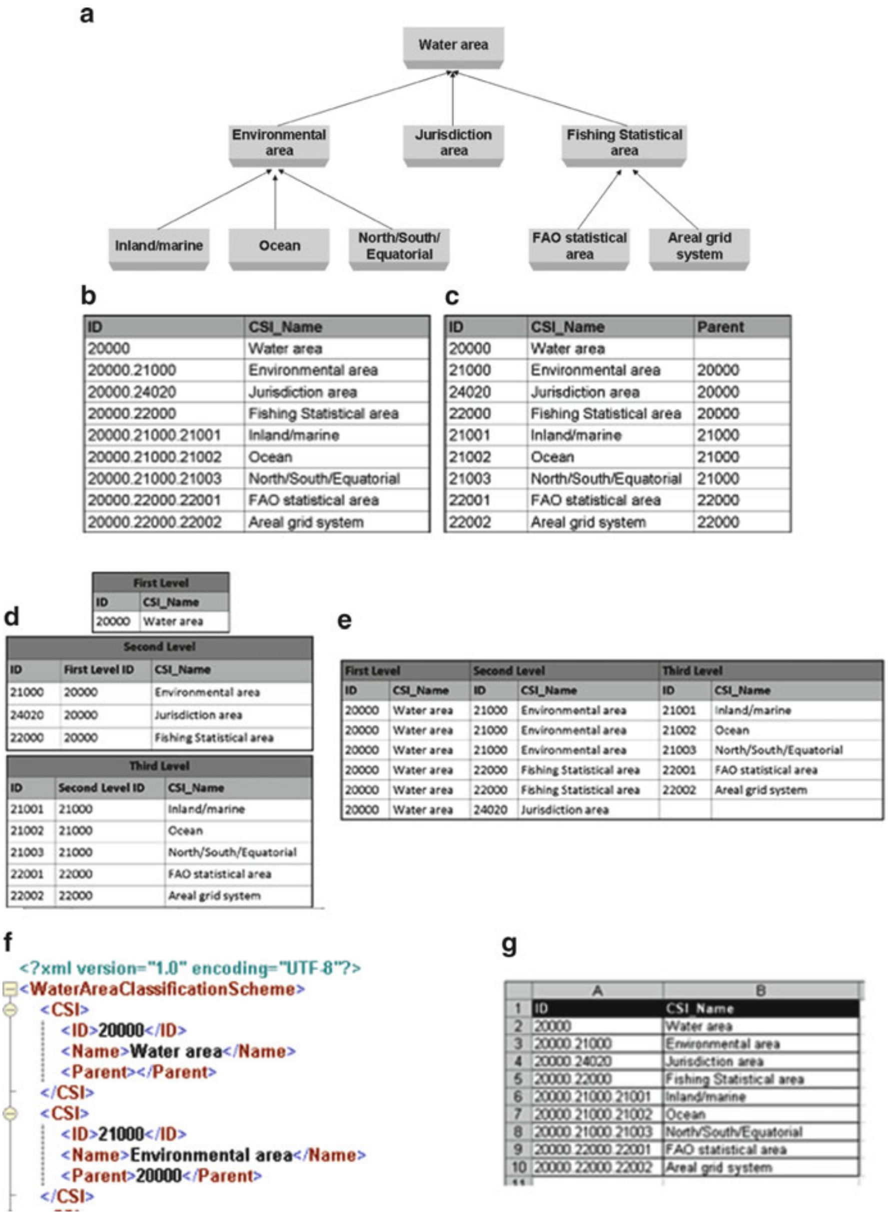
- *Path enumeration* (Brandon 2005): A path enumeration model (see Fig. 6.2b) is a recursive structure for hierarchy representations and is defined as a model that stores, for each node, the path (as a string) from the root to the node. This string is the concatenation of the node code in the path from the root to the node.
- *Adjacency list* (Brandon 2005): An adjacency list model is a recursive structure for hierarchy representations comprising a list of nodes with a linking column to their parent nodes. Figure 6.2c shows this model.
- *Snowflake* (Malinowski and Zimányi 2006): A snowflake model is a normalized structure for hierarchy representations. For each hierarchy level, a table is created. In this model, each hierarchy node has a column linked to its parent node. Figure 6.2d shows this model.
- *Flattened* (Malinowski and Zimányi 2006): A flattened model is a denormalized structure for hierarchy representations. The hierarchy is represented by a table where each hierarchy level is stored in a different column. Figure 6.2e shows this model.

Next, we present two *data models for thesauri*.

- *Record-based model* (Soergel 1995): A record-based model is a denormalized structure that for every term uses a record with information about the term, such as synonyms, broader, narrower, and related terms. This model looks like the flattened model for classification scheme.
- *Relation-based model* (Soergel 1995): A relation-based model leads to a more elegant and efficient structure. Information is stored in individual pieces that can be arranged in different ways. Relationship types are not defined as fields in a record, they are simply data values in a relationship record, thus new relationship types can be introduced with ease. There are three entities: (1) a term entity, which contains the overall set of terms; (2) a term-term relationship entity, in which each record contains two different term codes and the relationship between them; and (3) a relationship source entity, which contains the overall resource relationships.

Next we present a *data model for lexica*.

- *Record-based model* (Soergel 1995): This model can also be used for lexicons because the use of a record for every lexical resource and information about that lexical resource is possible.



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

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

First Level	
ID	CSI_Name
20000	Water area

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<WaterAreaClassificationScheme>
  <CSI>
    <ID>20000</ID>
    <Name>Water area</Name>
    <Parent></Parent>
  </CSI>
  <CSI>
    <ID>21000</ID>
    <Name>Environmental area</Name>
    <Parent>20000</Parent>
  </CSI>
```

	A	B
1	ID	CSI_Name
2	20000	Water area
3	20000 21000	Environmental area
4	20000 24020	Jurisdiction area
5	20000 22000	Fishing Statistical area
6	20000 21000 21001	Inland/marine
7	20000 21000 21002	Ocean
8	20000 21000 21003	North/South/Equatorial
9	20000 22000 22001	FAO statistical area
10	20000 22000 22002	Areal grid system
11		

Fig. 6.2 Example of classification scheme. (**a**) Excerpt of the Water Area classification scheme, (**b**) Path Enumeration data model, (**c**) Adjacency List data model, (**d**) Snowflake data model, (**e**) Flattened data model, (**f**) XML implementation for the Adjacency List data model, (**g**) Spreadsheet implementation for the Path Enumeration data model

- *Relation-based model* (Soergel 1995): It can also be used for lexicons because the storage of information about the lexicon in individual pieces is possible.

According to the *implementation*, we classify NORs into:

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

In summary, Fig. 6.1 shows how a given type of NOR can be modeled following one or more data models, each of which implemented in different ways at the implementation layer. Figure 6.1 shows, as an example, a classification scheme modeled following a path enumeration model. In this case, the classification scheme is implemented in a database and in an XML file.

To exemplify the non-ontological categorization presented with a real life classification scheme, we use an excerpt from the FAO water area classification presented in (Fig. 6.2a). This classification schema is modeled following a path enumeration model (Fig. 6.2b), an adjacency list model (Fig 6.2c), a snowflake model (Fig. 6.2d), and a flattened model (Fig. 6.2e). Figure 6.2f presents an XML implementation of the adjacency list model, and Fig. 6.2g presents a spreadsheet implementation of the path enumeration model of the same classification scheme.

It is worth mentioning that this first categorization of NORs is neither exhaustive nor complete. Currently, we are enriching it by adding examples taken from RosettaNet[8] and Electronic Data Interchange, EDI[9].

Moreover, we can map available non-ontological resources to our categorization. Next we present a brief list of them.

- The United Nations Standard Products and Services Code, UNSPSC[10], is a classification scheme, modeled with the path enumeration data model and stored in a relational database.
- WordNet[11], a lexical database for English, is a lexicon, modeled with the relation-based data model and stored in several implementations; a particular implementation of it is a relational database.

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

[9] http://www.edibasics.co.uk/

[10] http://www.unspsc.org/

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

- UMLS12[12] is a very large, multipurpose, multilingual thesaurus that contains information about biomedical and health-related concepts. It is modeled with the record-based model and stored in a flat file.
- MeSh[13], the Medical Subject Headings, is a classification scheme, modeled with the path enumeration data model.
- The Art and Architecture Thesaurus[14] is modeled with the record-based data model and implemented in XML.
- The ISCO-08 International Standard Classification of Occupations[15] is a classification scheme modeled with the path enumeration data model and implemented in a database and spreadsheet.
- The European Training Thesaurus, ETT[16], is modeled with the record-based data model and implemented in XML.
- The Classification of Fields of Education and Training, FOET[17], is a classification scheme modeled with path enumeration data model and implemented in XML and spreadsheet.
- The Aquatic Sciences and Fisheries Abstracts thesaurus, ASFA[18], is modeled with the record-based data model and implemented in XML.
- The AGROVOC thesaurus[19] is modeled with the relation-based data model and implemented in a database.
- The Fisheries Global Information System, FIGIS[20], is modeled with the adjacency list data model and implemented in a database.
- The Classification of Italian Education Titles published by the National Institute of Statistics, ISTAT[21], is a classification scheme modeled with the flattened data model and implemented in a spreadsheet.

[12] http://www.nlm.nih.gov/pubs/factsheets/umlsmeta.html

[13] http://www.nlm.nih.gov/mesh/

[14] http://www.getty.edu/research/tools/vocabularies/aat/index.html

[15] http://www.ilo.org/public/english/bureau/stat/isco/index.htm

[16] http://libserver.cedefop.europa.eu/ett/en/

[17] http://ec.europa.eu/eurostat/ramon/nomenclatures/index.cfm?TargetUrl=DSP_GEN_DESC_VIEW_NOHDR&StrNom=EDU_TRAINI&StrLanguageCode=EN

[18] http://www.fao.org/fishery/asfa/8/en

[19] http://aims.fao.org/website/AGROVOC-Thesaurus/sub

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

[21] http://en.istat.it/

6.3 Methodological Guidelines for Reusing Non-ontological Resources

Once we have defined and categorized the non-ontological resources to be dealt with, we present the methodological guidelines for reusing them. The goal of the non-ontological resource reuse process is to choose the most suitable non-ontological resource for building ontologies. Domain experts, software developers, and ontology practitioners carry out this process by taking as input the ontology requirements specification document (ORSD)[22] 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 extent, covers the expected domain. Figure 6.3 shows the filling card used in the process of reusing non-ontological resources, which includes the definition, goal, input, output, performer of the process, and period of execution.

This process includes the activities and tasks presented in Fig. 6.4 and is explained next.

6.3.1 Activity 1. Search Non-ontological Resources

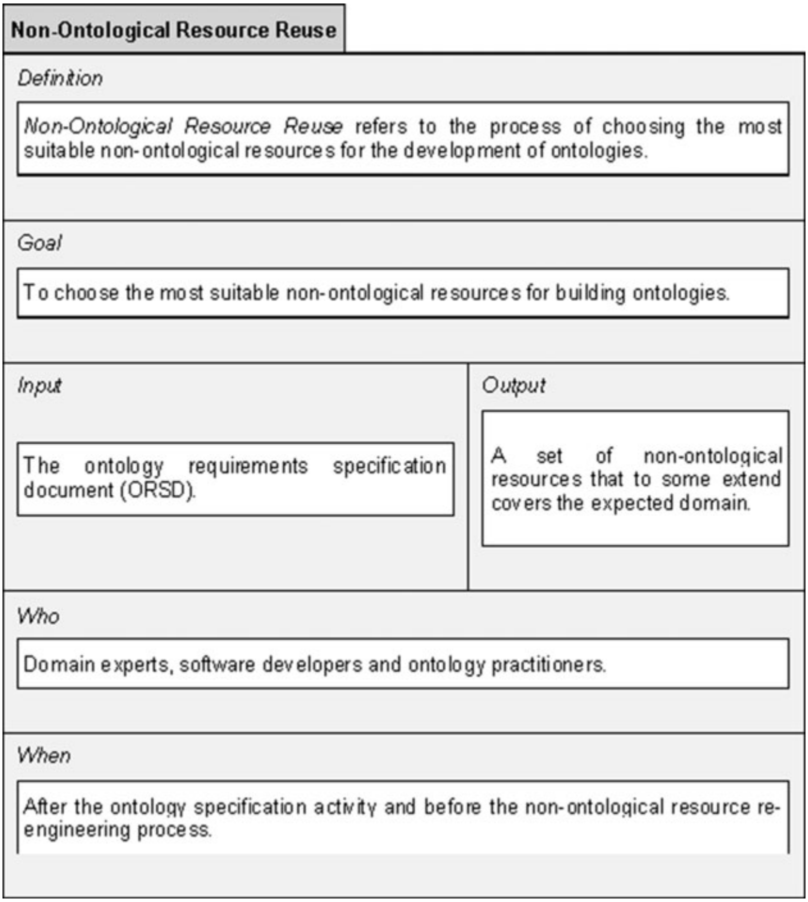
The goal of the activity is to search non-ontological resources from highly reliable web sites, domain-related sites, and resources within organizations. Domain experts, software developers, and ontology practitioners carry out this activity, taking as input the ORSD. They use the terms that have the highest frequency in the ORSD to search for the candidate non-ontological resources that cover the desired terminology. The activity output is a set of candidate non-ontological resources that may belong to any of the identified typologies described in Sect. 6.2.

6.3.2 Activity 2. Assess the Set of Candidate Non-ontological Resources

The goal of the 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 consider the following measurable criteria: (1) coverage, (2) precision plus two subjective criteria, (3) quality[23], and (4) consensus. These criteria are inspired on the work proposed in Gangemi et al. (2006).

[22] This document is the outcome of the ontology specification activity (Suárez-Figueroa et al. 2009) (see Chapter 5).

[23] A deep analysis of the quality of the resource is out of the scope of this chapter.



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

The ontology requirements specification document (ORSD).

Output

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.

Fig. 6.3 Non-ontological resource reuse filling card

6.3.2.1 Task 2.1 Extract Lexical Entries

The goal of this task is to extract the lexical entries of the non-ontological resources. The task is carried out by software developers and ontology practitioners by taking as input the non-ontological resources and extracting their lexical entries with terminology extraction tools.

6.3.2.2 Task 2.2 Calculate Precision

The goal of this task is to calculate the precision of the candidate non-ontological resources. Precision is a measure widely used in information retrieval (Baeza-Yates and Ribeiro-Neto 1999) and is defined as the proportion of retrieved material that is

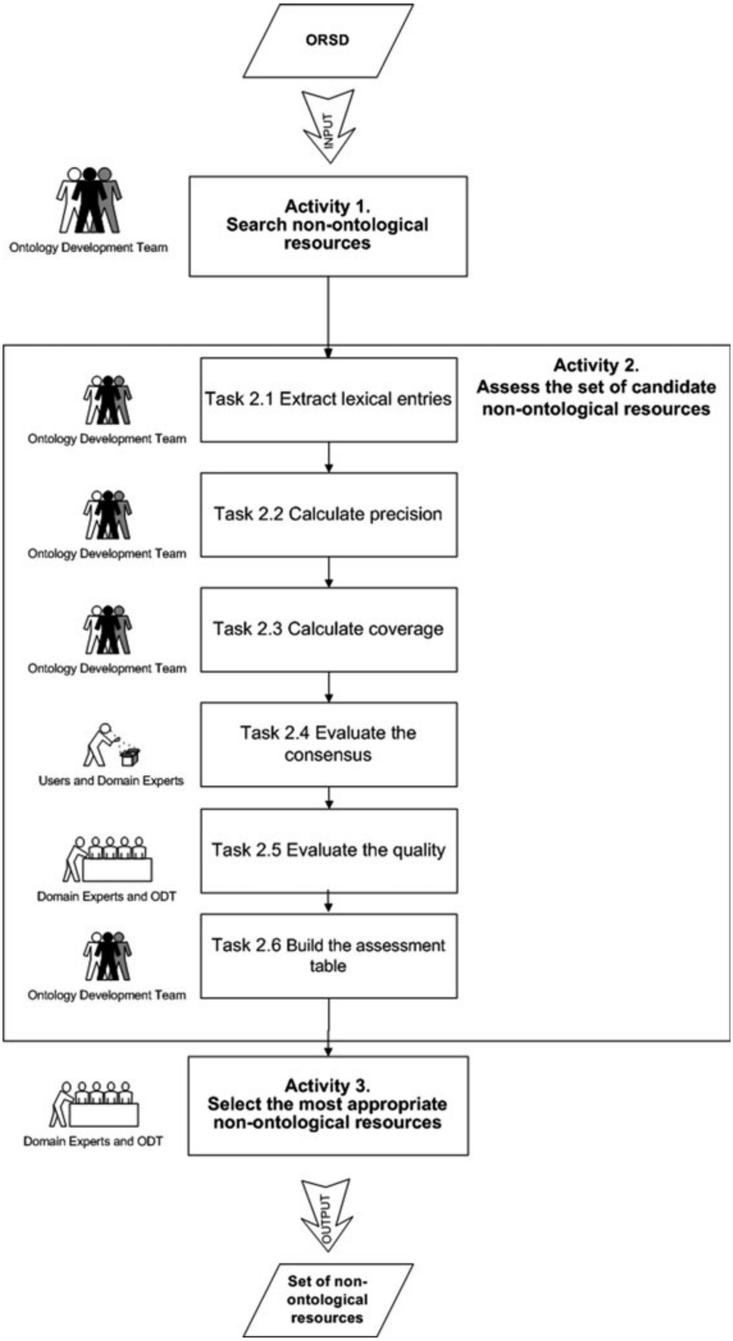

ORSD
INPUT
Ontology Development Team
Activity 1.
Search non-ontological resources
Activity 2.
Assess the set of candidate non-ontological resources
Ontology Development Team
Task 2.1 Extract lexical entries
Ontology Development Team
Task 2.2 Calculate precision
Ontology Development Team
Task 2.3 Calculate coverage
Users and Domain Experts
Task 2.4 Evaluate the consensus
Domain Experts and ODT
Task 2.5 Evaluate the quality
Ontology Development Team
Task 2.6 Build the assessment table
Domain Experts and ODT
Activity 3.
Select the most appropriate non-ontological resources
OUTPUT
Set of non-ontological resources

Fig. 6.4 Activities for the non-ontological resource reuse process

actually relevant. This task is carried out by software developers and ontology practitioners by taking as input the lexical entries extracted for the non-ontological resources and the terminology gathered in the ORSD. To adapt this precision measure into our context, we need to define:

- *NORLexicalEntries* as the set of lexical entries extracted from the non-ontological resource
- *ORSDTerminology* as the set of identified terms included in the ORSD

Now we can define the precision, in our context, as the proportion of the lexical entries of the non-ontological resource that are included in the identified terms of the ORSD over the lexical entries of the non-ontological resource. This is expressed as follows:

$$Precision = \frac{|\{NORLexicalEntries\} \cap \{ORSDTerminology\}|}{|\{NORLexicalEntries\}|}$$

6.3.2.3 Task 2.3 Calculate Coverage

The goal of this task is to calculate the coverage of the non-ontological resources. Coverage is based on the recall measure used in information retrieval (Baeza-Yates and Ribeiro-Neto 1999). Recall is defined as the proportion of relevant material actually retrieved in answer to a search request. This task is carried out by software developers and ontology practitioners by taking as input both the lexical entries extracted from the non-ontological resources and the terminology gathered in the ORSD. To adapt this measure into our context, we use the aforementioned definitions of *NORLexicalEntries* and *ORSDTerminology*. In this context, coverage is the proportion of the identified terms of the ORSD that are included in the lexical entries of the non-ontological resource over the identified terms of the ORSD. This is expressed as follows:

$$Coverage = \frac{|\{NORLexicalEntries\} \cap \{ORSDTerminology\}|}{|\{ORSDTerminology\}|}$$

6.3.2.4 Task 2.4 Evaluate the Consensus

The goal of this task is to evaluate the consensus of the non-ontological resources. Consensus is a subjective and not quantifiable criterion. This task is carried out by domain experts, taking as input the non-ontological resources for stating whether the non-ontological resources contain terminology agreed upon by the community or not. We propose a preliminary starting point for this evaluation. Domain experts have to check whether the resource is coming from:

- A standardization body or any entity whose primary activity is to develop, coordinate, promulgate, revise, amend, reissue, or otherwise maintain standards; for example, the International Organization for Standardization (ISO), the American National Standards Institute (ANSI), and the World Wide Web Consortium (W3C)
- Large organizations across national governments, such as the Food and Agriculture Organization of the United Nations (FAO), the World Health Organization (WHO), the United Nations Educational, Scientific and Cultural Organization (UNESCO), and the International Olympic Committee (IOC)
- A large enough user community to make it profitable for developers to use it as a means of general interoperability

Either the resource is coming from any of the aforementioned parties or not, domain experts may state that the resource has reached some degree of consensus.

6.3.2.5 Task 2.5 Evaluate the Quality

The goal of this task is to evaluate the quality of the resource. We do not intend to provide a deep analysis of the quality of the resource but to offer some preliminary considerations about it. In this chapter, we propose to check the following quality attributes:

- Good documentation of the resource.
- Lack of anomalies of the non-ontological resource, such as redundancies or inconsistencies.
- Reliability of the non-ontological resource. This means analyzing whether we can trust the resource or not.

6.3.2.6 Task 2.6 Build the Assessment Table

The goal of this task is to create an assessment table of the non-ontological resources. Software developers and ontology practitioners carry out this task, taking as input the non-ontological resources with their respective values for precision, coverage, consensus, and quality criteria, for the construction of the assessment table. This table is shown in Table 6.1. The first column shows the non-ontological resources found. The

Table 6.1 Assessment table for the NORs

NOR	Precision	Coverage	Consensus	Quality
NOR 1	NOR 1 precision value	NOR 1 coverage value	(Yes/no)	(Yes/no)
NOR 2	NOR 2 precision value	NOR 2 coverage value	(Yes/no)	(Yes/no)
NOR 3	NOR 3 precision value	NOR 3 coverage value	(Yes/no)	(Yes/no)

precision column shows the precision value calculated for each non-ontological resource. Then, the coverage column shows the coverage value calculated for each non-ontological resource. Next, the consensus column depicts the domain experts' judgment about whether the non-ontological resource has been agreed on by the community or not (Yes/No). Finally, the quality column illustrates the domain experts, software developers, and ontology practitioners' judgment about whether the resource has an acceptable level of quality or not (Yes/No).

6.3.3 Activity 3. Select the Most Appropriate Non-ontological Resources

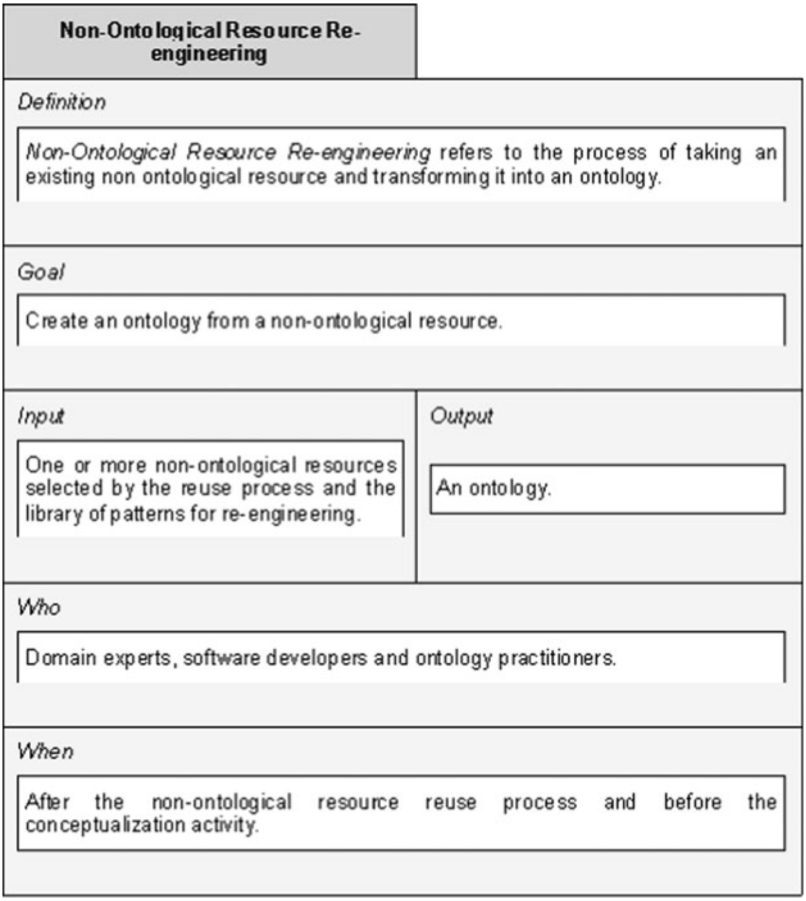
The goal of this activity is to select the most appropriate non-ontological resources to be transformed into an ontology. This activity is carried out by domain experts, software developers, and ontology practitioners, taking as input the non-ontological resource assessment table. The selection is performed manually and we recommend looking for resources with:

- Consensus. This criterion is taken into account in the first place because if the resource to be reused contains terminology agreed upon by the community, the effort and time spent in finding out the right labels for the ontology terms will decrease considerably.
- Quality. This criterion is taken into account in the second place because if the resource to be reused has an acceptable level of quality, then the resultant ontology should also have it.
- High value of coverage. This criterion is taken into account in the third place because our third concern is to consider most of the ORSD terms identified.
- High value of precision. This criterion is taken into account in the fourth place because our fourth concern is the proportion of non-ontological lexical entries over the identified terms of the ORSD.

The activity output is a ranked list of non-ontological resources that, to some extent, covers the expected domain. These resources will be ready for the re-engineering process.

6.4 Methodological Guidelines for Re-engineering NORs into Ontologies

In this section, we depict the prescriptive methodological guidelines for re-engineering NORs. The goal of the method for re-engineering non-ontological resources is to transform a non-ontological resource into an ontology. The output of the process is an ontology. Figure 6.5 shows the filling card of the non-ontological



Non-Ontological Resource Re-engineering

Definition

Non-Ontological Resource Re-engineering refers to the process of taking an existing non ontological resource and transforming it into an ontology.

Goal

Create an ontology from a non-ontological resource.

Input

One or more non-ontological resources selected by the reuse process and the library of patterns for re-engineering.

Output

An ontology.

Who

Domain experts, software developers and ontology practitioners.

When

After the non-ontological resource reuse process and before the conceptualization activity.

Fig. 6.5 Non-ontological resource re-engineering filling card

resource re-engineering process, which includes the definition, goal, input, output, performer of the process, and time execution.

The NOR re-engineering process consists of the three activities depicted in Fig. 6.6.

6.4.1 *Activity 1. Non-ontological Resource Reverse Engineering*

The goal of this activity is to analyze a non-ontological resource, to identify its underlying terms, and to create representations of the resource at the different levels of abstraction (design, requirements, and conceptual).

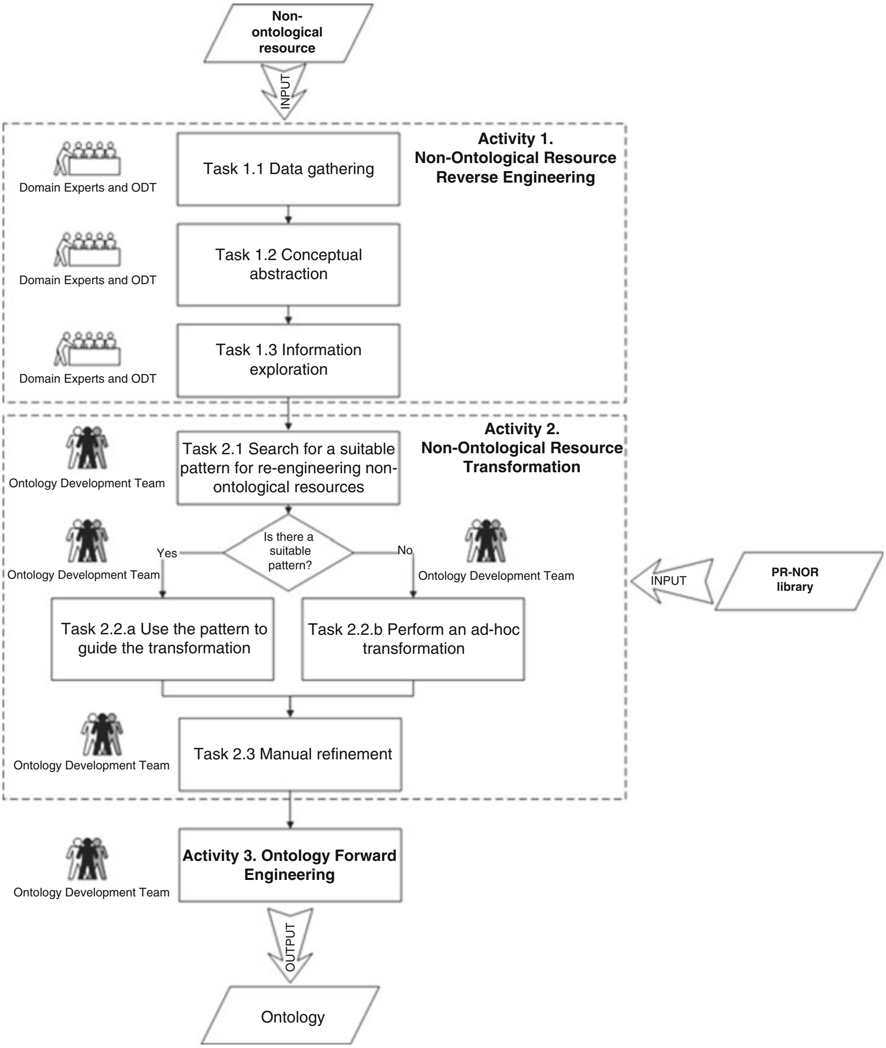

Non-ontological resource
INPUT
Activity 1. Non-Ontological Resource Reverse Engineering
Domain Experts and ODT
Task 1.1 Data gathering
Domain Experts and ODT
Task 1.2 Conceptual abstraction
Domain Experts and ODT
Task 1.3 Information exploration
Activity 2. Non-Ontological Resource Transformation
Ontology Development Team
Task 2.1 Search for a suitable pattern for re-engineering non-ontological resources
Is there a suitable pattern?
Yes
No
Ontology Development Team
Ontology Development Team
INPUT
PR-NOR library
Task 2.2.a Use the pattern to guide the transformation
Task 2.2.b Perform an ad-hoc transformation
Ontology Development Team
Task 2.3 Manual refinement
Ontology Development Team
Activity 3. Ontology Forward Engineering
OUTPUT
Ontology

Fig. 6.6 Re-engineering process for non-ontological resources

6.4.1.1 Task 1.1 Data Gathering

The goal of this task is to search and compile all the available data and documentation about the non-ontological resource, including purpose, components, data model, and implementation details.

6.4.1.2 Task 1.2 Conceptual Abstraction

The goal of this task is to identify the schema of the non-ontological resource including the conceptual components and their relationships. If the conceptual schema is not available in the documentation, the schema should be reconstructed manually or with a data modeling tool.

6.4.1.3 Task 1.3 Information Exploration

The goal of this task is to find out how the conceptual schema of the non-ontological resource and its content are represented in the data model. If the non-ontological resource data model is not available in the documentation, the data model should be reconstructed manually or with a data modeling tool.

6.4.2 Activity 2. Non-ontological Resource Transformation

This activity has as a goal to generate a conceptual model from the non-ontological resource. We propose the use of patterns for re-engineering non-ontological resources (PR-NOR) to guide the transformation process.

6.4.2.1 Task 2.1 Search for a Suitable Pattern for Re-engineering Non-ontological Resource

The goal of this task is to find out if there is any applicable re-engineering pattern that transforms the non-ontological resource into a conceptual model. The search is performed in the ODP Portal24[24], which includes the PR-NOR library, and with the following criteria: (1) non-ontological resource type, (2) internal data model of the resource, and (3) the transformation approach.

6.4.2.2 Task 2.2.a Use Re-engineering Patterns to Guide the Transformation

The goal of this task is to apply the re-engineering pattern obtained in Task 2.1 (see Sect. 6.4.2.1) to transform the non-ontological resource into a conceptual model. If a suitable pattern for re-engineering non-ontological resources is found, then the conceptual model is created from the non-ontological resource following the procedure established in the pattern for re-engineering. Alternatively, the software

[24] http://ontologydesignpatterns.org

library, described later in Sect. 6.5, can be used for generating the ontology automatically.

6.4.2.3 Task 2.2.b Perform an Ad Hoc Transformation

The goal of this task is to set up an ad hoc procedure that transforms the non-ontological resource into a conceptual model when a suitable pattern for re-engineering cannot be found. This ad hoc procedure may be generalized to create a new pattern for re-engineering non-ontological resources.

6.4.2.4 Task 2.3 Manual Refinement

The goal of this task is to check whether any inconsistency appears after the transformation. Software developers and ontology practitioners, with the help of domain experts, can fix manually any inconsistencies generated from the transformation.

6.4.3 Activity 3. Ontology Forward Engineering

The goal of this activity is to generate the ontology. We use the ontology levels of abstraction to depict this activity because they are directly related to the ontology development process. The conceptual model obtained in Task 2.2.a (Sect. 6.4.2.2) or 2.2.b (Sect. 6.4.2.3) is transformed into a formalized model, according to a knowledge representation paradigm such as description logics, first order logic, or F-logic. Then, the formalized model is implemented in an ontology language.

6.5 Technological Support

Our technological support consists in (1) a PR-NOR pattern library that includes the set of patterns for re-engineering non-ontological resources and the implementation of (2) NOR$_2$O, a software library that implements the transformation process suggested by the patterns.

Table 6.2 Template of pattern for re-engineering non-ontological resource

Slot	Value
General information	
Name	Name of the pattern
Identifier	An acronym composed of component type + abbreviated name of the component + number
Component type	Pattern for re-engineering non-ontological resource (PR-NOR)
Use case	
General	Description in natural language of the re-engineering problem addressed by the pattern for re-engineering non-ontological resources
Example	Description in natural language of an example of the re-engineering problem
Pattern for re-engineering non-ontological resource	
Input: resource to be re-engineered	
General	Description in natural language of the non-ontological resource
Example	Description in natural language of an example of the non-ontological resource
Graphical representation	
General	Graphical representation of the non-ontological resource
Example	Graphical representation of the example of non-ontological resource
Output: designed ontology	
General	Description in natural language of the ontology created after applying the pattern for re-engineering the non-ontological resource
Graphical representation	
(UML) General solution ontology	Graphical representation, using the UML profile (Brockmans and Haase 2006), of the ontology created for the non-ontological resource being re-engineered
(UML) Example solution ontology	A graphical representation example, which uses the UML profile (Brockmans and Haase 2006), of the ontology created for the non-ontological resource being used
Process: how to re-engineer	
General	Algorithm for the re-engineering process
Example	Application of the algorithm to the non-ontological resource example
Time complexity	The time complexity of the algorithm
Additional notes	Additional notes of the algorithm
Formal transformation	
General	Formal description of the transformation made with the formal definitions of the resources
Relationships (optional)	
Relations to other modeling components	Description of any relation to other PR-NOR patterns or other ontology design patterns

Table 6.3 Set of patterns for re-engineering NORs

N	Identifier	Type of NOR	NOR data model	Target
1	PR-NOR-CLTX-01	Classification scheme	Path enumeration	Ontology schema (TBox)
2	PR-NOR-CLTX-02	Classification Scheme	Adjacency list	Ontology schema (TBox)
3	PR-NOR-CLTX-03	Classification scheme	Snowflake	Ontology schema (TBox)
4	PR-NOR-CLTX-04	Classification scheme	Flattened	Ontology schema (TBox)
5	PR-NOR-CLAX-10	Classification scheme	Path enumeration	Ontology (TBox + ABox)
6	PR-NOR-CLAX-11	Classification scheme	Adjacency list	Ontology (TBox + ABox)
7	PR-NOR-CLAX-12	Classification scheme	Snowflake	Ontology (TBox + ABox)
8	PR-NOR-CLAX-13	Classification scheme	Flattened	Ontology (TBox + ABox)
9	PR-NOR-TSTX-01	Thesaurus	Record based	Ontology Schema (TBox)
10	PR-NOR-TSTX-02	Thesaurus	Relation based	Ontology Schema (TBox)
11	PR-NOR-TSAX-10	Thesaurus	Record based	Ontology (TBox + ABox)
12	PR-NOR-TSAX-11	Thesaurus	Relation based	Ontology (TBox + ABox)
13	PR-NOR-LXTX-01	Lexicon	Record based	Ontology schema (TBox)
14	PR-NOR-LXTX-02	Lexicon	Relation based	Ontology schema (TBox)
15	PR-NOR-LXAX-10	Lexicon	Record based	Ontology (TBox + ABox)
16	PR-NOR-LXAX-11	Lexicon	Relation based	Ontology (TBox + ABox)

6.5.1 Patterns for Re-engineering Non-ontological Resources

In this section, we introduce the 16 patterns that perform the transformations of NORs into ontologies. Patterns for re-engineering NORs (PR-NOR) define a procedure that transforms the NOR terms into ontology representational primitives.

Next, we present the template proposed that describes the patterns for re-engineering non-ontological resources (PR-NOR). We have modified the tabular template used in Villazón-Terrazas et al. (2008) for describing the PR-NORs. The meaning of each field is shown in Table 6.2.

According to the NOR categorization presented in Sect. 6.2, we propose patterns for re-engineering classification schemes, thesauri, and lexicons (see Table 6.3). For every data model, we can define a process with a well-defined sequence of activities in order to extract the NOR terms and then to map these terms to a conceptual model of an ontology. This process is expressed as an algorithm. Moreover, it is worth mentioning that we refer to *ontology schema* as TBox, and just *ontology* as TBox and ABox. These patterns are included in the ODP Portal[25].

The re-engineering patterns take advantage of the use of the ontology design patterns[26] for creating the ontology code. So, most of the code generated follows the best practices already identified by the community (see section *Process* on Table 6.2).

[25] http://ontologydesignpatterns.org

[26] Ontology design patterns are included in the ODP portal. The ODP portal is a Semantic Web portal dedicated to ontology design best practices for the Semantic Web, emphasizing particularly ontology design patterns (OPs)

Although we have identified five types of NORs, here we just list patterns for re-engineering classification schemes, thesauri, and lexica (see Table 6.3).

6.5.1.1 Semantics of the Relations Among the NOR Terms

The TBox transformation approach converts the resource content into an ontology schema. TBox transformation tries to impose a formal semantics on the resource by making explicit the semantics hidden in the relations of the NOR terms. To this end, each NOR term is mapped to a class, and then, the semantics of the relations among those entities must be discovered and then made explicit. Thus, patterns that follow the TBox transformation approach must discover first the semantics of the relations among the NOR terms. To perform this task, we rely on WordNet, which organizes the lexical information into meanings (senses) and synsets. What makes WordNet remarkable is the existence of various relations explicitly declared between the word forms (e.g., lexical relations, such as synonymy and antonymy) and the synsets (meaning to meaning or semantic relations, e.g., hyponymy/hypernymy relation, meronymy relation). Here, we want to prove that we can rely on an external resource for making explicit the relations. For this purpose, first, we rely on WordNet, and then, as a future line of this work, we may rely on other information resources, such as DBpedia[27].

Algorithm 1 describes how to make explicit the semantics of the relations in the NOR terms. The abbreviation of the algorithm name is *getRelation*.

6.5.2 *NOR$_2$O*

This section presents NOR$_2$O, a Java library that implements the transformation process suggested by the patterns for re-engineering non-ontological resources (PR-NOR). The library performs the ETL process[28] for transforming the non-ontological resource components into ontology terms. A high-level conceptual architecture diagram of the modules involved is shown in Fig. 6.7.

Algorithm 1 Discovering the semantics of the relations – *getRelation*

```
Take two related terms from the NOR, ti and tj
defaultRelation ← userDefinedRelation
if contains(ti,tj) then
    relation ← ti.subClassOf.tj
```

(continued)

[27] http://www.dbpedia.org/

[28] Extract, transform, and load (ETL) of legacy data sources is a process that involves (1) extracting data from the outside resources, (2) transforming data to fit operational needs, and (3) loading data into the end target resources (Kimball and Caserta 2004).

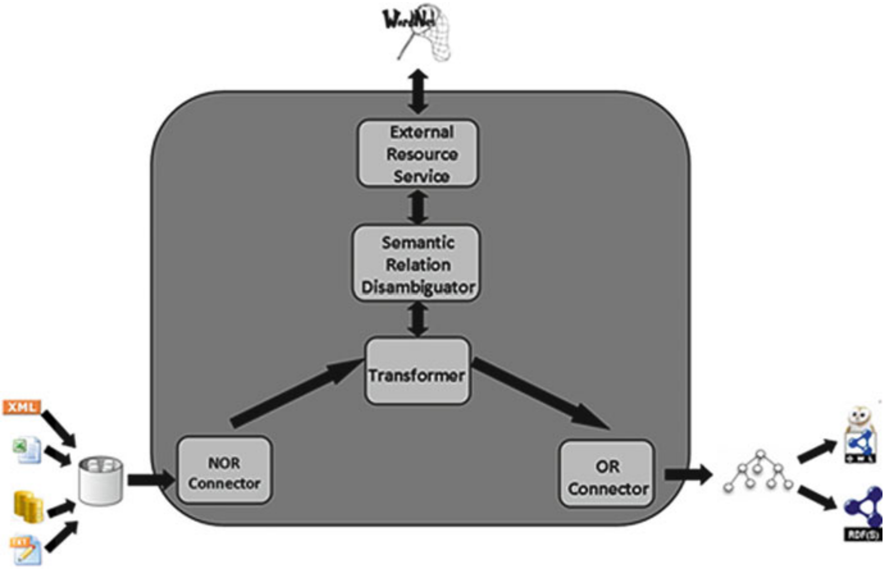


Fig. 6.7 Modules of the NOR$_2$O software library

Algorithm 1 Discovering the semantics of the relations – *getRelation*

```
else if contains(tj,ti) then
    relation ← tj.subClassOf.ti
else
    wordnetRelation ← WordNet(ti, tj)
    if wordnetRelation = = hyponym then
        relation ← ti.subClassOf.tj
    else if wordnetRelation = = hypernym then
        relation ← tj.subClassOf.ti
    else if wordnetRelation = = meronym then
        relation ← ti.partOf.tj
    else if wordnetRelation = = holonym then
        relation ← tj.partOf.ti
    else
        relation ← defaultRelation
    end if
end if
return relation
```

Figure 6.7 depicts the modules of the PR-NOR software library: NOR `Connector`, `Transformer`, `Semantic Relation Disambiguator`, `External Resource Service`, and `OR Connector`. In the following sections, these modules are described in detail. For illustrating the modules, the example of the transformation of the ASFA thesaurus[29] into an ontology schema[30] is provided.

6.5.2.1 NOR Connector

The NOR Connector loads classification schemes, thesauri, and lexicons modeled with their corresponding data models, and implemented in databases, XML, flat files, and spreadsheets.

This module utilizes an XML configuration file for describing the NOR. An example of the XML configuration file is presented in Listing 6.1. The Listing shows how the file describes a thesaurus. The thesaurus has two schema entities, *Term* and *NonPreferredTerm*, is modeled following the record-based data model, and is implemented in XML.

Listing 6.1 NOR Connector configuration file example

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

[29] http://www4.fao.org/asfa/asfa.htm

[30] http://mccarthy.dia.fi.upm.es/ontologies/asfa.owl

6.5.2.2 Transformer

This module performs the transformation suggested by the patterns by implementing the sequence of activities included in the patterns. The module transforms the NOR elements, loaded by the `NOR Connector` module, into internal model representation elements. It also interacts with the `Semantic Relation Disambiguator` module for obtaining the suggested semantic relations of the NOR elements.

The `Transformer` also utilizes an XML configuration file, called prnor. xml, for describing the transformation between the NOR elements and the ontology elements. This XML configuration file has only one section, *PRNOR*, which includes the description of the transformation from the NOR schema components (e.g., schema entities, attributes, and relations) into the ontology elements (e.g., classes, object properties, data properties, and individuals). Additionally, it indicates the transformation approach (e.g., TBox, ABox, or Population).

Two examples of the XML configuration file are shown in Listings 6.2 and 6.3.

Listing 6.2 indicates that the pattern follows the TBox transformation approach and that it transforms the elements of the *CSItem* schema component into ontology classes. Also, by default, it transforms the *subType* schema relation into a *subClassOf* relation and the *superType* schema relation into a *superClassOf* relation, unless the `Semantic Relation Disambiguator` module suggests another relation.

Listing 6.2 PR-NOR Connector configuration file example – Classification Scheme

```
<Prnor identifier=”PR–NOR–CLTX–01” transformationApproach=”TBox”
 topLevelClass=”Protection_Activities” externalResource=”WordNet”>
        <Class from=”CSItem” identifier=”[CSName].␣.[CSIdentifier]”>
                <ObjectProperty from=”subType” to=”subClassOf”/>
                <ObjectProperty from=”superType” to=”superClassOf”/>
        </Class>
</Prnor>
```

Listing 6.3 indicates that the pattern follows the TBox transformation approach and that it transforms the elements of the *Term* schema component into ontology classes. Also, by default, it transforms the *NT* schema relation into a *superClassOf* relation, the *RT* schema relation into a *relatedTerm* relation, and the *BT* schema relation into a *subClassOf* relation, unless the `Semantic Relation Disambiguator` module suggests another relation. Finally, the *UF* schema relation is transformed into a *rdfs:label*, and the module uses WordNet as external resource for disambiguation.

Listing 6.3 PR-NOR Connector configuration file example – Thesaurus

```
<Prnor identifier=”PR–NOR–TSTX–01” transformationApproach=”TBox”
 externalResource=”WordNet”>
          <Class from=”Term” identifier=”[Identifier]”>
                    <ObjectProperty from=”NT” to=”superClassOf”/>
                    <ObjectProperty from=”RT” to=”relatedTerm”/>
                    <ObjectProperty from=”BT” to=”subClassOf”/>
                    <ObjectProperty from=”UF” to=”rdfs:label”/>
         </Class>
</Prnor>
```

6.5.2.3 Semantic Relation Disambiguator

This module is in charge of obtaining the semantic relation between two NOR elements. Basically, the module receives two NOR elements from the `Transformer` module and returns the semantic relation between them. First, the module verifies whether it can obtain the *subClassOf* relation by identifying attribute adjectives[31] within the two given elements of the resource. If this is not the case, then the module connects the external resource through the `External Resource Service` module to get the relation.

The TBox transformation approach converts the resource content into an ontology schema. To this end, each NOR term is mapped to a class, and then the semantics of the relations among those entities is made explicit. Thus, patterns that follow the TBox transformation approach must make explicit the semantics of the relations among the NOR terms. To perform this task, we rely on WordNet, which organizes the lexical information into meanings (senses) and synsets.

Algorithm 1, presented in Sect. 6.5.1.1, describes how to make explicit the semantics of the relations in the NOR terms.

It is worth mentioning that, when asserting the *partOf* relation the algorithm takes advantage of the use of the `PartOf content pattern`[32] to guarantee that the OWL code generated follows common practices in ontological engineering.

6.5.2.4 External Resource Service

The `External Resource Service` is in charge of interacting with external resources for obtaining the semantic relations between two NOR elements. At this

[31] Attributive adjectives are part of the noun phrase headed by the noun they modify, for example, happy is an attributive adjective in “happy people.” In English, the attributive adjective usually precedes the noun in simple phrases but often follows the noun when the adjective is modified or qualified by a phrase acting as an adverb.

[32] http://ontologydesignpatterns.org/wiki/Submissions:PartOf

moment, the module interacts with WordNet. We are now implementing the access to DBpedia[33] due to the reasons explained in Sect. 6.5.1.1.

6.5.2.5 OR Connector

The `Ontological Resource (OR) Connector generates the ontology` in OWL Lite. To this end, this module relies on the OWL API[34]. It also utilizes an XML configuration file for describing the ontology to be generated.

An example of the XML configuration file is shown in Listing 6.4. The listing indicates that the ontology generated will be stored in the *asfa.owl* file, that its name will be *asfa ontology*, and that it will be implemented in OWL.

Listing 6.4 OR Connector configuration file example

```
<Or name=”asfa␣ontology”
 ontologyURI=”http://mccarthy.dia.fi.upm.es/ontologies/asfa.owl”
 ontologyFile=”asfa.owl” implementation=”OWL”
 alreadyExist=”no” separator=”#”>
</Or>
```

Finally, to conclude the description of the software library, it is worth mentioning that the implementation of this library follows a modular approach; therefore, it is possible to extend it and include other types of NORs, data models, and implementations in a simple way, as well as to exploit other external resources for making explicit the hidden semantics in the relations of the NOR terms.

6.6 Example

In order to evaluate the methodological guidelines proposed in this chapter, we conducted two experiments in real case scenarios within the SEEMP[35] and mIO![36] projects.

6.6.1 SEEMP Project

The main objective of this project was to develop an interoperable architecture for public employment services (PES). The resultant architecture consisted of (1) a reference ontology, the core component of the system, that acts as a common “language” in the form of a set of controlled vocabularies that describes the details

[33] http://dbpedia.org/

[34] http://owlapi.sourceforge.net/

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

[36] http://www.cenitmio.es/

of a job posting; (2) a set of local ontologies, each PES uses its own local ontology, which describes the employment market in its own terms; (3) a set of mappings between each local ontology and the reference ontology; and (4) a set of mappings between the PES schema sources and the local ontologies.

In the following sections, we describe the application of our methodological guidelines for reusing and re-engineering non-ontological resources when building an occupation ontology.

6.6.1.1 Reusing Non-ontological Resources

This section presents the application of the method for reusing non-ontological resources within the SEEMP project. It shows the process we followed for selecting the non-ontological resources to be reused when building the occupation domain ontology.

Activity 1. Search Non-ontological Resources

Following the suggestions of some domain experts, we searched for the occupation classifications at (1) the Ramon Eurostat Portal[37], (2) the ONET web site[38], and (3) the companies the project partners. Thus, we found the following classifications:

- Standard Occupational Classification System (SOC)
- International Standard Classification of Occupations (ISCO-88)
- International Standard Classification of Occupations, for European Union purposes, ISCO-88 (COM)
- Occupational Information Network (ONET)
- EURES[39] proprietary occupation classification

Activity 2. Assess the Set of Candidate Non-ontological Resources

The goal of this activity was to assess the set of candidate non-ontological resources. Experts of the occupation domain, software developers, and ontology practitioners carried out this activity taking as input the set of candidate non-ontological resources.

Task 1. Extract Lexical Entries

Within this task, we extracted the lexical entries of the aforementioned occupation classifications. We developed an ad hoc extraction tool for performing automatically the extraction task.

[37] http://ec.europa.eu/eurostat/ramon/

[38] http://online.onetcenter.org/

[39] http://www.eurodyn.com/

Task 2. Calculate Precision

Since we were dealing with occupations related to the IT domain, it was impossible to cover all the IT domain occupations already identified in the ontology requirements specification document. Thus, we used a constant that represents the complete set of IT domain occupations. In this case, the cardinality of the complete set is *K*. Therefore, the intersection of the complete set with the set of terms available in the ORSD is the set of terms of the ORSD. Next, we present the precision for each occupation classification:

$$Precision = \frac{card\{\{NORLexicalEntries\} \cap \{ORSDTerminology\}\}}{card\{NORLexicalEntries\}}$$

- $SOCPrecision = \frac{6 \cap K}{26162} = \frac{6}{26162} = 0.0002$
- $ISCO - 88Precision = \frac{9 \cap K}{544} = \frac{9}{544} = 0.0165$
- $ISCO - 88COMPrecision = \frac{9 \cap K}{520} = \frac{9}{520} = 0.0173$
- $ONETPrecision = \frac{21 \cap K}{1167} = \frac{21}{1167} = 0.0179$
- $EURESPrecision = \frac{89 \cap K}{355} = \frac{89}{355} = 0.2507$

Task 3. Calculate Coverage

Again, since we were dealing with the occupations related to the IT domain, it was impossible to cover all the IT domain occupations in the ORSD. Thus, we used a constant *K* that represents the complete set of IT domain occupations. Next, we present the coverage for each occupation classification:

$$Coverage = \frac{card\{\{NORLexicalEntries\} \cap \{ORSDTerminology\}\}}{card\{ORSDTerminology\}}$$

- $SOCPrecision = \frac{6 \cap K}{K} = \frac{6}{K}$
- $ISCO - 88Precision = \frac{9 \cap K}{K} = \frac{9}{K}$

Table 6.4 Assessment table for SEEMP occupation standards

NOR	Precision	Coverage	Consensus
SOC	0.0002	6/K	No
ISCO-88	0.0165	9/K	No
ISCO-88 COM	0.0173	9/K	Yes
ONET	0.0179	21/K	No
EURES	0.2507	89/K	Yes

- $ISCO - 88COMPrecision = \frac{9 \cap \mathrm{K}}{\mathrm{K}} = \frac{9}{\mathrm{K}}$
- $ONETPrecision = \frac{21 \cap \mathrm{K}}{\mathrm{K}} = \frac{21}{\mathrm{K}}$
- $EURESPrecision = \frac{89 \cap K}{K} = \frac{89}{K}$

Task 4. Evaluate the Consensus

It was important for the project that resources focused on the current European reality because the user partners involved in SEEMP are European, and the outcoming prototype has to be validated in European scenarios. Thus, domain experts confirmed whether the resources were built with the consensus of the European community or not. They also explained that ISCO-88(COM) and EURES proprietary occupation classification contains terminology that had already reached a consensus.

Table 6.4 summarizes all the information of each non-ontological resource.

Activity 3. Select the Most Appropriate Non-ontological Resources

Following Table 6.1 we selected a non-ontological resource, the EURES proprietary occupation classification.

We followed the same process for selecting the non-ontological resources when building the remaining ontologies. We provide a table (see Table 6.5) that summarizes the selection of standards, codes, and classifications accomplished for building every domain ontology.

6.6.1.2 Re-engineering Non-ontological Resources

In this section, we present the application of the method for re-engineering non-ontological resources within the SEEMP project. Once we select the non-ontological resource, we have to transform it into an ontology. Next, we describe the process of generating an occupation ontology from the EURES proprietary occupation classification.

Table 6.5 Standards, codes, and classifications reused

Domain	Candidate standards/classifications	Selected standards/ classifications	Justification
Economic sector	ISIC, NACE, NAICS	NACE	Best coverage and European scope
Education fields	ISCED 97, FOET	FOET	Best coverage and European scope
Education levels	ISCED 97	ISCED 97	Worldwide scope, widely accepted
Currency	Pacific exchange, ISO 4217, WordAtlas	ISO 4217	Worldwide scope, widely accepted
Geographic	ISO 3166, Regions of the World	ISO 3166	Worldwide scope, widely accepted
Language	ISO 639	ISO 639	Worldwide scope, widely accepted
Language levels	CEFR	CEFR	European scope, widely accepted
Driving licence	EU driving licence	EU driving licence	European legislation
Skills	EURES	EURES	Coverage and European scope
Contract types	LE FOREM proprietary classification, ARL proprietary classification	Mix of both classifications	Acceptable coverage in SEEMP scope
Work condition	LE FOREM proprietary classification	LE FOREM proprietary classification	Acceptable coverage in SEEMP scope

Activity 1. Non-ontological Resource Reverse Engineering

In this activity, we gathered documentation on the EURES occupation classification from the European Dynamics SEEMP user partner. From this documentation, we extracted the schema of the classification scheme, which consists of two tables, *CVO OCCGROUP* and *CVO OCCUGROUP NAME*. Since the data model was not available in the documentation, it was necessary to extract it for the resource implementation itself. The EURES occupation classification is modeled following the snowflake data model and is implemented in a MS Access database.

Activity 2. Non-ontological Resource Transformation

Within this activity, we carried out the following tasks:

1. We identified the transformation approach, the TBox transformation, i.e., transforming the resource content into an ontology schema.
2. Then, we searched our local pattern repository for a suitable pattern to re-engineer NORs, taking into account the transformation approach (TBox

Table 6.6 Resources transformed in the SEEMP project

Resource	Type	Data model	Implementation	Pattern used
NACE	Classification scheme	Path enumeration	Database	PR-NOR-CLTX-01
FOET	Classification scheme	Path enumeration	Database	PR-NOR-CLTX-01
ISCED 97	Classification scheme	Adjacency list	Database	PR-NOR-CLTX-02
ISO 4217	Classification scheme	Snowflake	XML	PR-NOR-CLAX-12
ISO 3166	Classification scheme	Snowflake	XML	PR-NOR-CLAX-12
ISO 639	Classification scheme	Snowflake	XML	PR-NOR-CLAX-12
CEFR	Classification scheme	Proprietary model	Proprietary format	
EU driving licence	Classification scheme	Snowflake	Proprietary format	
EURES skill	Classification scheme	Path enumeration	Database	PR-NOR-CLTX-01
LE FOREM contracts	Proprietary classification	Proprietary model	Proprietary format	

transformation), the non-ontological resource type (classification scheme), and the data model (snowflake data model) of the resource.

3. The most appropriate pattern found for this case was the PR-NOR-CLTX-03 pattern. This pattern takes as input a classification scheme modeled with a snowflake data model and produces an ontology schema.

Activity 3. Ontology Forward Engineering

WSML[40] is the ontology implementation language used in the SEEMP project. Because of the number of occupations of the EURES classification, it was not practical to create the ontology manually. Therefore, we created an ad hoc wrapper, implemented in Java, that reads the data from the resource implementation and automatically creates the corresponding classes and relations of the new ontology following the suggestions given by the pattern for re-engineering NORs and the conceptual model.

We followed this process for all the resources identified, being the patterns used those presented in Table 6.6.

[40] http://www.wsmo.org/wsml/

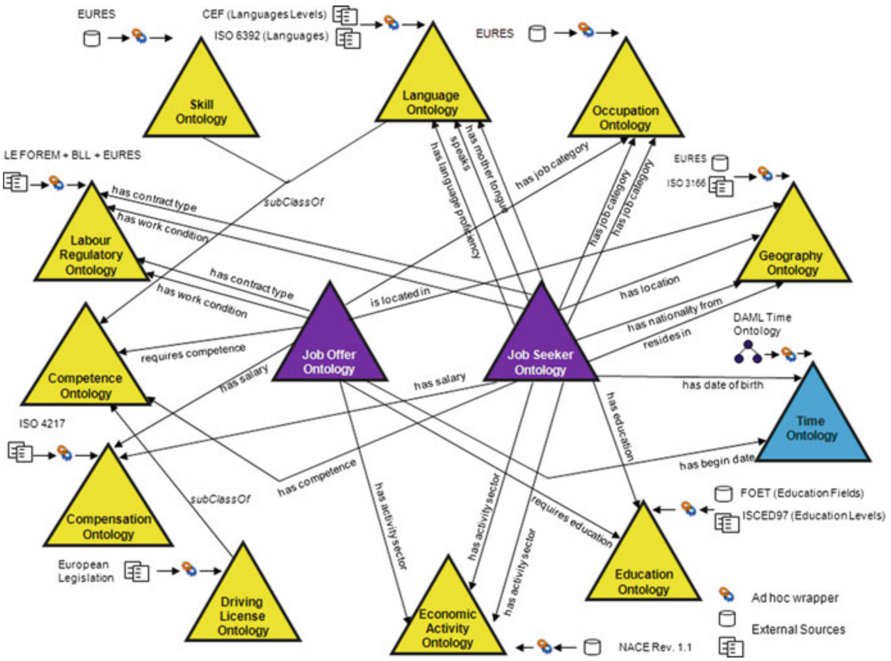


Fig. 6.8 SEEMP reference ontology

Table 6.7 SEEMP reference ontology statistical data

Ontology	Concepts	Attributes	Axioms	Instances	Efforts (man.months)
SEEMP RO	1,985	315	1,037	1,449	6

6.6.1.3 Analysis of the Applicability of the Method

The SEEMP Reference Ontology (SEEMP RO) was developed following the method for reusing and re-engineering non-ontological resources. It is composed of 13 modular ontologies: *competence, compensation, driving licence, economic activity, education, geography, job offer, job seeker, labur regulatory, language, occupation, skill, and time*. The main sub-ontologies are the *job offer* and *job seeker*, which are intended to represent the structure of a job posting and a CV, respectively. While these main two sub-ontologies were built taking as a starting point some HRXML recommendations, the others derived from some available international standards (like NACE, ISCO-88 (COM), FOET, etc.), employment services classifications, and international codes (like ISO 3166, ISO 6392, etc.) that best fitted the European requirements. Figure 6.8 presents these 13 modular ontologies (each ontology is represented by a triangle), 10 of which were obtained after re-engineering the standard/classification. The SEEMP Reference Ontology is available at http://oeg-upm.net/index.php/en/ontologies/99-hrmontology.

In order to illustrate the dimension of the ontology and the ontological engineers' efforts required to build it, some statistical data are shown in Table 6.7.

Our experience in SEEMP has shown us that the approach of building ontologies by reusing and re-engineering non-ontological resources already agreed upon allows building ontologies faster, with less resources, and with an immediate consensus. This approach permits making explicit the knowledge implicitly coded in organization models and standards. By building ontologies in this fashion, we facilitate that ontologies become reference ontologies in their respective domains.

With respect to the application of the method for reuse and re-engineering, this was especially useful for guiding the steps of the ontological engineers involved since this method provides detailed and sufficient guidelines. In addition, the existence of a well-defined and structured process for building the ontology network in the e-employment domain eased the planning, coordination, and communication with other non–Semantic Web members of the development team, which in turn helped to convey reliability to the final result.

6.6.2 mIO! *Project*

The main objective of the *mIO!* project is to develop ubiquitous services in an intelligent environment, adapted to every user and its context by means of mobile interfaces. The project relies on ontologies for modeling the knowledge.

The following sections describe the application of our methodological guidelines for reusing and re-engineering non-ontological resources when building a geographical ontology, which includes continents, countries, and regions.

6.6.2.1 Reusing Non-ontological Resources

This section describes the activities carried out for reusing non-ontological resources.

Activity 1. Search Non-ontological Resources

Following some of the suggestions made by the domain experts, we searched geographical location resources on highly reliable web sites. Next, we list the geographic location classifications:

- ISO 3166[41] Maintenance Agency (ISO 3166/MA) ISO's focal point for country codes

[41] http://www.iso.org/iso/en/prods-services/iso3166ma/index.html

- Guide to regions of the world[42]
- Regions of the world[43]

Activity 2. Assess the Set of Candidate Non-ontological Resources

Once we had the set of candidate non-ontological resources, we needed to assess them according to the following criteria: precision, coverage, consensus, and quality of the resources.

Task 2.1 Extract Lexical Entries
Within this task, we extracted the lexical entries of the aforementioned geographic location classifications. For this purpose, we used TreeTagger[44], a syntactic annotator.
Task 2.2 Calculate Precision
It was impossible to cover all the geographic locations in the ORSD. Thus, we used a constant K that represents the cardinality of the complete set of geographical locations. Next, we present the precision for each geographic location classification:

$$Precision = \frac{card\{\{NORLexicalEntries\} \cap \{ORSDTerminology\}\}}{card\{NORLexicalEntries\}}$$

- $ISO3166 = \frac{195 \cap K}{200} = \frac{195}{200} = 0.975$
- $GuidetoregionsoftheWorld = \frac{102 \cap K}{193} = \frac{102}{193} = 0.528$
- $RegionsoftheWorld = \frac{110 \cap K}{154} = \frac{110}{154} = 0.714$

Task 2.3 Calculate Coverage
Again, it was impossible to cover all the geographic locations in the ORSD. Thus, we used a constant K that represents the cardinality of the complete set of

[42] http://www.countriesandcities.com/regions/

[43] http://park.org/Regions/

[44] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

geographic locations. Next, we present the coverage for each geographic location classification:

$$Coverage = \frac{card\{\{NORLexicalEntries\} \cap \{ORSDTerminology\}\}}{card\{ORSDTerminology\}}$$

- $ISO3166 = \frac{195 \cap K}{K} = \frac{195}{K}$
- $GuidetoregionsoftheWorld = \frac{102 \cap K}{K} = \frac{102}{K}$
- $RegionsoftheWorld = \frac{110 \cap K}{K} = \frac{110}{K}$

Task 2.4 Evaluate the Consensus
It was important for the project that resources focused on the current worldwide reality because the outcoming prototype will be validated by users.

Thus, domain experts evaluated whether the resource was built with the consensus of the worldwide community or not. They confirmed that ISO 3166 has the full consensus of the community, whereas the other resources have not.

Task 2.5 Evaluate the Quality
In this case, domain experts evaluated whether the resource was built with an acceptable level of quality. They confirmed that ISO 3166 has an acceptable level of quality.

Task 2.6 Build the Assessment Table
Table 6.8 summarizes all the information related to each non-ontological resource.

Activity 3. Select the Most Appropriate Non-ontological Resources

According to Table 6.8, we selected the following non-ontological resource: ISO 3166.

Table 6.8 Assessment table for the *mIO!* geographical locations

NOR	Precision	Coverage	Consensus	Quality
ISO 3166	0.975	195/K	Yes	Yes
Guide to regions of the World	0.528	102/K	No	No
Regions of the World	0.714	110/K	No	No

6.6.2.2 Re-engineering Non-ontological Resources

This section presents the application of the method for re-engineering non-ontological resources within the *mIO!* project. Once we have the non-ontological resource selected, the ISO 3166, we had to transform it into an ontology. Next, we describe the process of generating a geographical location ontology.

Activity 1. Non-ontological Resource Reverse Engineering

In this activity, we gathered documentation about ISO 3166 from its web site. From this documentation, we extracted the schema of the classification scheme, which consists of one entity *ISO 31661 Entry*. Since the data model was not available in the documentation, it was necessary to extract it for the resource implementation itself. ISO 3166 is modeled following the snowflake data model and implemented in XML.

Activity 2. Non-ontological Resource Transformation

In this activity, we carried out the following tasks:

1. We identified the transformation approach, the ABox transformation, i.e., the transformation of the resource schema into an ontology schema, and the resource content into ontology instances.
2. Then we searched our local pattern repository for a suitable pattern to re-engineer NORs, taking into account the transformation approach (ABox transformation), the non-ontological resource type (classification scheme), and the data model (snowflake data model) of the resource.
3. The most appropriate pattern for this case is the PR-NOR-CLAX-12 pattern. This pattern takes as input a classification scheme modeled with a snowflake data model.
4. Finally, we followed the procedure defined by the pattern selected for transforming the resource components into ontology elements.

Activity 3. Ontology Forward Engineering

In this activity, we formalized and implemented the ontology in OWL. The ontology is available at http://mccarthy.dia.fi.upm.es/ontologies/.

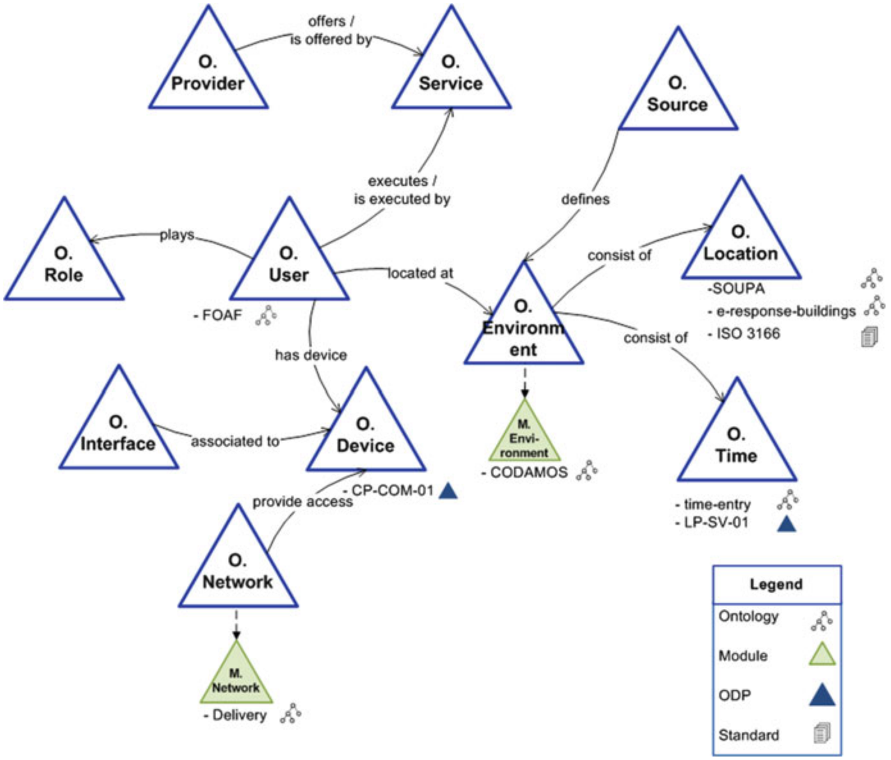


Fig. 6.9 *mIO!* Ontology network

Table 6.9 *mIO!* Ontology statistical data

Ontology	Concepts	Attributes	Axioms	Instances	Efforts (man.months)
mIO! ontology	432	276	154	120	3

6.6.2.3 Analysis of the Applicability of the Method

The network of ontologies of the *mIO!* project was developed following the NeOn Methodology (Suárez-Figueroa et al. 2008). This ontology is composed of 11 modular ontologies: *provider*, *service*, *source*, *geographical location*, *environment*, *time*, *device*, *user*, *network*, *interface*, and *role*. Only the geographical location ontology was built according to the method for reusing and re-engineering non-ontological resources. The other ontologies were built by reusing available ontologies or modules.

Figure 6.9 presents the *mIO!* ontology network and includes the location subontology. The ontology network is available at http://oeg-upm.net/ index.php/en/ ontologies/82-mio-ontologies

In order to illustrate the dimension of the ontology and the efforts required by the ontological engineers to build it, we outline some data in Table 6.9.

Our experience in *mIO!* has served us to demonstrate that the approach of building ontologies by reuse and re-engineering non-ontological resources already agreed upon allows building ontologies faster, with less resources, and with consensus. With respect to the application of the method for reuse and re-engineering, this was especially useful for guiding the steps of the ontological engineers involved since the method provides detailed and sufficient guidelines.

6.7 Conclusions

In this chapter, we have provided a method and its technological support that rely on re-engineering patterns in order to speed up the ontology development process by reusing and re-engineering as much as possible available non-ontological resources. Moreover, we have introduced a three-level categorization of NORs according to three different features: type of NOR, data model, and implementation. Finally, we have presented two use cases of the proposed approach.

References

Baeza-Yates Ricardo, Ribeiro-Neto Berthier (1999) Modern information retrieval, 1st edn. Addison Wesley, Harlow. ISBN 020139829X

Brandon D (2005) Recursive database structures. J Comput Sci Coll 1:295–304

Brockmans S, Haase P (2006) A metamodel and UML profile for networked ontologies. A complete reference. Technical report, University Karlsruhe, 2006

Carkenord B (2002) Why build a logical data model. http://www.embarcadero.com/resources/tech papers/datamodel.pdf

Gangemi A, Pisanelli D, Steve G (1998) Ontology integration: experiences with medical terminologies. Ontol Inf Syst 1:163–178

Gangemi A, Guarino N, Masolo C, Oltramari A (2003) Sweetening WORDNET with DOLCE. AI Mag 24(3):13–24, ISSN 0738–4602

Gangemi A, Catenacci C, Ciaramita M, Lehmann J (2006) Modelling ontology evaluation and validation. In: Proceedings of the 3rd European Semantic Web Conference (ESWC2006), LNCS, vol 4011. Springer, Budva, 2006

García R, Celma O (2005) Semantic integration and retrieval of multimedia metadata. In: Proceedings of the ISWC 2005 workshop on knowledge markup and semantic annotation (Semannot'2005), Galway, Ireland

Gómez-Pérez A, Manzano-Macho D (2004) An overview of methods and tools for ontology learning from texts. Knowl Eng Rev 19(3):187–212. ISSN 0269–8889, doi: http://dx.doi.org/ 10.1017/ S0269888905000251

Gómez-Pérez A, Fernández-López M, Corcho O (2003) Ontological engineering, Advanced information and knowledge processing. Springer, New York/London. ISBN 1–85233–551–3

Hepp M (2006) Products and services ontologies: a methodology for deriving owl ontologies from industrial categorization standards. Int J Semant Web Inf Syst 2(1):72–99

Hepp M (2007) Possible ontologies: how reality constrains the development of relevant ontologies. IEEE Internet Comput 11(1):90–96

Hepp M,de Brujin J (2007) GenTax: a generic methodology for deriving OWL and RDF-S ontologies from hierarchical classifications, thesauri, and inconsistent taxonomies. In: Proceedings of the 4th European Semantic Web Conference (ESWC2007). Springer, Innsbruck

Hirst G (2004) Ontology and the lexicon. In: Handbook on ontologies in information systems. Springer, Berlin, pp 209–230

Hodge G (2000) Systems of knowledge organization for digital libraries: beyond traditional authority files. http://www.clir.org/pubs/reports/pub91/contents.html

Hyvöonen E, Viljanen K, Tuominen J, Seppöalöa K (2008) Building a national semantic web ontology and ontology service infrastructure -the FinnONTO approach. In: ESWC, vol 1. Springer, Heidelberg, pp 95–109

ISO 2788 (1986) Documentation – guidelines for the establishment and development of monolingual thesaurus. International Standard Organization (ISO), Report ISO 2788

ISO/IEC FDIS 11179–1 (2004) Information technology – metadata registries – part 1: framework. International Standard Organization (ISO), Report ISO/IEC FDIS 11179–1

Kimball R, Caserta J (2004) The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data. Wiley, New York. ISBN 0764567578

Maedche A, Staab S (2001) Ontology learning for the semantic web. IEEE Intell Syst 16:72–79

Malinowski E, Zimányi E (2006) Hierarchies in a multidimensional model: from conceptual modeling to logical representation. Data Knowl Eng 59:348–377

Pinto S, Tempich C, Staab S (2004) DILIGENT: towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In: Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004). IOS Press, Amsterdam, Washington, DC, pp 393–397, ISBN 1–58603–452–9

Sabou M, Angeletou S, d'Aquin M, Barrasa J, Dellschaft K, Gangemi A, Lehman J, Lewen H, Maynard D, Mladenic D, Nissim M, Peters W, Presutti V, Villazón-Terrazas B (2007) Selection and integration of reusable components from formal or informal specifications. Technical report, NeOn project deliverable D2.2.1, 2007

Schnurr H-P, Studer R, Sure Y (2001) Knowledge processes and ontologies. IEEE Intell Syst 1(16):26–34

Soergel D (1995) Data models for an integrated thesaurus database. Compat Integr Order Syst 24(3):47–57

Suárez-Figueroa MC, Aguado de Cea G, Buil C, Dellschaft K, Fernández-López M, García-Silva A, Gómez-Pérez A, Herrero G, Montiel-Ponsoda E, Sabou M, Villazón-Terrazas B, Yufei Z (2008) NeOn Methodology for building contextualized ontology networks. Technical report, NeOn project deliverable D5.4.1, 2008

Suárez-Figueroa MC, Gómez-Pérez A, Villazón- Terrazas B (2009) How to write and use the ontology requirements specification document. In: OTM Conferences (2), pp 966–982, 2009

Villazón-Terrazas B, Angeletou S, García-Silva A, Gómez-Pérez A, Maynard D, Suárez-Figueroa MC, Peters W (2008) NeOn deliverable D2.2.2 methods and tools for supporting reengineering. Technical report, NeOn, 2008

Villazón-Terrazas B, Suárez-Figueroa MC, Gómez-Pérez A (2010) A pattern-based method for re-engineering non-ontological resources into ontologies. Int J Semant Web Inf Syst 6(4):27–63

Wright SE, Budin G (eds) (1997) Handbook of terminology management, basic aspects of terminology management. John Benjamins Publishing Company, Amsterdam

Chapter 7
Ontology Development by Reuse

Mariano Fernández-López, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez

Abstract This chapter presents methodological guidelines that allow engineers to reuse generic ontologies. This kind of ontologies represents notions generic across many fields, (*is part of*, *temporal interval*, etc.). The guidelines helps the developer (a) to identify the type of generic ontology to be reused, (b) to find out the axioms and definitions that should be reused and (c) to adapt and integrate the generic ontology selected in the domain ontology to be developed. For each task of the methodology, a set of heuristics with examples are presented. We hope that after reading this chapter, you would have acquired some basic ideas on how to take advantage of the great deal of well-founded explicit knowledge that formalizes generic notions such as time concepts and the *part of* relation.

7.1 Introduction

Ontologies play an important role in many knowledge-intensive applications, by formally defining the conceptualization used by the application and by facilitating interoperability. Building ontologies from scratch can in general be expensive. In this sense, one way of reducing the time and costs associated with the ontology development process is by reusing available ontological resources. Ontologies developed by reuse can also build on existing good practices (from well-developed ontologies), thus increasing the overall quality of the results.

M. Fernández-López (✉)
Escuela Politécnica Superior, Universidad San Pablo CEU, Urbanización Montepríncipe sn., 28668 Boadilla del Monte, Madrid, Spain
e-mail: mfernandez.eps@ceu.es

M.C. Suárez-Figueroa • A. Gómez-Pérez
Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo sn., 28660 Boadilla del Monte, Madrid, Spain
e-mail: mcsuarez@fi.upm.es; asun@fi.upm.es

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_7, © Springer-Verlag Berlin Heidelberg 2012

As mentioned in Chap. 2, the NeOn Methodology presents nine scenarios for building networks of ontologies. One of these scenarios is *Building Ontology Networks by Reusing Ontological Resources*. In this scenario, ontology developers analyze whether existing ontological resources can be reused in the context of building an ontology.

The reuse of ontological resources is encouraged by a recent increase in the number of ontologies available online.

According to our experience, the reuse of ontological resources is useful for (a) saving time and resources during the ontology development and (b) refining the Ontology Requirement Specification Document (ORSD) (see Chap. 5) taking into account the knowledge represented in the candidate ontological resources to be reused. The latter case refers to the situation in which the engineer finds axioms and/or definitions of terms that did not appear in the ORSD. For example, in the development of a drug ontology, the engineer may find a type of drug that had not been considered in the ORSD. For the sake of simplicity, in this chapter, it is assumed that the reuse does not imply modifications in the ORSD. If such modifications are required, an iterative-incremental life cycle model should be followed (see Chap. 2).

The ontological resource reuse process is often influenced by the type of ontology to be reused. Ontologies can model domain entities (e.g., drug, disease, pharmaceutical product) or generic entities, which are considered to be generic across many fields (van Heijst et al. 1997). For example, the *part of* relation can be used to link objects in the mechanical domain (the spark plug is part of the motor) and also in the domain of cultural activities (the interpretation of Radetzky March is part of the New Year Concert). Hence, such generic ontologies can be reused in a wider range of domains.

However, the reuse of large ontologies such as WordNet[1] or the NCI ontology (Golbeck et al. 2003) can cause difficulties because they tend to contain far more definitions than most applications would normally need. Hence, in the context of a reuse process, sometimes elements of an ontology (e.g., modules or statements) have to be extracted first, to be integrated in the new ontology (d'Aquin M et al. 2007b). For this reason, different levels of granularity in the reuse of ontologies can be distinguished:

- Ontologies can be reused as a whole if they closely meet the expectations and the needs of the ontology engineer.
- In certain cases, only one part or *module*[2] of an ontology is relevant for reuse. For example, when building an ontology about lung cancer, it is not always necessary to reuse an entire ontology about the human body; it suffices to reuse a module describing concepts related to the lung.

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

[2] We consider a module (d'Aquin M et al. 2007b) as a part of the ontology that defines the relevant set of terms for a particular purpose.

- In other cases, only some knowledge components from the ontology (the description of a particular entity, the branch in the taxonomic hierarchy in which an entity appears, or entity neighborhoods in the ontology) are relevant for the development needs. In these cases, the reuse of ontological knowledge is performed at the *statement*[3] level, providing the ontology developer with better control over the material being reused.

This chapter focuses on providing methodological guidelines for the reuse of generic ontologies, although most of the recommendations are also applicable to the reuse of domain ontologies.

7.2 Methodological Guidelines for Reusing Generic Ontologies

Table 7.1 presents a filling card with the information concerning the generic ontology reuse process. The card includes the definition, the goal, the inputs and outputs, the performer of the process, and the time scheduled for the process.

Figure 7.1 shows the workflow and the activities for carrying out the generic ontology reuse process, that is, selecting the ontology to be reused, and customizing and integrating it in the ontology to be developed.

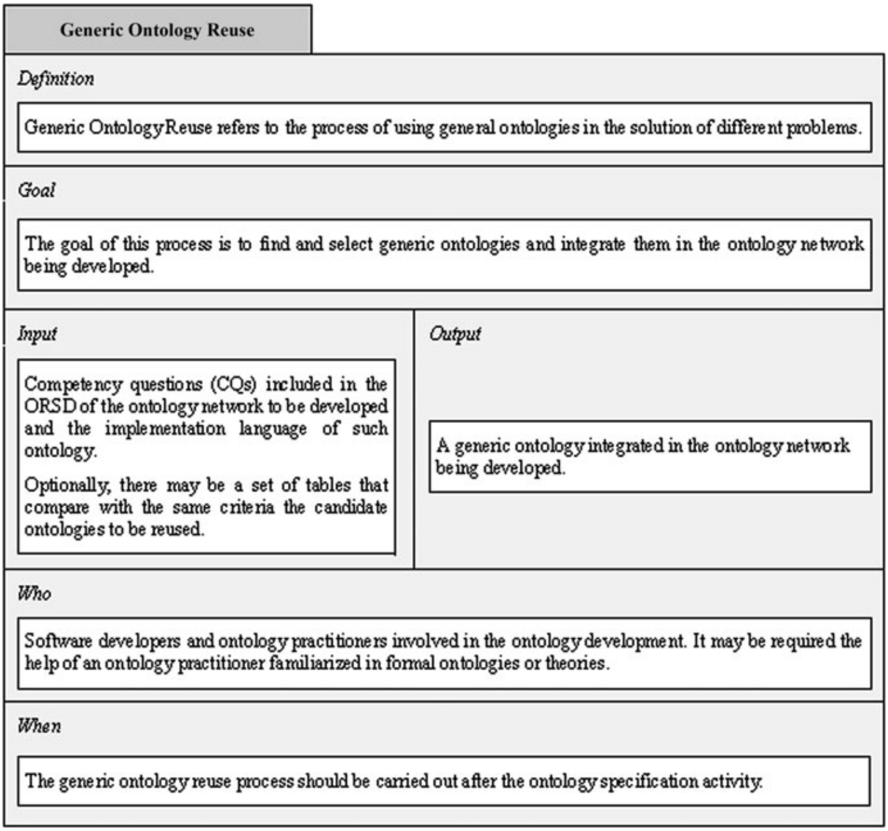
The activities shown in Fig. 7.1 are explained in more detail in the rest of the chapter. For the sake of simplicity, the different activities involved in the whole process are explained, and it is considered the reuse of just one ontology. When reusing more than one ontology, the process described should be performed iteratively.

Along the exposition, an example of reusing a generic ontology in the development of the pharmaceutical product ontology network (PPO) (see Chap. 20) is presented. This ontology will be used as a bridge between proprietary systems for managing financial and product knowledge interoperability in pharmaceutical laboratories, companies, and distributors in Spain. In this ontology reuse task, we have taken into account the four *competency questions* (CQs) shown in Table 7.2. They have been obtained from Chap. 7 of the NeOn deliverable D5.4.1 (Suárez-Figueroa et al. 2008).

The reader can find additional information on ontology reuse in (Suárez-Figueroa 2010).

[3] An ontology statement (or triple) contains the following three components: *subject*, *predicate*, and *object*.

Table 7.1 Generic ontology reuse filling card



Generic Ontology Reuse

Definition

Generic OntologyReuse refers to the process of using general ontologies in the solution of different problems.

Goal

The goal of this process is to find and select generic ontologies and integrate them in the ontology network being developed.

Input	*Output*
Competency questions (CQs) included in the ORSD of the ontology network to be developed and the implementation language of such ontology. Optionally, there may be a set of tables that compare with the same criteria the candidate ontologies to be reused.	A generic ontology integrated in the ontology network being developed.

Who

Software developers and ontology practitioners involved in the ontology development. It may be required the help of an ontology practitioner familiarized in formal ontologies or theories.

When

The generic ontology reuse process should be carried out after the ontology specification activity.

7.2.1 *Activity 1: Selecting the Generic Ontology to be Reused*

The goal of this activity is to select the most appropriate generic ontologies to be reused in the ontology being developed. It is worth mentioning that instead of reusing available ontologies, practitioners can implement from scratch the necessary axioms and definitions according to some existing formalization, for example, the one appearing in Annex. On the one hand, the advantage of reusing available ontologies implemented in a formal language is that ontology developers will save effort in the transformation of a formalization that is not suitable for run-time reasoning. On the other hand, the advantage of starting from an existing formalization is that ontology developers will save effort in the searching, comparison, and evaluation of candidate ontologies to be reused. In this chapter, we focus on the reuse option.

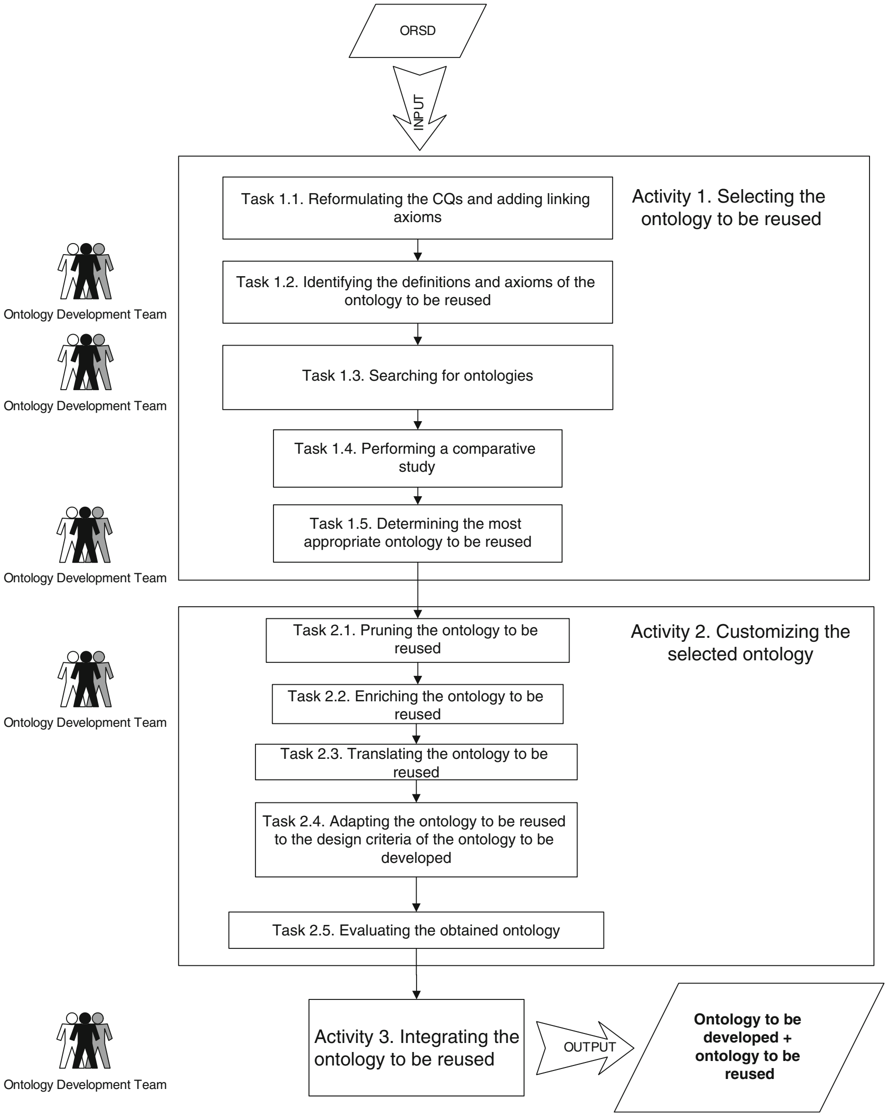

ORSD
INPUT
Task 1.1. Reformulating the CQs and adding linking axioms
Activity 1. Selecting the ontology to be reused
Task 1.2. Identifying the definitions and axioms of the ontology to be reused
Task 1.3. Searching for ontologies
Task 1.4. Performing a comparative study
Task 1.5. Determining the most appropriate ontology to be reused
Ontology Development Team
Ontology Development Team
Ontology Development Team
Task 2.1. Pruning the ontology to be reused
Activity 2. Customizing the selected ontology
Task 2.2. Enriching the ontology to be reused
Task 2.3. Translating the ontology to be reused
Task 2.4. Adapting the ontology to be reused to the design criteria of the ontology to be developed
Task 2.5. Evaluating the obtained ontology
Ontology Development Team
Activity 3. Integrating the ontology to be reused
OUTPUT
Ontology to be developed + ontology to be reused
Ontology Development Team

Fig. 7.1 Activities for reusing generic ontologies

This activity takes as input the ORSD (Chap. 5) and is divided in the following tasks:

Task 1.1 Reformulating the CQs and adding linking axioms. The main goal of this task is to reformulate the CQs included in the ORSD of the ontology that is being developed with vocabulary that could potentially belong to ontologies to be reused but that do not explicitly appear in the CQs. Additionally, another goal of

Table 7.2 Excerpt of informal host competency questions (pharmaceutical product ontology case)

CQ id	Informal CQ	Example of answer
CQ_1	What drugs do have paracetamol?	Algidol® Apiretal® Bisolgrip® Cortafriol® Dolgesic® Dolostop® Efferalgan® Frenadol® Gelocatil® Pharmagrip® Termalgin®
CQ_2	Which is the composition of Frenadol®?	Caffeine Chlorpheniramine citrate Dextrometorphan Paracetamol
CQ_3	Which is the main active ingredient of Frenadol®?	Paracetamol
CQ_4	Which substances do Frenadol® interacts with?	Ethyl alcohol Isoniazid Propranolol Rifampicin

this task is to identify axioms that link terms of the CQs to terms that could be reused. The first column of Table 7.3 shows some typical cases (case 1, case 2, and case 3[4]) that guide the engineer in transforming CQs and adding linking axioms. The third column shows the action to carry out in each case. Finally, as an example, the second and fourth columns present the PPO CQ that matches each case and the result of applying the action corresponding to the case. For example, given that the case 2 (Table 7.3) proposes to reformulate CQs using the term *is part of*, the CQ_1 (Table 7.2), *what drugs do have paracetamol?*, can be expressed as *which drugs is paracetamol part of?* Given that the term *is part of* appears in the new formulation, the engineer knows that a mereology can be reused (see Annex to review basic mereology notions).

This task is useful to make explicit abstract terms such as *is part of*, *temporal point*, and *temporal interval* that can be reused from mereological or time ontologies.

Task 1.2 Identifying the definitions and axioms of the ontology to be reused. The goal here is to identify which definitions and axioms can be potentially reused in the ontology to be developed. The terms whose definition could be reusable from other ontologies are those terms appearing in the pre-glossary of the ORSD (specifically in slot 7) (see Chap. 5) and the new terms that appear in the reformulated CQs

[4] The rest of the cases are presented in Suárez-Figueroa (2010).

Table 7.3 Analysis and transformation of the competency questions and addition of linking axioms and rules (Task 1.1)

Case	Competency question	Action to carry out	Result of the action
Case 1. Ontology developers are interested in knowing the parts of an object without including the object itself	CQ_2. Which is the composition of Frenadol®?	Reformulate the CQ to mention the term *is proper part of*	Which are the proper parts of Frenadol®?
Case 2. Ontology developers are interested in knowing the parts of an object including the object itself	CQ_1. What drugs do have paracetamol? (The inclusion of the substance itself is because paracetamol itself could be a drug)	Reformulate the CQ to mention the term *is part of*	Which drugs is paracetamol part of?
	CQ_4. Which substances do Frenadol® interacts with?		Which substances do the parts of Frenadol® interacts with?
Case 3. The CQ refers to a relation *S* that is subrelation of *isPartOf*	CQ_3. Which is the main active ingredient of Frenadol®?	Introduce a linking axiom establishing that *S* is subrelation of *is part of*	Introduce, when the mereology implementation is reused, the following axiom: *Is main active ingredient of* is a subrelation of *is part of*?

obtained in Task 1.1. The second column of Table 7.4 presents some heuristics that are useful to find mereology definitions and axioms that could be potentially reused (for the rest of the heuristics see footnote 4). Such a table shows that the properties of *is part of* that are useful for PPO are reflexivity, antisymmetry, transitivity, and the weak supplementation principle. For example, if ontology developers are interested in knowing what substances contains a particular substance (e.g., iron), they need to apply transitivity, since the substance in question can be an indirect part of the drug. For instance, iron is part of ferrous sulfite, and this is, in its turn, part of Mol Iron®, which is a drug. Moreover, the definition of the term *is proper part of* should be reused to answer questions like CQ_2, where the interest is not located in the drug itself.

Task 1.3 Search for ontologies. The ontology development team should search for ontologies that implement the axioms and definitions identified in Task 1.2.

To perform this task, ontology developers can use a general purpose search engine (e.g., Google[5]), Semantic Web search engines (e.g., Swoogle[6], Watson[7],

[5] http://www.google.es/

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

[7] http://watson.kmi.open.ac.uk

Table 7.4 Identification of definitions and axioms to reuse from a mereology (Task 1.2)

Axioms and definitions	When they are useful	The condition is fulfilled
A.1. *Is part of* reflexivity	Recommended if its implementation is possible, to ensure the right meaning of *part of*	*Yes*
A.2. *Is part of* antisymmetry	Recommended if its implementation is possible, for consistency verification	*Yes*
A.3. *Is part of* transitivity	X has parts $X_1, X_2, \ldots, X_\mathrm{n}$. In its turn, there is some X_i with parts $X_{i1}, X_{i2}, \ldots, X_{im}$. That is, X has several levels of parts. Besides, ontology developers are interested in all the levels when they ask: *which are the parts of X?*	*Yes*
D.1. *Is proper part of* definition	The case 1 (see Table 7.3) is fulfilled	*Yes*
A.4. *Is part of* weak supplementation	Recommended if its implementation is possible, for consistency verification	*Yes*

Sindice[8], Sigma[9]), repositories (e.g., the Protégé ontology library[10], the Open Biological and Biomedical Ontologies[11], and Cupboard[12]), and other known ontologies (for instance, mereology terms can be reused from Dolce-Lite[13], SUMO-OWL[14], etc.).

For example, Watson is a Semantic Web search engine developed as part of the NeOn project which provides features to search, select, and integrate ontologies available online (d'Aquin and Motta 2011). Watson collects, indexes, and provides access to ontologies crawled from the web. From a user interface perspective, it can be seen as a classical search engine, taking as input keywords (e.g., based on the ORSD) and providing as a result a list of ontologies that match these keywords, together with information about each ontology, and about entities in them that are relevant to the given keywords. Ontologies and entities can be further explored online, using the provided navigation features. As part of its indexing process, Watson also extracts information about each ontology, such as the underlying language, its size, and metadata that the corresponding file might include. Search results from Watson often include thousands of ontologies. They can be further reduced by using filters (search options) regarding the scope of the search (in local names, labels, comments, or any other literal of an entity), the type of entities to consider (classes, properties, or individuals), and how strict the match should be. In addition, developers' background knowledge helps in the filtering.

[8] http://sindice.com/

[9] http://sig.ma/

[10] http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary

[11] http://www.obofoundry.org/

[12] http://cupboard.open.ac.uk

[13] http://www.loa-cnr.it/ontologies/DOLCE-Lite.owl

[14] http://www.ontologyportal.org/translations/SUMO.owl.txt

In addition to its user interface, Watson includes a set of open APIs making it possible for application developers to find and exploit online ontologies directly from the provided infrastructure. This API has been used to create an interface to Watson from the NeOn Toolkit, where definitions of specific classes and properties can be found and reuse: the Watson plugin (d'Aquin et al. 2008). Using the Watson plugin, an initial "skeleton" model can be defined as a basis for searching relevant definitions from online ontologies. Selecting a concept or a property, the user can obtain list of statements that corresponds to alternative representations of this class and properties, and directly integrate such representations (partially or completely) in the ontology under development.

In addition to the Watson plugin, other developments have been integrated with Watson with the goal of facilitating ontology search and reuse. For example, in addition to extracted information, Watson provides a simple visual summary of each ontology using the key concept extraction mechanism described in (Peroni et al. 2008). Mechanisms such as visual summaries and the Watson indexing process were also reused to create Cupboard (d'Aquin and Lewen 2009), an ontology repository system, where users can publish ontologies and search them in a way similar to Watson.

As an example, Table 7.5 presents some ontologies that define mereological relations.

Task 1.4 Performing a comparative study. The goal here is to compare the candidate ontologies obtained in Task 1.3 with the axioms and definitions identified in Task 1.2. This comparative study is represented in the form of a table to facilitate its use. In the table, each row represents the set of definitions (or axioms) identified in Task 1.2, and each column, the ontologies found in Task 1.3.

As an example, a comparative table of ontologies implementing mereologies is shown in Table 7.6. The symbol "X" means that the feature is represented in the ontology. In the example, the definitions of *underlap* and *disjoint*, and the weak supplementation principle are formalized in formal mereologies (see Annex), but they do not appear in any of the OWL ontologies that appear in the table.

Task 1.5 Determining the most appropriate ontology to be reused. The goal of this task is to determine which of the candidate ontologies identified in Task 1.3 is the most appropriate to be reused in the ontology being developed. To determine such an ontology, the analysis following Fig. 7.2 is carried out.

Table 7.5 Mereology implementations (Task 1.3)

Found mereology implementations	*Project or institution*
Single part whole[a]	*W3C*
SUMO-OWL	*IEEE Standard Upper Ontology working group*
Dolce-Lite	*Italian Research Council (CNR)*
Oswebsite[b]	*OS Open data*
OBO	*Open Biological and Biomedical Ontologies*[c]

[a] http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/part.owl

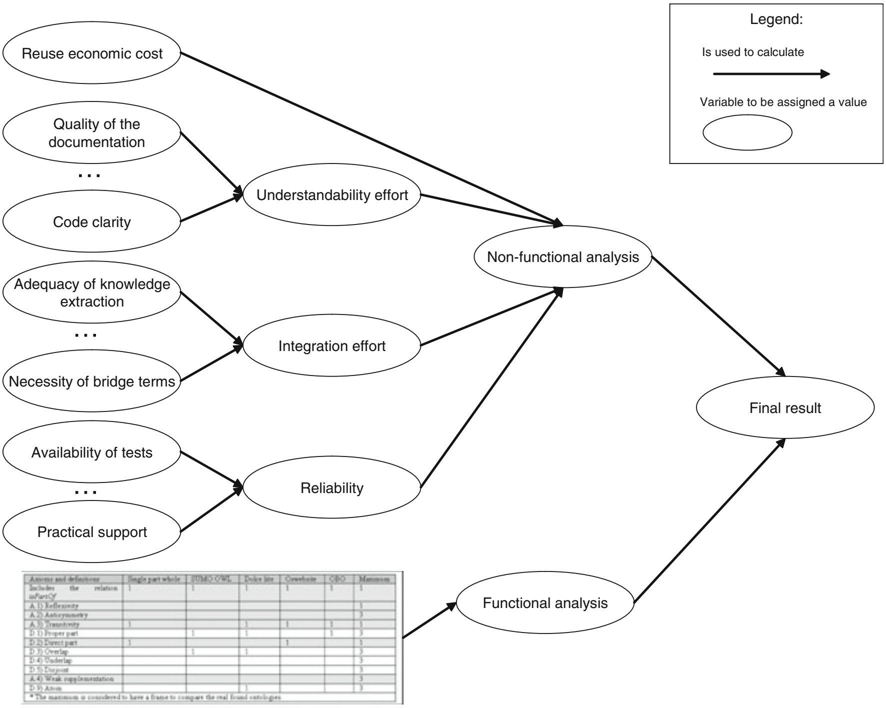
[b] http://www.ordnancesurvey.co.uk/oswebsite/ontology/Mereology.owl

[c] http://www.berkeleybop.org/ontologies/obo-all/relationship/relationship.owl

Table 7.6 Comparative study of mereology ontologies (Task 1.4)

Axioms and definitions	Single part whole	SUMO-OWL	Dolce-Lite	Oswebsite	OBO
Includes the relation *isPartOf*	X	X	X	X	X
A.1. Reflexivity					
A.2. Antisymmetry					
A.3. Transitivity	X		X	X	X
D.1. Proper part		X	X		X
D.2. Direct part	X			X	
D.3. Overlap		X	X		
D.4. Underlap					
D.5. Disjoint					
A.4. Weak supplementation					

The shadow features are required by the host ontology of section 6 use case (Task 1.2)



Axioms and definitions	Single part whole	SUMO-OWL	Dolce lite	Oswebsite	OBO	Maximum
Includes the relation *isPartOf*	1	1	1	1	1	1
A.1) Reflexivity						1
A.2) Antisymmetry						1
A.3) Transitivity	1		1	1	1	1
D.1) Proper part		1	1		1	1
D.2) Direct part	1			1		1
D.3) Overlap		1	1			1
D.4) Underlap						1
D.5) Disjoint						1
A.4) Weak supplementation						1
D.9) Atom			1			1

* The maximum is considered to have a frame to compare the real found ontologies

Fig. 7.2 How to take the decision of choosing an ontology (Task 1.5)

Features identified in Task 1.2 (reflexivity, transitivity, etc.) are called functional features, while the tag non-functional is used for the rest of features (reuse economic cost, code clarity, etc.). A weight of 0.75 has been assigned to non-functional features and 0.25 to the functional ones. These weights have been assigned because, according to the author's experience, adding new functional features to an ontology that scores well with respect to the non-functional ones is, in most cases, easier than overcoming the lack of compliance in non-functional properties. The exact value of each weight can be obtained using different procedures. One of them is by means of the utility theory (Jiménez et al. 2003). Another one is by means of former experience; this is the option we have used. That is, we have adjusted the weights so that the quantitative result applied to different cases of generic ontology reuse is equal to the one recommended by experienced people in this task. The first option follows a prescriptive approach, whereas the second, a descriptive one.

With the objective of having a reference to compare the scores, the score of an ideal ontology has been considered as a normalization denominator. Let us note that if this ideal reference is not provided, it is not easy to know the significance of the difference between the ontology scores. Thus, for example, without this ideal reference, if the difference between ontologies o_1 and o_2 is 0.4, the engineer cannot necessarily determine how large such difference is.

Given an ontology *ont*, the following formula to calculate the score of the functional features analysis is used:

$$ScoreFunctionalFeatures_{\mathrm{ont}} = \frac{\sum_i \mathrm{value}_{\mathrm{ont}}(\mathrm{functionalFeature}_i)}{\sum_i \mathrm{value}_{\mathrm{idealOnt}}(\mathrm{functionalFeature}_i)} \times 100\% \quad (7.1)$$

where $\texttt{value}_{\mathrm{ont}}(\texttt{functionalFeature}_i)$ is the value of functional feature *i* for the ontology *ont*, and $\texttt{value}_{\mathrm{idealOnt}}(\texttt{functionalFeature}_i)$ is the value of functional feature *i* for an ideal ontology, that is, the number of features (axioms and definitions) obtained in Task 1.4.

Concerning non-functional features analysis, it is carried out on the basis of the following four dimensions:

- *Reuse economic cost*. It refers to the estimate of the economic cost needed for accessing and using the candidate ontology. If the candidate ontology has any type of license, then the cost of acquisition and/or exploitation should be taken into account (Gómez-Pérez and Lozano-Tello 2005).
- *Understandability effort*. It refers to the estimate of the effort needed for understanding the candidate ontology. In this case, the following criteria should be analyzed:
  - *Quality of the documentation*. It refers to whether there is any communicable material used to describe or explain different aspects of the candidate ontology (e.g., modeling decisions). The documentation should explain the statements contained in the ontology so that a nonexpert could understand them (Pinto and Martins 2001).
  - *Availability of external knowledge sources*. It refers to whether the candidate ontology has references to documentation sources and/or if experts are easily available.
  - *Code clarity*. It refers to whether the code is easy to understand and modify, that is, if the knowledge entities follow unified patterns and are intuitive

(Pinto and Martins 2001). It is advantageous to use the same pattern to make sibling definitions, thus improving ontology understanding and making it easier to include new definitions (Gómez-Pérez and Rojas 1999). For example, if it has been decided to distinguish between Frenadol product and Frenadol substance, the same distinction should be made for the rest of drugs (Efferalgan, Dolostop, etc.). Clarity also refers to whether the code is well documented, that is, if it includes clear and coherent definitions and comments for the knowledge entities represented in the candidate ontology. The difference between this criterion and the *quality of the documentation* is that *clarity* refers to the comments and the definitions inside the code; meanwhile, the *quality of the documentation* refers to external documentation (papers, manuals, etc.).

- *Integration effort*. It refers to the estimate of the effort needed for integrating the candidate ontology into the ontology being developed. In this case, the following criteria should be analyzed:

  – *Adequacy of knowledge extraction*. It refers to whether it is easy to identify parts of the candidate ontology to be reused and to extract them. For example, in large and not modularized ontologies (e.g., SUO), the difficulty to extract the part of the knowledge we are interested in is especially high.
  – *Adequacy of naming conventions*. It refers to whether both ontologies (the candidate and the one being developed) follow the same rules for naming the different ontology components (e.g., concept names should start with capital letters, relation names should start with non-capital letters).
  – *Adequacy of the implementation language*. It refers to whether both languages (the candidate ontology's and the ontology's being developed) are the same, or at least are able to represent similar knowledge with the same granularity.
  – *Knowledge clash*. It refers to whether there are contradictory bits of knowledge between the candidate ontology to be reused and the ontology being developed (e.g., discrete time versus continuous time assumption).
  – *Adaptation to the reasoner*. It refers to whether the adaptation of definitions and axioms that satisfy the existing restrictions of the reasoner is needed (e.g., explicit definitions can be included in OWL ontologies; however, this kind of definitions cannot be included in ontologies written in Prolog).
  – *Necessity of bridge terms*. It refers to whether it is necessary to create new linking axioms and/or relations to integrate the candidate ontology to be reused into the ontology being developed.

- *Reliability*. It refers to an analysis of whether ontology developers can trust the candidate ontology to be reused. In this case, the following criteria should be considered:

  – *Design criteria*. It refers to whether the ontology has been built according to the design criteria assumed by the development team of the domain ontology.

  For example, one design criterion is the use of standards of the domain (on pharmacy, medicine, etc.) if they exist.
- *Availability of tests*. It refers to whether tests are available for the candidate ontology to be reused. Although it is not still usual in ontological engineering, the development team could publish the tests used during the ontology construction.
- *Former evaluation*. It refers to whether the ontology has been properly evaluated, not only by means of automatic unit tests but also by domain and ontology modeling experts.
- *Theoretical support*. It refers to whether the candidate ontology is supported by a sound theory, explicitly described in a document.
- *Development team reputation*. It refers to whether the development team of the candidate ontology is known to be experienced and competent.
- *Purpose reliability*. It refers to whether the candidate ontology has been developed as a simple example or for a stronger purpose.
- *Popularity*. It refers to whether there are well-known projects or ontologies reusing the candidate ontology (Lozano-Tello 2002).

Table 7.7 shows the criteria (organized by dimensions) and the ways to measure them. In the table, for each criterion, there is (a) a range of values (an interval of linguistic values or a natural number), (b) an explanation of how to measure the criterion, and (c) a numerical weight. The numerical weights are proposed here by default, according to the importance the authors give to the different criteria; for example, the criterion *design criteria* is extremely important for us, and therefore we assign a numerical weight of 10; however, the criterion *purpose reliability* is not so important for us, therefore, we give it a weight of 3. It is worth mentioning that such numerical weights depend on the importance the ontology developer gives to the different criteria, and that such weights can be modified. The symbols (+) and (−) in the weights are specified to indicate whether the criterion counts in a positive or a negative way, respectively. These symbols cannot be modified by the ontology developer.

Thus, ontology developers should fill a table and analyze the candidate ontologies with respect to the abovementioned criteria, taking into account the different ways to measure each criterion and the possible values that can be assigned.

Having filled Table 7.7 with different values for each criterion and for each candidate ontology, ontology developers should obtain a score for each candidate ontology and then decide which one is the most appropriate. To obtain such a score, the following method is proposed:

- To transform linguistic values, the following transformation rules are proposed:
  - Value = Unknown → $\text{Value}_\text{T} = 0$.
  - Value = Low → $\text{Value}_\text{T} = 1$ if the weigh is (+), 3 otherwise.
  - Value = Medium → $\text{Value}_\text{T} = 2$.
  - Value = High → $\text{Value}_\text{T} = 3$ if the weigh is (+), 1 otherwise.

Table 7.7 Decision criteria to select an ontology (Task 1.5)

Criteria	Range of values	How to measure it	Weight
Reuse cost			
Reuse economic cost	{Unknown, low, medium, high}	Asking the owner for an estimate	(−) 10
Understandability effort			
Quality of the documentation	{Unknown, low, medium, high}	Analyzing if the ontology has documentation and if such documentation really explains the ontology itself, as well as modeling criteria used during the ontology development	(+) 8
Availability of external knowledge	{Unknown, low, medium, high}	Analyzing if in the ontology documentation there is any reference to external sources that could be used to better understand the ontology	(+) 7
Code clarity	{Unknown, low, medium, high}	Inspecting the ontology code analyzing the complexity of the definitions (and axioms) implemented in the ontology	(+) 8
Integration effort			
Adequacy of knowledge extraction	{Unknown, low, medium, high}	Analyzing if the ontology is modularized or if it can be modularized in an easier way	(+) 9
Adequacy of naming conventions	{Unknown, Low, Medium, High}	Comparing the naming conventions of both ontologies	(+) 5
Adequacy of the implementation language	{Unknown, low, medium, high}	Comparing the ontology language of both ontologies. If both languages are different, analyzing the loss of knowledge in the translation	(+) 7
Knowledge clash	{Unknown, low, medium, high}	Comparing modeling decisions of both ontologies	(−) 7
Adaptation to the reasoner	{Unknown, low, medium, high}	Comparing the reasoners related to the ontology language of both ontologies	(+) 7
Necessity of bridge terms	{Unknown, low, medium, high}	Inspecting the ontology code and the result of Task 1.1 (see Table 7.3)	(−) 6
Reliability			
Design criteria	{Unknown, low, medium, high}	Analyzing if the ontology is built according to the design criteria assumed by the development team of the domain ontology	(+) 10
Availability of tests	{Unknown, low, medium, high}	Analyzing if the ontology documentation refers to existing unit tests	(+) 8
Former evaluation	{Unknown, low, medium, high}	Analyzing if the ontology documentation refers to different types of evaluation (automatic unit tests, human evaluation, etc.)	(+) 8

(continued)

Table 7.7 (continued)

Criteria	Range of values	How to measure it	Weight
Theoretical support	{Unknown, low, medium, high}	Analyzing if the ontology documentation refers to the theory on which the ontology is based	(+) 10
Development team reputation	{Unknown, low, medium, high}	Searching for information about the ontology development team (other ontologies developed, papers published, etc.)	(+) 8
Purpose reliability	{Unknown, low, medium, high}	Analyzing if the ontology documentation refers to the purpose for which the ontology was developed	(+) 3
Popularity	{Unknown, low, medium, high}	Analyzing if the ontology documentation refers to other ontologies and/or projects reusing the ontology	(+) 7

- where:
- $\mathrm{Value}_{\mathrm{T}}$ is the transformed value
- Value is the linguistic value provided by the ontology developer

Given that we want to penalize ontologies about which we have less knowledge, we have assigned a value of 0 to *unknown*.

- The score that synthesizes the non-functional features contribution is the following weighted mean:

$$\mathit{ScoreNon\text{-}FunctionalFeatures}_{\mathrm{ont}} = \sum_i \mathrm{Value}_{\mathrm{Tont},i} \times \frac{\mathrm{Weight}_j}{\sum_i \mathrm{Weight}_i} \times 100\% \quad (7.2)$$

where:

- $\mathit{ScoreNon\text{-}FunctionalFeatures}_{\mathrm{ont}}$ is the score for the candidate ontology *ont* for the set of criteria
- j is a particular criterion of those included in Table 7.7
- $\mathrm{Value}_{\mathrm{Tont},j}$ is the transformed value for the criterion j in the ontology *ont*
- Weight_k is the numerical weight associated to the criterion k
- Finally, applying the aforementioned weights of 0.25 and 0.75 for functional and non-functional features respectively, the following formula is applied:

$$\begin{aligned}\mathit{Score} = 0.25 \times \mathit{ScoreFunctionalFeatures} + 0.75\\ \times \mathit{ScoreNon\text{-}FunctionalFeatures}.\end{aligned} \quad (7.3)$$

After applying the previous formula to all the candidate ontologies, ontology developers should select the candidate ontology with the best normalized scored.

Table 7.8 Determining the most appropriate mereology implementation (Task 1.5)

Criteria	Weight		Values				
			Single part whole	SUMO-OWL	Dolce-Lite	Oswebsite	OBO
Reuse cost							
Reuse economic cost	(−)	10	Low	Low	Low	Low	Low
Understandability effort							
Quality of the documentation	(+)	8	High	High	High	Unknown	Unknown
Availability of external knowledge	(+)	7	High	High	High	Unknown	Unknown
Code clarity	(+)	8	High	High	High	High	High
Integration effort							
Adequacy of knowledge extraction	(+)	9	High	High	Low	Low	Low
Adequacy of naming conventions	(+)	5	Low	High	Low	High	Low
Adequacy of the implementation language	(+)	7	High	High	High	High	High
Knowledge clash	(−)	7	Low	Low	Low	Low	Low
Adaptation to the reasoner	(+)	7	High	High	High	High	High
Necessity of bridge terms	(−)	6	Low	Low	Low	Low	Low
Reliability							
Design decisions	(+)	10	High	High	High	High	High
Availability of tests	(+)	8	Unknown	Unknown	Unknown	Unknown	Unknown
Former evaluation	(+)	8	Unknown	Unknown	Unknown	Unknown	Unknown
Theoretical support	(+)	10	High	High	High	Unknown	Unknown
Development team reputation	(+)	8	High	High	High	High	High
Purpose reliability	(+)	3	Low	Unknown	Unknown	High	High
Popularity	(+)	7	Unknown	Unknown	Unknown	Unknown	Unknown

As an example, in the context of the PPO case, we have filled in the values associated with the OWL versions of SUO-OWL and Dolce-Lite, which are shown in Table 7.8. The scores of the functional features have been obtained from Table 7.6.

The results in Task 1.5 have been very close. Given that we have found top-level concepts of SUMO-OWL like *biologically active substance* and *molecule* (and their ancestors) useful for PPO, the criterion *adequacy of knowledge extraction* has been assigned *high* for this ontology and, consequently, has obtained the best score (Table 7.9).

For this example, we have used a spreadsheet. For the future, we plan to support the automation of this task in NeOn Toolkit.

Table 7.9 Synthesis of the results of determining the most appropriate mereology implementation (Task 1.5)

	Single part whole	SUMO-OWL	Dolce-Lite	Oswebsite	OBO
Score for non-functional features. See formula (7.2). Henceforth, this result will be referred as (**3**)	85.33%	87.2%	79.73%	64.8%	62.13%
Score for functional features resulting from task. See Table 7.6 and formula (7.1). Henceforth, this result will be referred as (**4**)	33.33%	33.33%	50%	33.33%	33.33%
Final score = **0.75** × (**3**) + **0.25** × (**4**)	72.33%	**73.73%**	72.3%	56.93%	54.93%

7.2.2 *Activity 2: Customizing the Selected Generic Ontology*

The goal of this activity is to customize the ontology selected in Activity 1 according to the needs of the domain ontology being developed. This activity consists of the following tasks:

Task 2.1 Pruning the ontology to be reused according to the needed features. The goal of this task is to prune the selected ontology taking into account the features needed in the domain ontology that is being developed. Thus, for example, if the definition of overlap is defined in the generic ontology, but it is not necessary in the resulting ontology, it should be removed.

Task 2.2 Enriching the ontology to be reused. The goal of this task is to extend the ontology selected with the new conceptual structures needed in the domain ontology being developed. In the PPO example, we have added transitivity to the *part* and *properPart* object properties, reflexivity and antisymmetry to *part*, and asymmetry and irreflexivity to *properPart*.

When pruning and enriching the ontology, it is necessary to take into account that the axioms and definitions to be reused may be applicable to a category that does not completely include all the individuals of interest in our domain ontology. If this happens, an adaptation of the axioms and definitions should be performed.

Task 2.3 Translating the ontology to be reused into the implementation language of the domain ontology being developed. The goal of this task is to translate the selected ontology into the implementation language of the domain ontology being developed if those two ontologies are in different languages.

An ontology can be translated in an automatic or manual way. It is important to point out that a complete translation into different languages is not always possible. For example, let us suppose the following implementation in Prolog of `overlaps` and `disjoint`:

```
overlaps(X, Y) :- isPartOf(Z, X), isPartOf(Z, Y).
disjoint(X, Y) :- \+overlaps(X, Y).
```

The rule corresponding to disjoint cannot be implemented in OWL. In fact, let us note that given that Prolog works under the closed world assumption, if common parts of `substance1` and `substance2` have not been represented, the answer to the query:

```
?:- disjoint(substance1, substance2).
```

will be `true`. However, it is not possible to attain this effect directly with OWL (open world assumption).

Task 2.4 Adapting the ontology to be reused to the design criteria followed in the ontology to be developed. The following modifications have to be done in most cases: (a) changing names (concepts, properties) to adapt them to the naming conventions used in the ontology network being developed and (b) adding range to properties. For example, we have adapted the names to the convention used in PPO. Thus, *part* has been changed to *isPartOf*.

Task 2.5 Evaluating the obtained ontology. The goal of this task is to evaluate from a content perspective if there are no errors in the ontology. This task is described in detail in Chap. 9.

7.2.3 Activity 3: Integrating the Generic Ontology to be Reused in the Ontology Being Developed

The goal of this activity is to integrate the ontology obtained in Activity 2 in the ontology being developed. The development team should decide whether:

- To import the customized ontology. The advantage is that the resulting developed ontology will be structured in different modules[15] (see Chap. 10).
- To copy the customized ontology. This can be a good solution if the customized ontology belongs to the same domain as the one of the ontology to be developed. For example, if the customized ontology adds more drug types to a drug ontology.

In any case, links between terms of the reused ontology and the ontology to be developed should be established. In the case of PPO, we have taken advantage of the possibilities that SUMO-OWL offers us to easily represent different perspectives of the notion of drug, for example, drug as a substance that acts in our organism and drug as a product that can be sold. Moreover, given that transitivity, antisymmetry, etc. involve individuals, we have added an individual for each type of substance and product. Therefore, the application that uses the PPO maintains the individuals corresponding to particular entities (e.g., Frenadol C243, corresponding to Frenadol box with manufacturing lot C243) and the individuals that represent products and substances in a general way. Thus, for example, the

[15] The term *module* has here the pragmatic sense equivalent to the d'Aquin's reference cited in the Introduction.

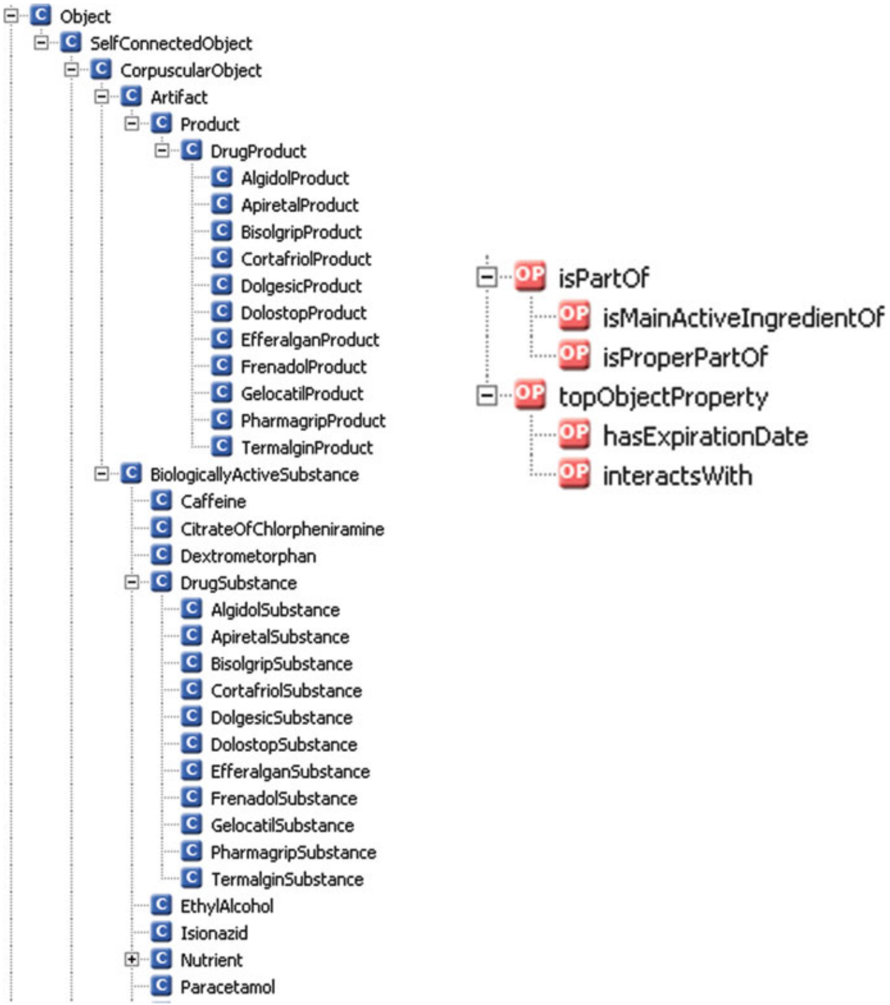


Fig. 7.3 Partial view of the concept and the object property hierarchies (Snapshot taken from NeOn Toolkit)

system can infer that *caffeine* is part of *Frenadol* because there is an individual of *caffeine* (also with tag "*caffeine*") that is part of an object *Frenadol*, that is, an individual of *Frenadol*. We have also added the axioms identified in Table 7.3 (see Sect. 7.2.1) (e.g., *isMainActiveIngredient* is subrelation of *isPartOf*).

To answer CQ_4, we have added this rule to the ontology:

```
interactsWith(?x, ?y), isPartOf(?x, ?z) ->
       interactsWith(?z, ?y)
```

Table 7.10 Formal host competency questions that require *part of* modeling (for the sake of simplicity, prefixes, and value data types are omitted in the answers)

<table>
<tr><th>Informal CQ</th><th>Formal CQ</th><th>Example of answer</th></tr>
<tr><td>What drugs do have paracetamol?</td><td># CQ1
SELECT ?X
WHERE
{
?X rdf:type ub:DrugSubstance .
ub:Paracetamol ub:isProperPartOf
?X .
}</td><td>---------------------
|X|
=====================
|FrenadolSubstance
|BisolgripSubstance
|CortafriolSubstance
|DolgesicSubstance
|TermalginSubstance
|AlgidolSubstance
|EfferalganSubstance
|DolostopSubstance
|GelocatilSubstance
|ApiretalSubstance
|PharmagripSubstance
---------------------</td></tr>
<tr><td>Which is the composition of Frenadol®?</td><td># CQ2
SELECT ?X
WHERE
{
 ?X ub:isProperPartOf
    ub:FrenadolSubstance .
}</td><td>---------------------
|X|
=====================
|Dextrometorphan
|CitrateOfChlorpheniramine
|Caffeine
|Paracetamol
---------------------</td></tr>
</table>

(continued)

Table 7.10 (continued)

Informal CQ	Formal CQ	Example of answer

Which is the main active ingredient of Frenadol®?	`# CQ3`
	
	`SELECT ?X`
	`WHERE`
	`{`
	` ?X ub:isMainActiveIngredientOf`
	`              ub:FrenadolSubstance`
	`.`
	`}`
		`---------------`
		`| X           |`
		`===============`
		`| Paracetamol |`
		`---------------`

Which substances do Frenadol® interacts with?	*`# CQ4`*
	
	`SELECT ?X`
	`WHERE`
	`{`
	` ub:FrenadolSubstance`
	`ub:interactsWith`
	`    ?X .`
	`}`
		`----------------`
		`| X            |`
		`================`
		`| Rifanpicin   |`
		`| Propranolol  |`
		`|  Isionazid   |`
		`| EthylAlcohol |`
		`----------------`

That is, if a substance ?*x* interacts with another substance ?*y*, then the latter interacts with every part of ?*x*. Thus, for example, given that paracetamol interacts with the ethyl alcohol, Frenadol® also interacts with ethyl alcohol.

A partial view of the resulting ontology is shown in Fig. 7.3.

The resulting ontology should be evaluated. In the PPO case, besides other tests, we have checked that the CQs are answered (see Table 7.10).

7.3 Conclusions and Future Work

The reuse of (well-developed) ontologies allows spreading good practices and increasing the overall quality of ontological models. In this chapter, we have presented how to carry out this process. The guidelines shown here provide the methodological assistance to Scenario 3 in the NeOn Methodology (Chap. 2).

Given that the reuse of an ontology usually implies pruning it, ontology reuse usually implies statement reuse (see (Suárez-Figueroa 2010) to know more about how to reuse domain ontologies as well as ontology statements). Consequently, we have not distinguished between these two classes of reuse.

It is also worth mentioning that interesting knowledge represented in ontologies may be found by chance. For instance, part of the knowledge on substances and products reused from SUMO-OWL has been found when we were searching for mereology knowledge.

The NeOn Toolkit includes the Watson plugin to support ontology search. An objective for future development will be to develop the necessary plugin to assist with the other tasks associated with ontology reuse, especially for the selection of the most appropriate ontology (Task 1.5).

In addition, it would be interesting to perform a comparison of the costs of (a) reusing generic ontologies versus (b) developing what is required from scratch.

Annex: Mereology

A *mereology* is a formal theory of parts and associated concepts (Borst 1997; Schneider 2003). We have said "*a* mereology" instead of "*the* mereology" because different assumptions can be taken into account in the formalization of parthood. Therefore, different mereologies can be proposed.

In the following paragraphs, we will show one of the mereologies presented by Varzi (2007).

Theory *M*. Most of the authors agree on the following core of axioms (named with A) and definitions (named with D) (Varzi 2007). Along these paragraphs, we use examples of territories to clarify the meaning of axioms and definitions. The mentions to administrative units really refer to their physical territories.

- *A.1. Reflexivity*. Every object of the universe of discourse is a part of itself. For instance, the EU is part of the EU.
- *A.2. Antisymmetry*. If an object x is a part of y, and y is a part of x, then x and y are the same object. For instance, if the territory T_1 is part of the territory T_2, then the only way so that T_2 is part of T_1 is being T_1 and T_2 the same territory.
- *A.3. Transitivity*. If x is a part of y, and y is a part of z, then x is a part of z. For instance, the Community of Madrid is part of Spain, and Spain is part of the EU; therefore, the Community of Madrid is a part of the EU.

A number of additional mereological predicates can be then introduced by definition:

- *D.1. Proper part*. A proper part is a part that is other that the individual itself. For example, Spain is proper part of the EU, since Spain is part of the EU and they are different entities.
- *D.2. Direct part*. X is direct part of y if and only if x is proper part of y and there is no part between x and y[16]. For example, Spain is direct part of the EU, but Madrid is not, since Spain is a part between Madrid and the EU.

[16] http://hcs.science.uva.nl/projects/NewKACTUS/library/lib/mereology.html

- *D.3. Overlap*. The relation *overlaps* is defined as a sharing part. That is, *x* and *y* overlap if and only if there is a *z* such that *z* is part of *x* and part of *y*. For instance, Nordic countries and the EU overlap, since there are Nordic countries which are parts of the EU.
- *D.4. Underlap*. The relation *underlaps* is defined as a sharing whole. That is, *x* and *y* underlap if and only if there is a *z* such that *x* and *y* are parts of *z*. For example, the Netherlands, Sweden, and Spain underlap the same common whole: the EU.
- *D.5. Disjoint*. The *disjoint* relation is the logical negation of *overlaps*. For example, Belgium and the Netherlands are disjoint territories.

Theory *M* may be viewed as embodying the common core of any mereological theory. A.1–A.3 should be extended to build a mereology.

Minimal mereology (*MM*). A way to extend *M* is assuming the following principle (Varzi 2007):

- *A.4. Weak supplementation principle*. Every object *x* with a proper part *y* has another part *z* that is disjoint from *y*. The domain of territories, for example, fulfills this principle. For example, given that Spain is proper part of the EU, then the EU has other parts that are disjoint from Spain: the Netherlands, Luxemburg, Sweden, etc.

Most of the authors strengthen that A.4 should be incorporated to *M* as a further fundamental principle on the meaning of *part of*. Other authors provide scenarios that could be counterexamples of this principle. However, it is far from being demonstrated that such supposed counterexamples have implications in computer applications.

The rest of mereologies starting from MM are explained with examples in (Fernández López et al. 2008; Suárez-Figueroa 2010).

References

Borst WN (1997) Construction of engineering ontologies. Centre for Telematica and Information Technology, University of Tweenty, Enschede

d'Aquin M, Lewen H (2009) Cupboard – a place to expose your ontologies to applications and the community. Demo at European Semantic Web Conference, ESWC 2009, Heraklion, Greece

d'Aquin M, Motta E (2011) Watson, more than a semantic web search engine, Semant Web J 2, IOS Press

d'Aquin M, Sabou M, Dzbor M, Baldassarre C, Gridinoc L, Angeletou S, Motta E (2007a) Watson: a gateway for the semantic web. Poster session of the European Semantic Web Conference, ESWC 2007, Busan

d'Aquin M, Schlicht A, Stuckenschmidt H, Sabou M (2007b) Ontology modularization for knowledge selection: experiments and evaluations. In: 18th international conference on Database and Expert Systems Applications, DEXA 2007, Regensburg, Germany

d'Aquin M, Sabou M, Motta E (2008) Reusing knowledge from the semantic web with the Watson Plugin. Demo at International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany

Fernández López M, Gómez-Pérez A, Suárez-Figueroa MC (2008) Selecting and customizing a mereology ontology for its reuse in a pharmaceutical product ontology. In: Grüninger M, Eschenbach C (eds) Formal Ontology in Information Systems. Fifth international conference (FOIS-2008), Saarbrücken, Germany. IOS Press, Amsterdam, pp 181–194

Golbeck J, Fragoso G, Hartel F, Hendler J, Parsia B, Oberthaler J (2003) The national cancer institute's thesaurus and ontology. J Web Semant1(1):75–80

Gómez-Pérez A, Lozano-Tello A (2005) Applying ONTOMETRIC method to measure the suitability of ontologies. In: Green P, Rosemann M (eds) Business systems analysis with ontologies. Idea Group Publishing, Hershey, pp 249–269

Gómez-Pérez A, Rojas MD (1999) Ontological reengineering and reuse. In: Fensel D, Studer R (eds) 11th European workshop on Knowledge Acquisition, Modeling and Management (EKAW 1999), Dagstuhl Castle, Germany, Lecture Notes in Artificial Intelligence LNAI 1621 Springer, Berlin, Germany, pp 139–156

Jiménez A, Ríos-Insua S, Mateos A (2003) A decision support system for multiattribute utility evaluation based on imprecise assignments. Decis Support Syst 36:65–79

Lozano-Tello A (2002) Métrica de idoneidad de ontologías. PhD Thesis, Universidad de Extremadura, Cáceres, Spain 2002

Peroni S, Motta E, d'Aquin M (2008) Identifying key concepts in an ontology through the integration of cognitive principles with statistical and topological measures. In: Third Asian semantic web conference, Bangkok, Thailand

Pinto HS, Martins JP (2001) A methodology for ontology integration. In: Gil Y, Musen M, Shavlik J (eds) First international conference on Knowledge Capture (KCAP 2001), Victoria, Canada. ACM Press, New York, pp 131–138

Schneider L (2003) How to build a foundational ontology: the object-centered high-level reference ontology OCHRE. In: Günter A, Kruse R, Neumann B (eds) Proceedings of the 26th annual German conference on Artificial Intelligence (KI-2003), Hamburg, Germany. Lecture Notes in Artificial Intelligence (LNAI-2821), Berlin, Germany, pp 120–134. (http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=12B7EC62C9601245457C735C07AA07A0?doi=10.1.1.1.3440&rep=rep1&type=pdf)

Suárez-Figueroa, MC (coordinator) (2008) D5.4.1. NeOn Methodology for building contextualized ontology networks. NeOn project

Suárez-Figueroa, MC (2010) NeOn Methodology for building ontology networks: specification, scheduling and reuse. PhD Thesis, Facultad de Informática, Universidad Politécnica de Madrid, Spain

van Heijst G, Schreiber ATh, Wielinga BJ (1997) Using explicit ontologies in KBS development. Int J Hum-Comput Stud 45:183–292

Varzi A (2007) In: Aiello M, Pratt-Hartmann I, van Benthem J (eds) Spatial reasoning and ontology: parts, wholes, and locations. Springer, Heidelberg, pp 945–1038

Chapter 8
Ontology Localization

Mauricio Espinoza Mejía, Elena Montiel-Ponsoda, Guadalupe Aguado de Cea, and Asunción Gómez-Pérez

Abstract In the context of the Semantic Web, resources on the net can be enriched by well-defined, machine-understandable metadata describing their associated conceptual meaning. These metadata consisting of natural language descriptions of concepts are the focus of the activity we describe in this chapter, namely, ontology localization. In the framework of the NeOn Methodology, ontology localization is defined as the activity of adapting an ontology to a particular language and culture. This adaptation mainly involves the translation of the natural language descriptions of the ontology from a source natural language to a target natural language, with the final objective of obtaining a multilingual ontology, that is, an ontology documented in several natural languages. The purpose of this chapter is to provide detailed and prescriptive methodological guidelines to support the performance of this activity.

8.1 Motivation

As with the World Wide Web, the success or failure of the Semantic Web will be determined to a large extent by easy access to and availability of high-quality and diverse content (Benjamins et al. 2002). In this respect, an important challenge that needs to be addressed is the multilingualism problem, which until now has not been properly investigated (Tjoa et al. 2005). This problem already exists in the current

M. Espinoza Mejía (✉)
Facultad de Ingeniería, Universidad de Cuenca. Cdla. Universitaria, Av. 12 de Abril sn, Cuenca, Ecuador
e-mail: mauricio.espinoza@ucuenca.edu.ec

E. Montiel-Ponsoda • G. Aguado de Cea • A. Gómez-Pérez
Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo sn, 28660 Boadilla del Monte, Madrid, Spain
e-mail: emontiel@fi.upm.es; lupe@fi.upm.es; asun@fi.upm.es

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_8, © Springer-Verlag Berlin Heidelberg 2012

web and should also be tackled in the Semantic Web. Studies on language distribution over WWW content show that even if English is the predominating language for documents, there exists an important amount of resources written in other languages, according to the following distribution: English 27.6%, Chinese 22.1%, Spanish 7.9%, Japanese 5.5%, French 4.6%, Portuguese 4.2%, German 3.7%, Arabic 2.9%, Russian 2.6%, Korean 2.2%, and other languages 16.7%[1]. In the case of the Semantic Web, the problem is similar; most of the ontologies that have been built so far have English as their basis. Nevertheless, although English is now the de facto language for science and technology, other spoken languages are used, and it is important to provide methods and tools both to support the definition of ontologies expressed in languages other than English and also to support interoperability across ontologies written in different languages.

Currently, a great effort is being applied to the construction of ontologies. Although access to top-quality ontologies (e.g., Galen[2], CYC[3], or AKT[4]) is in many cases free and unlimited for users around the world, most of these ontologies can be said to be essentially monolingual, that is, documented in one natural language, and this language is often English as an international lingua franca. However, there is a growing need for multilingual ontology resources that overcome communication barriers arising from cultural-linguistic differences, lack of excellent command of English, need for high precision in communication, etc. In fact, multilingual knowledge is even more prevalent in those countries that have more than one official language (Yang and Li 2003). For example, Chinese and English are the official languages in Hong Kong; French and English in Canada; and Dutch, French, and German in Belgium.

Moreover, the use of ontologies has grown not only in terms of the number of application domains but also in the number of natural languages chosen to build domain-specific knowledge bases. Thus, multilingual ontologies are nowadays demanded by institutions worldwide with a huge number of resources available in different languages. Basically, usage of multilingual ontologies traverses many disciplines and has become an urgent need in certain organizations. For instance, in agriculture, the Food and Agriculture Organization (FAO) has expressed the need for semantically structuring the information they have in different natural languages. Since all FAO official documents must be made available in Arabic, English, Chinese, French, Russian, and Spanish, a large amount of research has been carried out in translating large multilingual agricultural thesauri (Chun and Wenlin 2002), in mapping methodologies for thesauri (Liang et al. 2005; Liang and Sini 2006), and in defining requirements to improve the interoperability of these multilingual information resources (Caracciolo et al. 2007). In education, the

[1] Obtained on September 30, 2009, from http://www.internetworldstats.com

[2] http://www.co-ode.org/galen/

[3] http://www.opencyc.org/downloads

[4] http://www.aktors.org/publications/ontology/

Bologna declaration has introduced an ontology-based framework for qualification recognition (Vas 2007) across the European Union, in an effort to best match labor markets with employment opportunities. In e-learning, educational ontologies are used to enhance learning experience (Cui et al. 2004) and to empower system platforms with high adaptivity (Sosnovsky and Gavrilova 2006). In the finance domain, ontologies are used to model knowledge in the stock market domain (Alonso et al. 2005) and portfolio management (Zhang et al. 2002). In medicine, ontologies are employed to improve knowledge sharing and knowledge reuse. For example, a notable amount of research has focused on the creation of an ontology of traditional Chinese medicine.

A further factor that has increased the need for multilingual ontologies is the development of some ontology-based systems that need to interact with information in natural languages. Some examples of these applications are cross-lingual information retrieval (Guyot et al. 2005), multilingual question answering (Pazienza et al. 2005), and knowledge management (Segev and Gal 2008).

These examples can serve to highlight the importance of adding multilingualism, that is, multilingual information, to ontologies before trying to solve the numerous pending problems that still exist with the current monolingual approach. But, while there is a clear need for ontology localization, there are no well-defined and broadly accepted definitions of what the ontology localization activity entails. Moreover, to our knowledge, no other study has focused on the methodological guidelines for this activity. For this reason, in this chapter we present efficient, prescriptive, and detailed methodological guidelines for the ontology localization activity.

The rest of the chapter is organized as follows: Sect. 8.2 presents the scope of the methodological guidelines together with a detailed and systematic account of the most important items from practical and application perspectives. Section 8.3 describes the guidelines for the ontology localization activity (including the filling card and the activity workflow). Section 8.4 includes an example of how the proposed guidelines for ontology localization are used in practice and the results obtained. Finally, Sect. 8.5 summarizes the main conclusions from this work.

8.2 Scope of the Methodological Guidelines

Methodological guidelines for ontology localization can be discussed from different perspectives:

- Guidelines for the development of internationalized ontologies
- Guidelines for the localization of existing ontologies
- Guidelines for reaching a mature ontology localization process

In what follows, we briefly describe these types of guidelines with respect to three aspects: (1) the objective and scope of the guidelines, (2) the target audience, and (3) the work related to them. Then, we will deal with the design principles we considered in order to define the guidelines used in this chapter.

8.2.1 Guidelines for the Development of Internationalized Ontologies

Objective and scope: The aim of these guidelines is to improve the design and implementation of internationalized ontologies in order to reduce the cost of localization. Based on the software localization field (LISA), we understand internationalized ontologies as ontologies built with the aim of supporting multilingual descriptions of the conceptualizations they provide, since they are to be used in a multilingual scenario. However, this means that in their design phase, language- and culture-specific concepts are to be stored externally or removed, so as to enable possible reuse, and only those common concepts are captured in the ontology.

Target audience: These guidelines are intended particularly for ontology developers, who are concerned with the design and development of ontologies. In addition, they are intended for international institutions interested in planning, designing, and implementing internationalized domain ontologies.

Related work: These guidelines provide recommendations for naming ontology elements. The key goal here is that the ontology elements be clear (avoid ambiguity) and simple (easy to translate to other languages). Recently, some approaches have been defined (Flied et al. 2007; Schober et al. 2007), which propose naming conventions for ontology terms.

8.2.2 Guidelines for the Localization of Existing Ontologies

Objective and scope: The aim is to carry out the localization process of ontologies already conceptualized. Usually, these ontologies are designed without taking into account the multilingual and localization aspects. Therefore, these guidelines aim at reducing costs, improving its quality, and increasing the consistency of the localization activity.

Target audience: These guidelines are particularly intended for ontology stakeholders such as localization managers, translators, and reviewers, who are concerned with the ontology localization activity. In addition, they are intended for communities interested in localizing ontologies and those international firms that may promote multilingualism in their working environments for a variety of reasons.

Related work: These guidelines describe different stages for the localization activity. At each stage of the process, they explain the activities or tasks to be performed with the same style and granularity level as used by software development methodologies. To the best of our knowledge, no guidelines exist for supporting the ontology localization activity. However, software localization methodologies could be adapted for ontology localization, as these methodologies are very general.

8.2.3 Guidelines for Realizing a Robust Localization Process

Objective and scope: These guidelines are meant to help reach a robust localization process within an organization. Usually, most organizations pass through different stages of maturity before reaching a robust localization process. Therefore, these guidelines describe behaviors or best practices adopted by successful projects.

Target audience: We envisage that they can be intended for organizations dedicated to the development of Semantic Web applications that require the use of multilingual ontologies.

Related work: These guidelines should describe different maturity levels used to improve and appraise the ability of an organization to perform the functions required in the ontology localization activity. The localization maturity model (DePalma 2007) (LMM) is a new advance in the deployment of software localization. While LMM compliance will not guarantee success, it does increase the likelihood of succeeding by helping planners understand what others have experienced and learned before them. As this methodology is quite general, we believe it can be adapted to ontology engineering methodologies.

In this chapter, we propose general guidelines that cover the localization of existing ontologies (second group above). In the following sections, we will define the actors involved in the different tasks of the ontology localization activity. Then, we will describe in detail the tasks for carrying out this activity.

8.3 Methodological Guidelines for Ontology Localization

In this section, our purpose is to explain the guidelines set out to help ontology developers in the ontology localization activity. The principles that guide the construction of such guidelines are the following:

- The guidelines should be general enough in the sense that they should help software developers and ontology practitioners to localize ontologies in different natural languages and domains.
- The guidelines should define each activity or task precisely; they should clearly state its purpose, its inputs and outputs, the actors involved, when its execution is more convenient, and the set of methods, techniques, and tools to be used for executing them.
- To facilitate a prompt assimilation of the ontology localization by software developers and ontology practitioners, we present the guidelines in a prescriptive way, not specifically oriented to researchers.

First, we present the different kinds of actors involved in the ontology localization activity. Then, we describe the guidelines for localizing ontologies to different natural languages.

8.3.1 Ontology Localization Actors

The different tasks involved in the ontology localization activity are carried out by different actors according to the kind of roles that must be performed in each task. In the following, we describe briefly the main actors involved in the localization activity:

- *Domain experts and ontology development team*. The domain expert or experts and the ontology development team (ODT) are responsible for performing one of the first tasks in the ontology localization activity. Their work consists of selecting the right resources and tools to perform the ontology localization activity.
- *Localization manager*. The localization manager plays a key role in the localization activity, as he or she must prepare all technical aspects of the localization activity, including the localization material (e.g., identifying the ontology elements to be localized) and distributing it to the localization team, setting up the localization team, as well as assigning and monitoring the tasks. Another task to be performed by the localization manager is the updating and final quality revision of the translated ontology.
- *Linguists*. These specialists can either be:
  - *Translators (localization specialist)*. Once the localization manager assigns the localization tasks to each member, the translator or localization specialist takes care of discovering the most appropriate translations for each ontology element.
  - *Reviewers (QA specialist)*. The reviewer or quality assurance (QA) specialist reviews the translated ontology elements. A reviewer does not necessarily focus on the quality of the translations but on the linguistic and stylistic quality of the translated ontology elements. The revision is a final language check for spelling errors, grammar mistakes, and consistency.

The current industry trend is to use external localization service providers in the translation task to avoid the high fixed cost of using in-house translators and use translators focused on the target markets and knowing the up-to-date usage of particular languages. We conceived a similar situation for ontology localization, in which translators and reviewers can be internal or external to the organization that develops the ontology and who work in a distributed environment. Figure 8.1 shows the high-level overview of the people who are directly involved in the ontology localization activity, both on the localization service and on the ontology publisher side. The localization manager and the ontology expert are responsible for the communication between both groups. In Fig. 8.1, the *quality assurance department (QA department)* performs a final quality check on all localized ontology elements received from the localization service provider to find out possible problems in the translations.

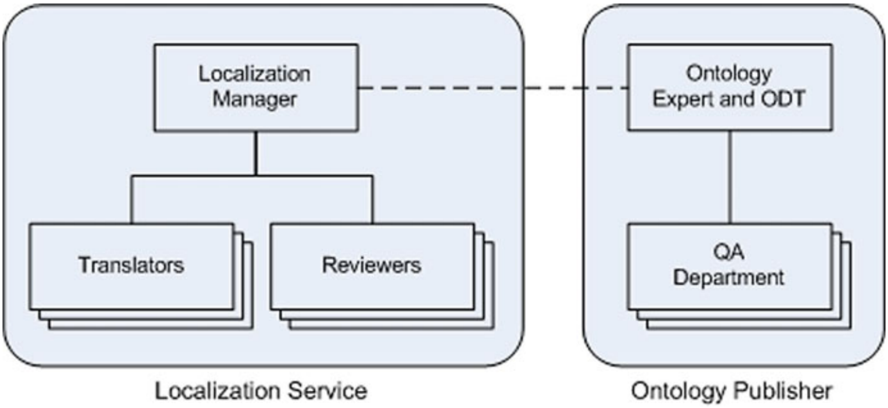


Fig. 8.1 Actors involved in the ontology localization activity

8.3.2 Ontology Localization Guidelines

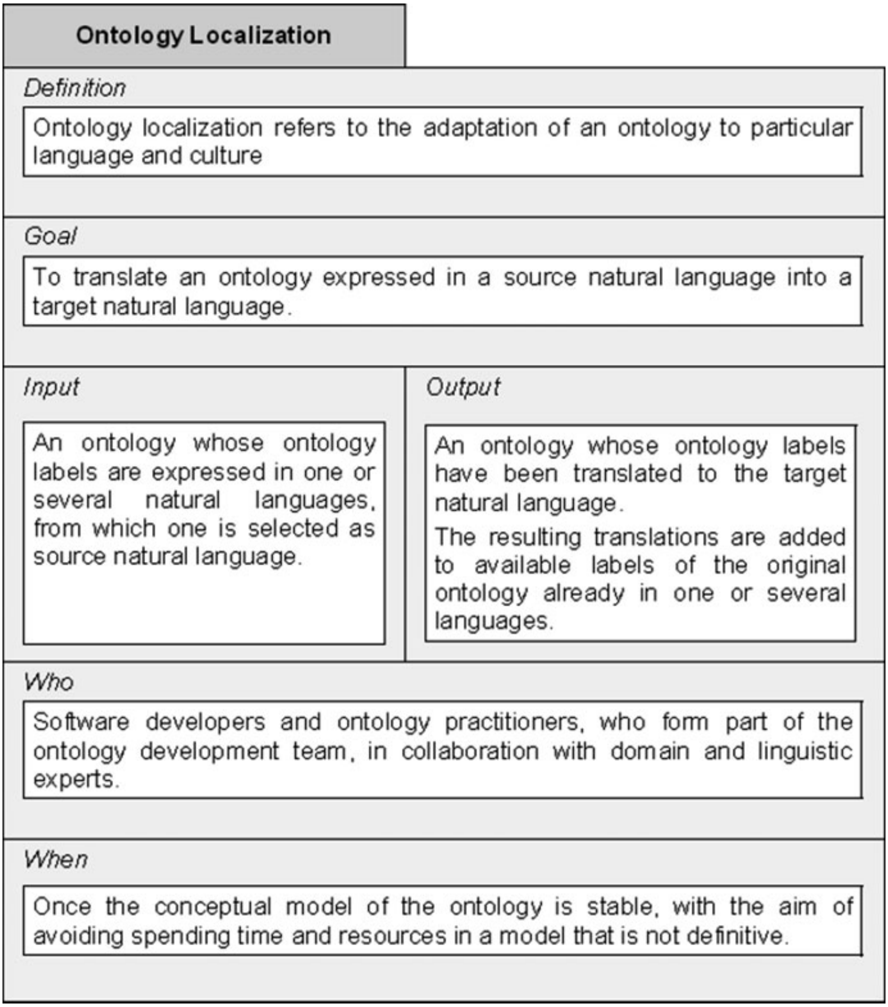
The ontology localization guidelines have been created in the context of the NeOn Methodology (see Chap. 2) to build ontology networks. Thus, taking into account the aforementioned methodological work, we provide the filling card for the ontology localization shown in Fig. 8.2. Such filling card explains the information of this activity in a practical and easy way.

The methodological guidelines for carrying out the ontology localization activity can be seen in Fig. 8.3. The workflow shows the main tasks involved, their inputs, outputs, and actors. The result of this activity is an enriched ontology (multilingual) with linguistic information (into target language) associated to each localized term.

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. Domain experts and ODT 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. The choice of a specific resource is performed manually, taking into account that the linguistic assets comply with the following characteristics:

- *Consensus*. Resources used should contain multilingual terminology consensually accepted by the community (authoritative resources), thus the effort and time spent in finding out adequate translation labels for ontology terms would decrease considerably. In this sense, internal resources, such as terminology



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

An ontology whose ontology labels are expressed in one or several natural languages, from which one is selected as source natural language.

Output

An ontology whose ontology labels 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.

Fig. 8.2 Ontology localization filling card

databases, glossaries, etc., maintained by the organization or individual itself are good representatives of consensual resources.

- *Broad coverage*. Resources should cover translation information from general to specific domain labels. It is advisable to use domain-specific resources (e.g., a glossary of financial terms or a legal dictionary) when translating domain ontologies, since they will contain the appropriate terminology. Also, since each resource supports different features and language sets, the selected resources should cover all target languages for current and possible future ontology localization projects.

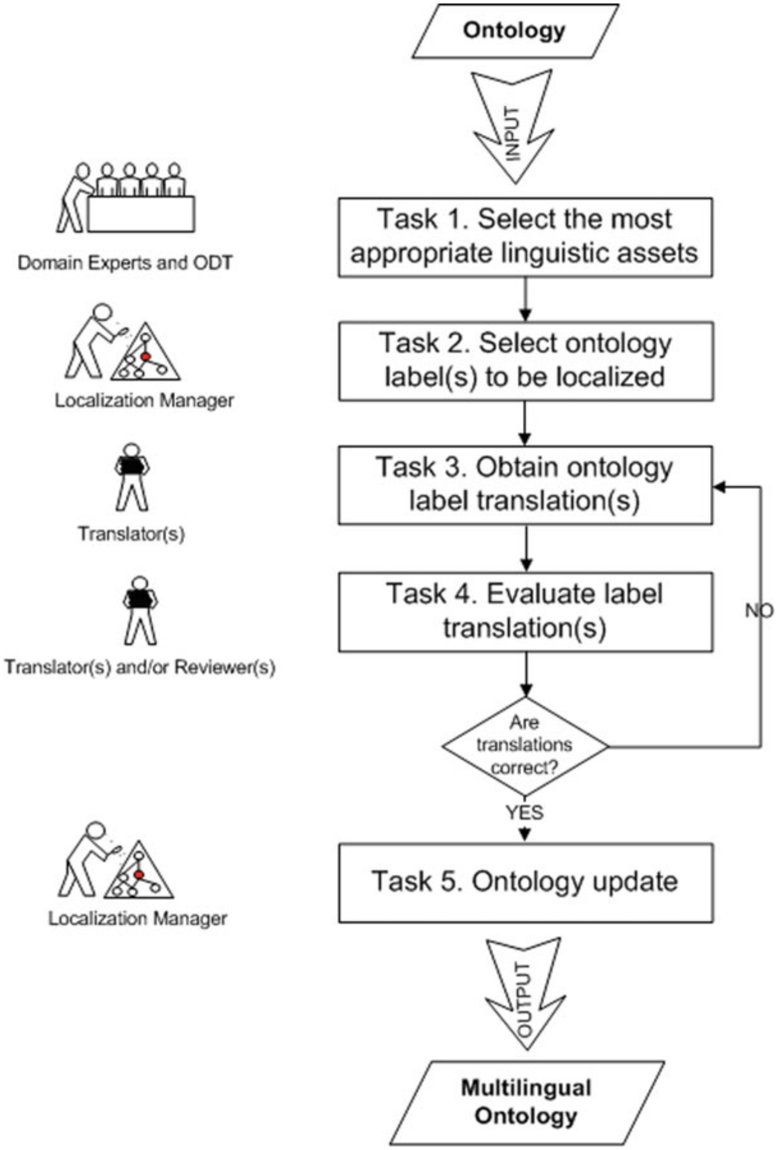

Ontology
INPUT
Domain Experts and ODT
Task 1. Select the most appropriate linguistic assets
Localization Manager
Task 2. Select ontology label(s) to be localized
Translator(s)
Task 3. Obtain ontology label translation(s)
Translator(s) and/or Reviewer(s)
Task 4. Evaluate label translation(s)
NO
Are translations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

Fig. 8.3 Tasks for ontology localization

- *High precision*. Resources used for ontology localization should be able to identify the morphological and lexicographical differences that exist between different natural languages.

To select the appropriate translation tool for performing the ontology localization activity, the preliminary guidelines presented in Table 8.1 are recommended.

Table 8.1 Ontology localization task and its corresponding tool

Type of ontology term	Translation tool	Comments
Ontology concepts, attributes, and relations	Ontology localization tool	The main difficulty at this level is related to the fact that labels for concepts, attributes, and relations are usually short (isolated) labels, not inserted in a sentence or text. Therefore, a tool designed for the purpose of translating ontologies is required at this stage to extract the label specification and its context correctly. The label context is in its turn required for discovering the label sense (for disambiguation purposes) Consider for example the word "plant," which depending on the context can be translated into Spanish as "planta" in the sense of "living organism" or "fábrica" in the sense of "industrial plant"
Ontology instances	Computer-aided translation tool or ontology localization tool	The main complication at this level is to decide which instances should be translated and which ones should not A big part of the instances are represented by proper names and, therefore should not be translated (e.g., a label containing "Michael Schumacher" should not be translated). However, other instances such as "South America" should be translated to other natural languages, as they have traditionally well-established and accepted translations
Ontology term annotations	Translation memory tool	The major cost involved at this level is the difficulty in translating correctly long pieces of text To provide a human-readable description of a term, the RDF(S) and OWL ontology languages use for example the rdfs: comment statement, where a textual comment can be added. Thus, this level involves the difficulty to translate a whole sentence (not isolated labels or terms) which is part of the annotation of concepts, attributes, or instances in ontologies

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

The goal of this task is to select the ontology label(s) to be localized. The localization manager carries out this task, taking as input an ontology whose labels are expressed in a source natural language and need to be localized to a target language.

By default, all labels of concepts, attributes, relations, and instances will be selected to be translated. However, it may happen that the ontology has been partially localized, and only the remaining labels need to be translated or retranslated if, according to the localization manager, they have not been properly translated.

The task output is a set of ontology labels and their context[5]. The context describes the meaning of a specific label in the ontology and consists of a small excerpt of ontology labels around the ontology label itself (e.g., direct hypernym labels, hyponym labels, etc.).

Task 3. Obtain ontology label translation(s).
For each ontology label, the goal of this task is to obtain the most appropriate translation in the target language. Translators carry out this task, taking as input the ontology label(s) to be localized. Different machine translation (MT) techniques (Stroppa et al. 2007; Gimpel and Smith 2008; Sato and Saito 2002) can be used to perform this task in an automatic manner. Basically, the MT techniques proposed in the literature ground their operation on some lexical or semantic resources for discovering the most appropriate translations. Thus, we can identify translation techniques based on dictionaries, terminologies, thesaurus, online services, corpora, ontologies, etc. The identification and combination of techniques will depend on two factors:

- *The type of domain knowledge represented in the ontology*. We mainly consider here two types of domains: *internationalized domains*, that is, domains whose categorization usually finds consensus among different cultures, and *culturally dependent domains*, that is, domains whose categorization is normally influenced by a certain culture.

  On the one hand, ontologies categorized within the first domain type will require translation techniques that allow identifying direct correspondences between words. Techniques based on linguistic resources such as dictionaries, terminologies, etc., can be used in this case. On the other hand, ontologies representing a culturally dependant domain (e.g., the judiciary), in which categorizations tend to reflect the particularities of a certain culture, will require translation techniques that allow identifying semantic correspondences.
- *The type of ontology element to be localized.* A second factor to be considered is the type of ontology elements to be localized. Depending on the ontology elements considered for localization, the algorithms of localization can be more or less complex. For example, the localization of ontology concepts and relations has a higher level of complexity than the localization of ontology instances because a big part of the instances are represented by proper names and have previously agreed translation or should not be translated.

The task output is a ranked set of labels in the target language for each ontology label(s).

Task 4. Evaluate label translation(s).

[5] In NLP, context refers to the environment in which a word is used and provides the information needed for figuring out the meaning of homonyms or polysemic words.

Translation quality measurements must accomplish two basic criteria:

- *Repeatable*. Two assessments of the same sample must yield similar results.
- *Reproducible and objective*. Different evaluators should arrive at a similar assessment for the same piece of translation.

The goal of this task is to evaluate label translations in the target language. At this stage, translators and/or reviewers carry out this activity taking as input the labels in the target language. The output of this task is a set of labels with its corresponding evaluation. Different linguistic criteria can be used for the evaluation of label translations. We propose two levels of evaluation criteria and for each level a set of tests, which should be automated as far as possible.

- *Semantic fidelity evaluation*. The aim of this evaluation criterion is to control that the label translation is conceptually equivalent to the ontology label in the source language. A way of evaluating the semantic fidelity is to perform a backward translation test, which 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.
- *Stylistic evaluation*. The aim is to control the clarity and syntax of the target language, which depends on the style of the source language and on the features of the individual idiolect. Special attention should be paid to certain stylistic aspects (e.g., "transport service" instead of "service of transport"), misspellings, and typos (e.g., "women" instead of "woman," "ig" instead of "big," etc.).

Task 5. Ontology update.
The goal of this task is to update the ontology with the label translations obtained for each localized label. The localization manager/QA department carries out this task taking as input the selected label translations. The activity output is an ontology enriched with labels in the target language associated to each localized term. The ontology enrichment can follow two different modeling options. If only labels in different languages are to be included in the ontology, we can make use of the rdfs:label and rdfs:comment properties of the OWL language (model 1). If, on the other hand, the final application demands further linguistic data than just labels, an external model capturing linguistic descriptions can be associated to the ontology (model 2). The choice of the modeling option for the linguistic information will be mainly determined by two factors:

- The type of domain of knowledge represented by the ontology
- The amount of linguistic information required by the final application

Taking these variables into account, we envision the two following scenarios:

- If the conceptualization represents a consensual domain, we can opt for the inclusion of multilingual information in the ontology (model 1) or for the association of an external model with the ontology (model 2). The decision between these two options will depend on the linguistic needs of the final application. An illustrative example of model 2 can be seen in Fig. 8.9

(see Sect. 8.4). If morphosyntactic data are needed for the purpose of information retrieval or information extraction, for example, the most suitable option will be the association of an external model. In the state of the art, we find some suitable models in this sense, such as LingInfo (Buitelaar et al. 2006), which enriches the ontology with morphosyntactic information, or LexInfo (Buitelaar et al. 2009), which additionally accounts for the syntactic realization of ontology terms in a certain linguistic structure.

- If the conceptualization represents a culturally dependent domain, and conceptualization mismatches among different cultures exist, we will opt for the association of an external model that permits to account for those cultural divergences at the terminological layer (model 2). In this sense, we refer to the LIR (linguistic information repository) (Peters et al. 2007; Montiel-Ponsoda et al. 2010), a model that permits to account for term variants within one language and cultural divergences across languages at the terminological layer.

8.4 Example

In this section, we include an example of the use of the proposed guidelines for the ontology localization activity and the obtained results. In particular, the example refers to the automatic localization of the "economy activity" ontology, an ontology developed in the SEEMP[6] project, using the LabelTranslator system.

LabelTranslator has been designed with the aim of automating ontology localization, and it has been implemented as a plugin of the ontology editor NeOn Toolkit. In its current version, it can localize ontologies in English, German, and Spanish. In its design, the guidelines proposed above have been followed.

In order to illustrate the results obtained by our system, we will consider the extract of the sample economy activity ontology shown in Fig. 8.4. Let us suppose that the user wants to translate the term "bars" from English into Spanish. According to the domain of the sample ontology, the correct translation of the

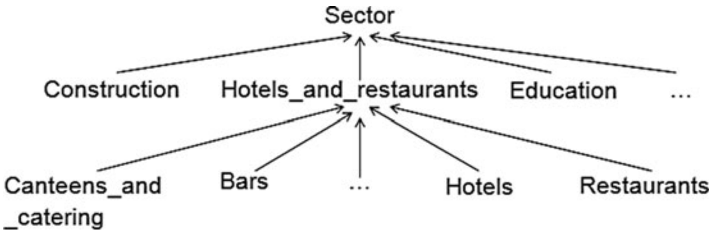


Fig. 8.4 Extract of the sample economy activity ontology

[6] http://droz.dia.fi.upm.es/hrmontology/

selected term should refer to a room or establishment where alcoholic drinks are served over a counter, not to a horizontal rod that serves as a support for gymnasts as they perform exercises to a rigid piece of either metal or wood, etc.

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

Task 1. Select the most appropriate linguistic assets.
The linguistic assets used by the current version of the LabelTranslator plugin are multilingual resources (Wiktionary or IATE), translation web services (Google Translate, BabelFish, etc.), Semantic Web resources (EuroWordNet and third-party resources retrieved through Watson[7], a search engine which indexes many ontologies available on the web), and remote lexical resources. The addition of further domain-specific resources is foreseen for domain ontologies.
Task 2. Select ontology label(s) to be localized.
Once an ontology has been created or imported in the NeOn Toolkit, LabelTranslator allows users and domain experts to manually/automatically sort out the ontology elements that should undergo localization. By right clicking on a frame (concept, attribute, or relation), the Translate action performs the translation of an ontology label (see Fig. 8.5).

For each ontology element, LabelTranslator retrieves its local context, its neighbor terms, which is interpreted by the system using a structure-level approach. In our approach, the context of an ontology term is used to disambiguate the lexical meaning of an ontology term. To determine the context of an ontology term, the system retrieves the labels of the set of terms associated with the term under consideration. The list of context labels comprises a set of names which can be direct label names and/or attributes label names, depending on the type of term that is being translated.

To mitigate risks associated with system performance, LabelTranslator limits the number of context labels used to disambiguate the translated label. Every context label is compared with the ontology label under consideration using a measure based on normalized Google distance (NGD) (Cilibrasi and Vitanyi 2004). NGD measures the semantic relatedness between any two terms, considering the relative frequency in which two terms appear in the web within the same documents. Those labels with the higher values of similarity are chosen (maximum 3[8]).

In Fig. 8.6, on the left, the dashed area represents all the context labels found for the ontology label “bars.” Our prototype finds nine labels, but only selects two (see the dotted area) to disambiguate the term. In the table on the right, we show for each type of ontology term (concept, attribute, or relation) the context labels that could

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

[8] The number of context labels used to disambiguate a translated label depends on the ontology domain. However, in our experiments we found that a threshold of three context labels reduce the time of response of the overall system and it is compatible with the range of good responses found by comparing the results with human evaluations.

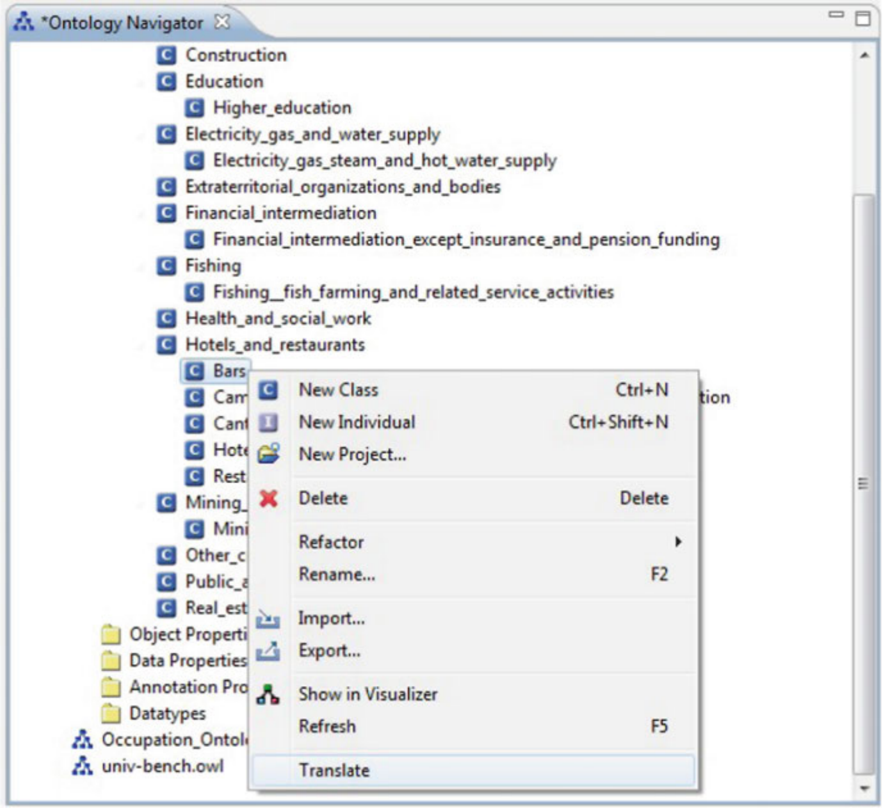


Fig. 8.5 Screenshot of the ontology navigator view with the Translate action used by the LabelTranslator plugin

be extracted. For instance, for the concept "bars" the system retrieves its hypernyms, hyponyms, attributes, and sibling concepts.

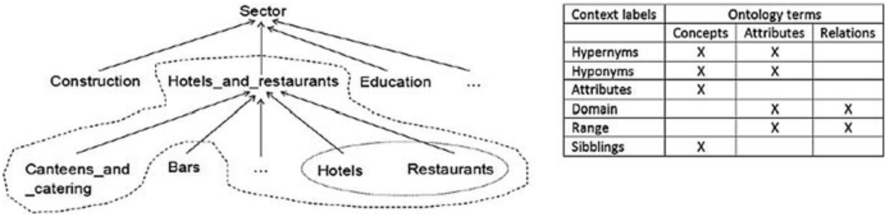
Task 3. Obtain ontology term translation(s).

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 Watson and EuroWordNet. Each sense is represented as a tuple:

$$s_k = <s, grph, descr>$$

where s is the list of synonym names, *grph* describes the sense by means of the hierarchical graph of hypernyms and hyponyms of synonym terms found in one



Context labels	Ontology terms		
	Concepts	Attributes	Relations
Hypernyms	X	X	
Hyponyms	X	X	
Attributes	X		
Domain		X	X
Range		X	X
Sibblings	X		

Fig. 8.6 Context of the ontology label "bars"

or more ontologies, and *descr* is a description in natural language of such a sense. As matching terms could correspond to ontology concepts, attributes, or instances, three lists of possible senses are associated with each translated label t: $S_t^{concept}$, $S_t^{attribute}$, $S_t^{instance}$. Notice that to perform cross-language sense translations, the external resources are limited to those resources that have multilingual information like EuroWordNet.

The multilingual retrieval of a word sense (synset) in EuroWordNet is done by means of the InterlingualIndex (ILI) that serves as a link among the different wordnets. For example, when a synset, such as "bar" with the meaning "the professional position," is retrieved from the English wordnet, its synset ID is mapped through the ILI to the synsets IDs of the same concept in the different language-dependent wordnets (German, Spanish, etc.) that describe the same concept but contain the word description in its specific language. A similar retrieval process is used in the case of multilingual ontologies but using the references between concepts and labels as offered by the standard rdfs:comment and rdfs:label properties.

Coming back to our example, in Fig. 8.7, we show the translations of the ontology label "bars" from English into Spanish; our prototype finds eight translations, but we only show three. Notice that t_1 has the desired semantics according to the similarity with the lexical and semantic ontology context (see Fig. 8.4).

- In Step 3, the system uses a disambiguation method to sort the translations according to their context. LabelTranslator carries out this task in relation to the senses of each translated label and the sense of the label under consideration. The ranking method we use to compare structures relies on an equivalence probability measure between two candidate structures, as proposed in Trillo et al. (2007). The aim is to discover whether the semantics of two ontology terms represent the same sense. At this stage, domain experts and translators may decide to choose the most appropriate translation among the ranked ones. By default, the system will consider the one in the highest position.

  In Fig. 8.8, we show a sample of the equivalent translations obtained for the term "bars." Notice that the translations obtained are ranked according to the ontology context.

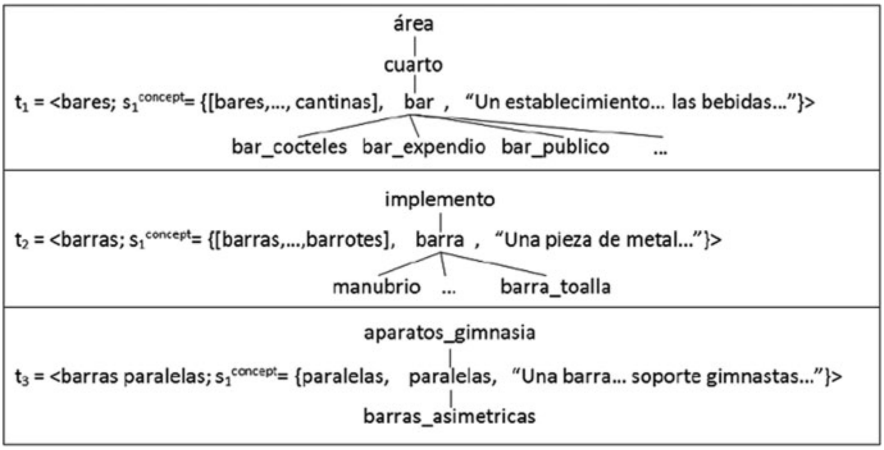
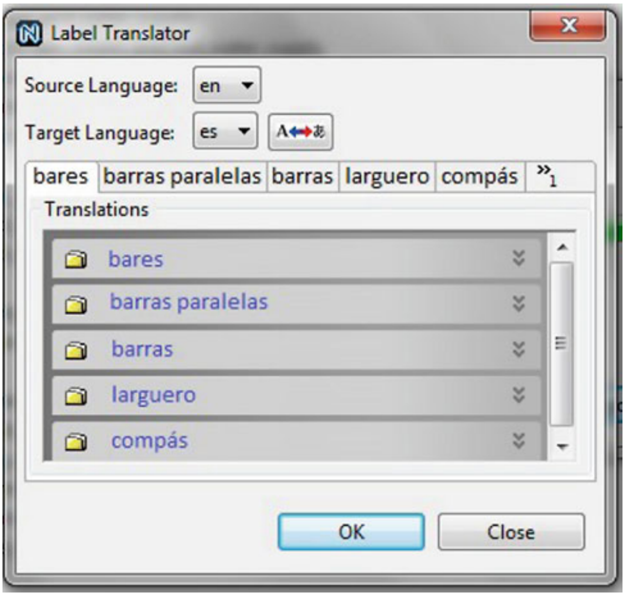

área
cuarto
t₁ = <bares; s₁concept= {[bares,..., cantinas], bar , "Un establecimiento... las bebidas..."}>
bar_cocteles bar_expendio bar_publico ...
implemento
t₂ = <barras; s₁concept= {[barras,...,barrotes], barra , "Una pieza de metal..."}>
manubrio ... barra_toalla
aparatos_gimnasia
t₃ = <barras paralelas; s₁concept= {paralelas, paralelas, "Una barra... soporte gimnastas..."}>
barras_asimetricas

Fig. 8.7 Some translations of the ontology label "bars" into Spanish



Source Language: en

Target Language: es

Fig. 8.8 Equivalent translations for the term "bars"

Task 4. Evaluate term translation(s).

The current version of LabelTranslator does not provide a method for semiautomatically evaluating the translations obtained in the previous step. Therefore, we used a manual evaluation to perform this task. Based on the NeOn methodological guidelines, we would identify the following situation:

Table 8.2 Ontology localization task and its corresponding tool

Original term (EN)	*Translation (ES)*	*Backward translation (EN)*
Bars	Bares	Bars Drinks cabinet
	Barras paralelas	Parallel bars
	Barras	Bar Rod Stick Loaf

- *Semantic fidelity evaluation***.** In order to evaluate the semantic fidelity of the translation, we would implement the "backward translation" criteria (Shigenobu 2007). Table 8.2 shows the semantic fidelity evaluation results (only few cases have been analyzed) for some terms translated into Spanish. The middle column shows the translations obtained by LabelTranslator in Spanish.

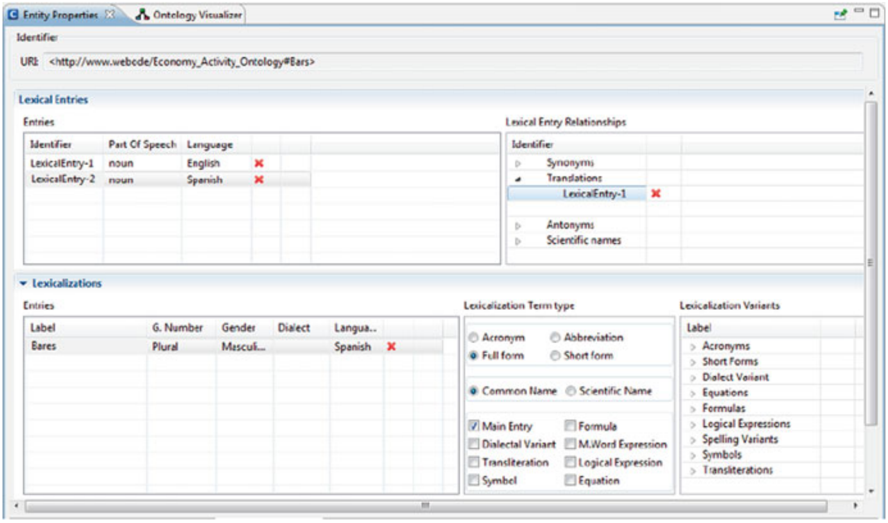
  In many cases, the backward translation did not match exactly the original meaning. Thanks to a deeper analysis, which took into consideration the context (hotels and restaurants), we identified that the translation "barras," for example, did not match the original meaning.
- *Stylistic evaluation*. The current version of LabelTranslator does not support an automated stylistic evaluation. This task was manually carried out by an expert in the domain. The translations proposed were consistent in all cases, according to the context of the ontology.

Task 5. Ontology update.

The ontology is updated with the resulting linguistic data, which are stored in the LIR model, a separate module adopted by the LabelTranslator NeOn plugin for organizing and relating linguistic information within the same language and across languages to domain ontologies. Figure 8.9 shows the linguistic information page of the sample term "bars." The linguistic page uses a model based on a modular approach to store the linguistic information associated to each ontology term. So, one can see that the translation proposed, "bares," is the full form of a term, is masculine, and is considered the main entry in this domain.

These guidelines have not been formally evaluated. Nevertheless, as shown above, we believe that the guidelines proposed are effective because following the different tasks and obtaining the expected results in each task they ensure that the progress is being achieved and that the goals of the localization activity are met at the end of it.

However, it is worth mentioning that the applicability of these guidelines has also been proved in one of the use cases of the NeOn project (Food and Agriculture Organization of the UN). In order to evaluate the quality of the translations obtained by our system, different experiments were designed. The experiments were carried out by comparing the translations provided by an expert (gold standard) with the translations provided by the ranking algorithm used in LabelTranslator.



Identifier

URI: <http://www.webode/Economy_Activity_Ontology#Bars>

Lexical Entries

Entries

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

Lexical Entry Relationships

Identifier	
▷ Synonyms	
◢ Translations	
LexicalEntry-1	✖
▷ Antonyms	
▷ Scientific names	

Lexicalizations

Entries

Label	G. Number	Gender	Dialect	Langua...	
Bares	Plural	Masculi...		Spanish	✖

Lexicalization Term type

○ Acronym ○ Abbreviation
● Full form ○ Short form

● Common Name ○ Scientific Name

☑ Main Entry □ Formule
□ 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

Fig. 8.9 Linguistic information associated to the ontology term “bars”

The ontology corpus used for the evaluation was selected from the set of the KnowledgeWeb[9] ontologies used to manage EU projects. The experimental results showed that our system suggested the correct translation 72% of the times. Also, the recall values obtained suggested that a high percentage of the correct translations were part of the final translations shown to the user. More details about these experiments can be found in Dzbor et al. (2009).

8.5 Conclusions

In this chapter, we have presented the methodological guidelines that we propose to help ontology practitioners in the localization activity. These guidelines assume that users have some knowledge on ontology localization. However, the guidelines are presented so that nonexperts can understand them. To the best of our knowledge, the study presented here is the first attempt to offer guidelines for the localization of ontologies.

These guidelines have not been formally evaluated. Nevertheless, we have validated their applicability using them in one of the ontologies used in the SEEMP project. We cannot assure that the guidelines will be valid in all localization scenarios, but further validation of the guidelines will be possible in future ontology localization projects with different settings.

[9] http://knowledgeweb.semanticweb.org/

The localization guidelines also meet the sufficient conditions of any methodological guidelines for localizing an ontology to different natural languages. Specifically, the ontology localization guidelines are:

- *Grounded on existing practices* because they have been defined by combining tasks of existing methodological guidelines
- *Collaborative* because they contemplate the participation and consensus of different actors distributed geographically
- *Open* because they do not limit the types of ontologies or the specific ontology terms (classes, object, or datatype properties) to be considered in localization nor the resources that should be employed in the actual translation
- *Usable* because they are clearly documented and their use does not involve a great effort

The applicability of the ontology localization methodology has been proved in the SEEMP project where this methodology has been used for the localization of the "occupation" ontology, by means of using the guidelines proposed in this chapter. In this project, we have proven that it is feasible to perform a manual localization using basic guidelines instead of a tool-focused approach.

References

Alonso LS, Bas LJ, Bellido S, Contreras J, Benjamins R, Gómez JM (2005) Deliverable 10.7 financial ontology, FP6-507483. In: WP10: case study eBanking

Benjamins R, Contreras J, Corcho O, Gómez-Pérez A (2002) Six challenges for the semantic web. In: Proceedings of the first international semantic Web conference (ISWC 2002). Springer, Berlin

Buitelaar P, Sintek M, Kiesel M (2006) A multilingual/multimedia lexicon model for ontologies. In: Sure Y, Domingue J (eds) The semantic Web: research and applications, 3rd European semantic Web conference (ESWC), Budva, Montenegro. Lecture notes in computer science. Springer, Berlin, pp 502–513

Buitelaar P, Cimiano P, Haase P, Sintek M (2009) Towards linguistically grounded ontologies. In: Proceedings of 6th annual European semantic Web conference, (ESWC). Lecture notes in computer science. Springer, Berlin, pp 111–125

Caracciolo, C, Sini, M, Keizer, J (2007) Requirements for the treatment of multilinguality in ontologies within FAO. In OWLED 2007 Workshop on OWL, Innsbruck (Austria). http://hdl.handle.net/10760/15660

Chun Ch, Wenlin L (2002) The translation of agricultural multilingual thesaurus. In: AFITA 2002, Asian agricultural information technology & management. Proceedings of the third Asian conference for information technology in agriculture, Beijing, pp 526–528

Cilibrasi R, Vitanyi P (2004) Automatic meaning discovery using google, manuscript, CWI

Cui G, Chen F, Chen H, Li S (2004) OntoEdu: A Case Study of Ontology-based Education Grid System for E-learning. In The Official Journal of Global Chinese Society FOR Computers in Education (GCCCE journal), Volume 2, pp 59–72

DePalma DA (2007) Moving Beyond the Ad Hocracy of Localization. Multilingual Localization: Getting Started Guide, pp. 6–8

Dzbor M, Suárez-Figueroa MC, Blomqvist E, Lewen H, Espinoza M, Gómez-Pérez A, Palma R (2009) D5.6.2 Experimentation and Evaluation of the NeOn Methodology, NeOn Project

Flied G, Kop C, Vöhringer J (2007) From OWL class and property labels to human understandable natural language. In: Proceeding of 12th international conference on applications of natural language to information systems. Lecture notes in computer science 4592. Springer, Berlin, pp 156–167

Gimpel K, Smith N (2008) Rich source-side context for statistical machine translation. In: StatMT'08: proceedings of the third workshop on statistical machine translation. Association for Computational Linguistics, Morristown, NJ, USA, pp 9–17

Guyot J, Radhouani S, Falquet G (2005) Ontology-based multilingual information retrieval. In: CLEF working notes multilingual track, pp 21–23

Liang A, Sini M (2006) Mapping AGROVOC and the Chinese agricultural thesaurus: definitions, tools, procedures. New Rev Hypermedia Multimedia 12(1):51–62

Liang A, Sini M, Chang C, Li S, Lu W, He C, Keizer J (2005) The mapping schema from Chinese agricultural thesaurus to AGROVOC. In: 6th agricultural ontology service (AOS) workshop on ontologies: the more practical issues and experiences, Vila Real, pp 1–6

Localization Industry Standards Association (LISA) What is globalization? http://www.lisa.org/What-Is-Globalization.48.0.html?&no_cache=1&sword_list[]=internationalization

Montiel Ponsoda, E, Aguado de Cea, G, Gómez-Pérez, A, and Peters, W (2010) Enriching Ontologies with Multilingual Information, Journal of Natural Language, Cambridge University Press, pp 1–27

Pazienza M, Stellato A, Zanzotto F, Henriksen L, Paggio P (2005) Ontology mapping to support ontology based question answering. In: Proceedings of the 2nd meaning workshop, Trento

Peters W, Montiel-Ponsoda E, Aguado de Cea G (2007) Localizing ontologies in OWL. In: Proceedings of OntoLex'07, co-located at the 6th international semantic web conference ISWC + ASWC 2007, Busan, South Korea

Sato K, Saito H (2002) Extracting word sequence correspondences with support vector machines. In: Proceedings of the 19th international conference on computational linguistics. Association for Computational Linguistics, Morristown, NJ, USA, pp 1–7

Schober D, Kusnierczyk W, Lewis SE, Lomax J, Members of the MSI, PSI Ontology Working Group, Mungall C, Rocca-Serra P, Smith B, Sansone SA (2007) Towards naming conventions for use in controlled vocabulary and ontology engineering. In Bioontology SIG Proceedings (ISMB 2007), Vienna, Austria, pp. 1–4

Segev A, Gal A (2008) Enhancing portability with multilingual ontology-based knowledge management. Decis Support Syst 45:567–584

Shigenobu T (2007) Evaluation and usability of back translation for intercultural communication. In: Aykin N (ed) Proceedings of the 2nd international conference on usability and internationalization (UI-HCII'07). Springer, Berlin/Heidelberg, pp 259–265

Sosnovsky S, Gavrilova T (2006) Development of Educational Ontology for c-programming. Int J Inf Theor Appl 13:303–308

Stroppa N, van den Bosch A, Way A (2007) Exploiting source similarity for SMT using context-informed features. In: Proceedings of the 11th international conference on theoretical and methodological issues in machine translation, Skvde, Sweden, pp 231–240

Tjoa AM, Andjomshoaa A, Shayeganfar F, Wagner R (2005) Semantic web challenges and new requirements. In: Proceedings of the 16th international workshop on database and expert systems applications (DEXA'05), Copenhagen

Trillo R, Gracia J, Espinoza M, Mena E (2007) Discovering the semantics of user keywords. J Univers Comput Sci 13(12):1908–1935

Vas R (2007) Educational ontology and knowledge testing. Electron J Knowl Manage 5 (1):123–130

Yang C, Li KW (2003) Automatic construction of English/Chinese parallel corpora. J Am Soc Inf Sci Technol 54(8):730–742

Zhang Z, Zhang C, Ong SS (2002) Building an ontology for financial investment. In: Intelligent data engineering and automated learning (IDEAL), data mining, financial engineering, and intelligent agents, second international conference. Springer, Berlin, pp 308–313

Chapter 9
Ontology (Network) Evaluation

Marta Sabou and Miriam Fernandez

Abstract Ontology evaluation refers to the activity of checking the technical quality of an ontology against a frame of reference. As such, it is of core importance for ontology engineering supporting scenarios such as ontology validation, knowledge selection, or the evaluation of knowledge extraction algorithms. In this chapter, we provide methodological guidelines for evaluating stand-alone ontologies as well as ontology networks. Our goal is not only to present the NeOn perspective on this issue but to also provide a practical outlook to the vast area of work in the area of ontology evaluation. Without performing an extensive state-of-the-art analysis of this research field, we aim to illustrate how various evaluation methods developed by the NeOn project, and not only, can be used at different stages of the evaluation process. We conclude the chapter with some concrete examples of performing ontology evaluation.

9.1 Motivation

Ontology (network) evaluation plays a key role in ensuring the quality of ontology networks, and it is employed within various ontology engineering scenarios. The main scenario is that of *ontology development*, namely the process during which the ontology is built. The goal in this case is to assess the quality and correctness of the obtained ontology. The process of ontology development can be achieved through different methods and the evaluation of the obtained ontology changes

M. Sabou (✉)
MODUL University Vienna, Am Kahlenberg 1, 1190 Vienna, Austria
e-mail: marta.sabou@modul.ac.at

M. Fernandez
Knowledge Media Institute (KMi), The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK
e-mail: m.fernandez@open.ac.uk

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_9, © Springer-Verlag Berlin Heidelberg 2012

accordingly. For example, an ontology could be obtained through *automatic extraction* from representative data sources such as text (Cimiano and Völker 2005) or databases (Cerbah 2008). In this case, an important research question refers to evaluating ontology extraction algorithms with respect to the quality of the produced artifacts, as well as comparing the various algorithms to each other. Ontology evaluation can often be used as a means to automatically assess the quality of the output of such algorithms.

Alternatively, the ontology development phase could also involve an *ontology evolution* activity where a base ontology is extended, either manually or through automatic means, in order to cover new domain terminology or to correspond to new application requirements (Chap. 11). In this case, the goal of ontology evaluation is to assess whether the new additions have impacted on the quality of the base ontology.

Additionally to ontology development, another scenario where ontology evaluation plays an important role is that of *ontology selection.* With the recent advances in the area of the Semantic Web, in particular the proliferation of online available ontologies and semantic search engines such as Watson[1] or Sindice[2], an increased number of applications are built by reusing external knowledge rather than building it from scratch (d'Aquin, et al. 2008). Examples include cross-ontology question answering (Lopez et al. 2010), relation detection, ontology evolution (Zablith et al. 2010), or ontology matching (Sabou et al. 2008). For these applications, it is crucial to evaluate, often entirely automatically, the quality of the reused knowledge. Ontology evaluation here refers to the situation where existing ontologies are evaluated (and often ranked) in terms of selected criteria in order to select the most appropriate one for the task at hand.

A final usage scenario is during the *ontology modularization* process that leads to a network of interconnected ontology modules (Chap. 10), whose quality is iteratively assessed in order to decide whether the modularization has reached the expected results.

In this chapter, we further explore ontology (network) evaluation by providing a definition (Sect. 9.2), methodological guidelines (Sect. 9.3), and concrete examples (Sect. 9.4).

9.2 Definitions and Filling Card

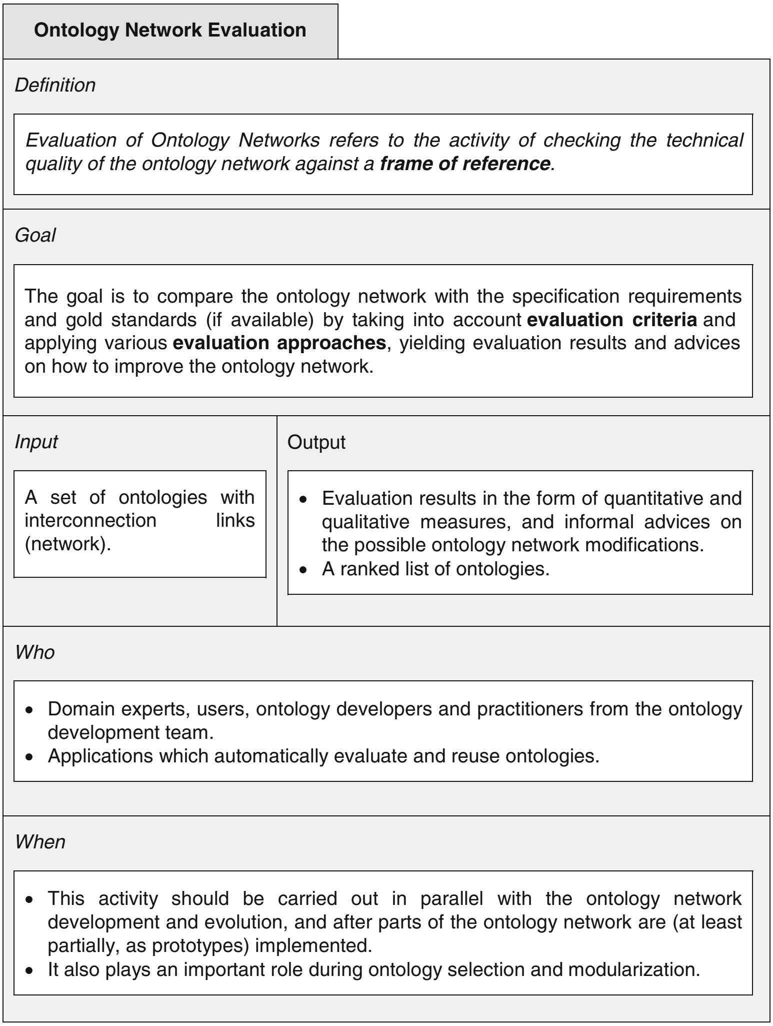
Ontology evaluation is defined as *the activity of checking the technical quality of an ontology against a frame of reference* (Suárez-Figueroa and Gómez-Pérez 2008). Intuitively, whenever an evaluation is performed for a certain ontology

[1] http://kmi-web05.open.ac.uk/WatsonWUI/

[2] http://sindice.com/

(or alignment) aspect (e.g., modeling correctness), the process is always guided by the evaluator's understanding of what is best and what is worse. In some cases, these boundaries (which we refer to as *frame of reference*) are clearly defined and tangible (e.g., a reference ontology, a reference alignment), but in other cases, they are weakly defined and may be different from one person to another, or even across evaluation sessions. The NeOn Glossary distinguishes two types of ontology evaluations depending on the frame of reference used:

- *Ontology validation* is the ontology evaluation activity that compares the meaning of the ontology definitions against the intended model of the world that it



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 gold standards (if available) by taking into account **evaluation criteria** and applying various **evaluation approaches**, yielding evaluation results and advices on how to improve the ontology network.

Input

A set of ontologies with interconnection links (network).

Output

- Evaluation results in the form of quantitative and qualitative measures, and informal advices on the possible ontology network modifications.
- A ranked list of ontologies.

Who

- Domain experts, users, ontology developers and practitioners from the ontology development team.
- Applications which automatically evaluate and reuse ontologies.

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.
- It also plays an important role during ontology selection and modularization.

Fig. 9.1 Filling card for ontology (network) evaluation

aims to conceptualize (an intangible frame of reference). 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?*

The *filling card* shown in Fig. 9.1 provides a structured summary of the ontology (network) evaluation activity. Section 2.5 describes the main components of a filling card in more detail.

9.3 Ontology Network Evaluation Workflow and Guidelines

In this section, we describe the NeOn methodological guidelines for carrying out the ontology network evaluation activity. Besides prescribing a methodology, our aim is also to provide a brief overview of the various evaluation methods and techniques that can be used in each step of the methodology.

We propose a component-based evaluation approach where each element of the network (e.g., ontologies and alignments between ontology pairs) is evaluated as a stand-alone individual and then the findings of these evaluations are summed up (Fig. 9.2). An alternative to this approach would be the evaluation of the entire network from the point of view of the users or the organization that will use the ontology network. Methodologically, this approach is similar to evaluating a stand-alone component using, for example, a task-based evaluation, and therefore, it is covered by Tasks 2 and 3 of the proposed workflow. Figure 9.2 shows the *workflow* and the tasks for carrying out the ontology network evaluation.

Task 1. Selecting individual components of the ontology network. In a first instance, the ontology development team identifies the elements of the network that need to be evaluated including individual ontologies (Maedche and Staab 2002; Burton-Jones et al. 2005; Alani et al. 2006; Fernandez et al. 2006), alignments between ontology pairs (Euzenat and Shvaiko 2007), ontology statements (Lopez et al. 2009), ontology relations, etc. Their decision should be based on two criteria: (1) which ontology network elements are critical for the overall network and (2) which of these elements 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. As we discussed before, the frame of reference is not necessarily tangible, but can be some idea of the perfect model, or canon, defined by the human evaluator for the particular evaluation task. Examples of frames of references will be given at Task 3.

Task 2. Selecting an evaluation goal and approach. For evaluating individual ontologies, the team needs to decide the goal of the evaluation and select an

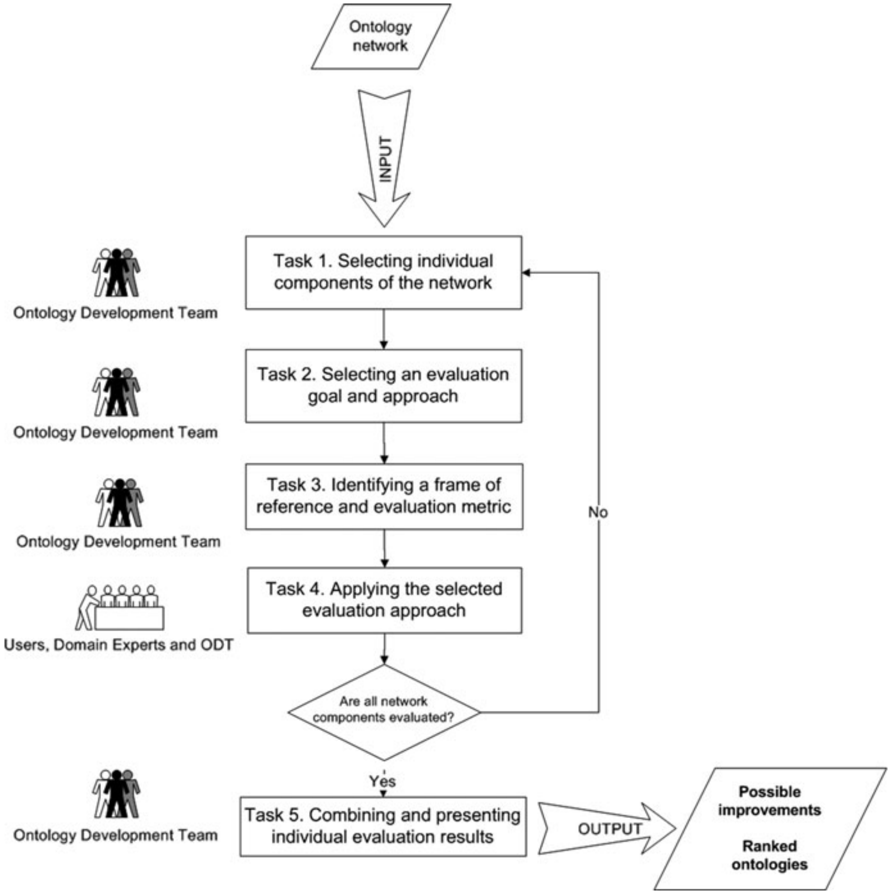

Ontology network
INPUT
Ontology Development Team
Task 1. Selecting individual components of the network
Ontology Development Team
Task 2. Selecting an evaluation goal and approach
Ontology Development Team
Task 3. Identifying a frame of reference and evaluation metric
Users, Domain Experts and ODT
Task 4. Applying the selected evaluation approach
Are all network components evaluated?
No
Yes
Ontology Development Team
Task 5. Combining and presenting individual evaluation results
OUTPUT
Possible improvements
Ranked ontologies

Fig. 9.2 Workflow and tasks for evaluating ontology networks

appropriate evaluation approach (as summarized in Table 9.1). We distinguish the following evaluation goals:

- *Domain coverage – Does the ontology cover a topic domain?* The extent to which an ontology covers a considered domain is an important factor to be considered both during the development and the selection of an ontology. The evaluation approaches employed to achieve this goal imply the comparison of the ontology to frames of references such as a gold standard ontology (Maedche and Staab 2002), or data sets that are representative for the domain (user-defined terms (Alani et al. 2006; Fernandez et al. 2006), tag sets (Cantador et al. 2007), document corpus (Brewster et al. 2004), etc.).
- *Quality of the modeling* in terms of the design and development process and in terms of the final result – *Does the ontology development process comply with*

Table 9.1 Evaluation goals, evaluation approaches, and relevant NeOn plugins

Evaluation goal	Evaluation approaches and relevant NeOn plugins
Domain coverage	Compare to a domain-specific gold standard ontology (Maedche and Staab 2002) Compare to unstructured or informal data (Brewster et al. 2004; Jones and Alani 2006) Compare to a user-defined set of terms – Sindice, Watson (Alani et al. 2006) Compare to an extended (using WordNet or other structured information sources) user-defined set of terms (Fernandez et al. 2006; Cantador et al. 2007)
Quality of modeling	Use human assessments to evaluate the syntactic, structural, and semantic quality of the ontology (Guarino and Welty 2004; Lozano-Tello and Gómez-Pérez 2004; Burton-Jones et al. 2005) Use reasoners to assess the logical correctness of the ontology (Horridge et al. 2009) Analyze the design and development process of the ontology to check its compliance with ontology modeling best practices/ODPs (Caracciolo and Heguiabehere 2009; Poveda-Villalón et al. 2009) Automatically compare to a reference alignment (Euzenat and Shvaiko 2007) Manually assess the quality of an alignment (Sabou et al. 2008) *NeOn plugins:* RaDON XDTools Alignment plugin
Suitability for an application/task	Use the ontology within an application/task and evaluate the task results and performance (Porzel and Malaka 2004; Strasunskas and Tomassen 2008; Fernandez et al. 2009) The work of Van Hage (Van Hage et al. 2007) presents two sampling-based evaluation approaches of ontology alignments
Adoption and use	Evaluation of the interlinking structure across ontologies – Sindice, Watson (Patel et al. 2003) Social rating systems (Lewen et al. 2006; Cantador et al. 2007) *NeOn plugin:* Watson for knowledge reuse

ontology modeling best practices/ODPs[3]*? Is the ontology modeled correctly?* Applicable both for the ontology development (Lozano-Tello and Gómez-Pérez 2004) and selection scenarios (Burton-Jones et al. 2005; Tartir et al. 2005), this evaluation goal focuses on the quality of the ontology which can be assessed using a wide range of approaches focusing on logical correctness or syntactic, structural, and semantic quality. Quality in terms or correctness, precision, and recall is an important goal when evaluating ontology alignments.

[3] ODP stands for Ontology Design Pattern.

- *Suitability for an application/task – Is the ontology suitable to use for a specific application/task?* (Porzel and Malaka 2004; Fernandez et al. 2009) *Will it produce the expected results?* (Strasunskas and Tomassen 2008) Different applications rely on different ontology (or alignment) characteristics. For example, for applications that use ontologies to support natural language processing tasks, domain coverage is often more important than logical correctness. As a result, measuring ontology (alignment) quality alone is not enough to predict how well the ontology (developed or selected) will support an application or a task. Task-based evaluations help assessing suitability for a task or application, rather than generic quality features.
- *Adoption and use – Has the ontology been reused (imported) as part of other ontologies?* (Sindice,[2] Watson[1]) *How did others rate the ontology?* (Cantador et al. 2007, Cupboard[4]) Understanding the extent of adoption of an ontology is of particular interest when selecting it, the assumption being that there is a direct correlation between the level of adoption and the quality of the ontology. Analyzing the degree of interlinking between an ontology and other ontologies (e.g., in terms of reused terms or ontology imports) as well as relying on social rating systems are two key approaches to achieve this goal.

Task 3. Identifying a frame of reference and evaluation metric. While in Task 2 the ontology development team decides on the key goal(s) of the evaluation and potential approaches, in Task 3, the team needs to select the concrete ingredients of the evaluation, consisting of:

- *A frame of reference – What are we comparing against?* The frame of reference denotes a set of representative resources that sets a baseline value against which the ontology should be compared.
- *Evaluation metric(s) – How to measure the features of the ontology that will be compared?* Example evaluation metrics are precision and recall, cost-based evaluation metrics, measures of similarity between an ontology or a mapping, and a corpus (domain knowledge), and lexical metrics. Table 9.2 summarizes the main evaluation metrics presented in the literature.

As exemplified in Table 9.2, evaluation metrics are generally specific for each frame of reference. There are however some generic metrics, such as precision and recall, which can be adapted for use with various frames of references.

Similarly to (Brank et al. 2005), we distinguish the following types of frames of references:

- *Gold standard*: The frame of reference is defined by a baseline ontology or some other kind of structured representation of the problem domain for which an appropriate ontology is needed. A gold standard is often used when the goal of

[4] http://cupboard.open.ac.uk:8081/cupboard-search/

Table 9.2 Evaluation metrics used for various evaluation frameworks

Frame of reference	Evaluation metric/approach
Gold standard	*Interpretability:* amount of terms of the ontology that have a WordNet[a] sense (Burton-Jones et al. 2005) *Clarity:* amount of WordNet senses of the ontological terms (Burton-Jones et al. 2005) *Lexical similarity:* average string matching between the set of gold standard terms and the set of ontology terms (Maedche and Staab 2002) *Taxonomical similarity:* maximum overlap between the concepts of the gold standard and the concepts of the ontology in terms of their "semantic cotopy" (their sets of super- and subconcepts) (Maedche and Staab 2002) *Relation similarity:* overlap between the relations of the gold standard and the relations of the ontology considering the geometric mean value of how similar their domain and range concepts are (Maedche and Staab 2002) *Precision and recall of an alignment with respect to a reference alignment* (gold standard): precision measures the ratio of correctly found correspondences (true positives) over the total number of returned correspondences (true and false positives). Recall measures the ratio of correctly found correspondences (true positives) over the total number of expected correspondences (true positives and true negatives) (Euzenat 2007) *Semantic precision/semantic recall for alignment evaluation:* This measure proposes an abstract generalization of precision and recall to discriminate among different degrees of alignment correctness (Euzenat 2007)
Application-based	*History:* number of times an ontology has been accessed *Insertion, deletion, and substitution errors:* errors according to the improvements in the task's output after fixing these errors in the employed ontology (Porzel and Malaka 2004) *Search task fitness* and *search enhancement capability:* these measures evaluate ontology quality in the context of an ontology-driven web search task (Strasunskas and Tomassen 2008) *Watson's topology measures:* these measures are used in the context of a relation correctness evaluation task (Fernandez et al. 2009)
Data-driven	*Class match:* coverage of an ontology with respect to a set of search terms (Alani et al. 2006) *Best fit ontology:* ontology that maximizes its conditional probability given a corpus. The probability is computed considering the terms and document clusters within the corpus (Brewster et al. 2004)
Assessment by humans	*Syntactic quality:* number of syntactical errors in the ontology (Burton-Jones et al. 2005) *Accuracy:* number of false statements in the ontology (Burton-Jones et al. 2005) *Trust:* correctness and usefulness of the information delivered by a certain reviewer with respect to the ontology (Lewen et al. 2006). This measure is defined and exploited in collaborative systems (d'Aquin et al. 2009) *Collaborative evaluation:* collaborative assessment of ontologies based on manual user evaluation (Cantador et al. 2007) *Essence:* assess if an entity is true in every possible world (Guarino and Welty 2004)

(continued)

Table 9.2 (continued)

Frame of reference	Evaluation metric/approach
	Identity: assess if individual entities of the world are the same or different (Guarino and Welty 2004) *Unity:* recognizes all the parts that form an individual entity (Guarino and Welty 2004)
Topology-based	*Topology of the graph:* set of topological evaluation measures including *number of classes, number of properties, number of individuals, ontology popularity* (number of ontologies importing a given ontology), and *ontology depth and breadth* (maximum, minimum, average, and variance); extracted from Watson *Density:* number of subclass, sibling, and domain relations of a given concept (Alani et al. 2006) *Semantic similarity:* closeness of the concepts of interest in the ontology structure (Alani et al. 2006) *Betweenness:* number of paths that pass through each node of the ontological graph (Alani et al. 2006) *Comprehensiveness:* number of classes and properties of an ontology (Burton-Jones et al. 2005) *Authority:* normalized value of times that an ontology is imported in the network (Burton-Jones et al. 2005) *OntoRank:* ranks ontologies based on the interlinking structure among ontologies in the network. Different versions of the similar evaluation principle are found in (Patel et al. 2003; Ding et al. 2005) *Relationship richness:* diversity of relations and placement of relations in the ontology (Tartir et al. 2005) *Attribute richness*: average number of properties per class (Tartir et al. 2005) *Inheritance richness*: average number of subclasses per class (Tartir et al. 2005) *Class richness*: ratio between the number of classes that contain instances divided by the total number of classes in the ontology (Tartir et al. 2005) *Average population:* ratio between the number of ontology instances and classes (Tartir et al. 2005) *Cohesion:* number of separated, connected components of the ontological graph (Tartir et al. 2005)
Language-based	Thirty-eight modeling language-specific criteria: if the *language allows axioms embedded in terms*, can *define disjoint decompositions*, etc. (Lozano-Tello and Gómez-Pérez 2004)
Methodology-based	Eleven methodology-based evaluation metrics: precision factors (e.g., the *delimitation of phases in the ontology construction)*, usability factors (e.g., the *quality of manuals*), and maturity factors (e.g., the *importance of the developed ontology*) (Lozano-Tello and Gómez-Pérez 2004)

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

the evaluation is *domain coverage*. For alignments, a reference alignment can play the role of a gold standard.

- *Application-based*: The frame of reference consists of the set of "ideal" results that an application should return when plugging the "perfect" ontology

(or alignment) into it. This frame of reference pertains to the assessment of the ontology's (alignment's) *suitability for an application/task*.
- *Data-driven*: The frame of reference is a collection of unstructured or informal data (e.g., text), which represents the problem domain. Similarly to structured representations used as gold standards, unstructured data collections are also mostly used to support the evaluation of *domain coverage*.
- *Assessment by humans*: The frame of reference is defined by human judgments that measure ontology features (or alignment characteristics) not recognizable by machines. Humans can (relatively) easily assess several *ontology quality* features which are not amenable to automatic processing. Human ratings also help to assess the level of *adoption and use* of the ontologies. Human-based ontology ratings are exploited to automatically select the most appropriate ontology according to previous users' experiences (Cantador et al. 2007).

Additionally, and based on the way in which human evaluators assess *ontology quality* features (by comparison with their mental idea of the perfect model or canon for these features), we have identified the next three nontangible frames of references as ideal models of topologies, languages, and ontology-construction methodologies, which constitute the boundaries within which comparisons are based when performing the evaluations: (a) the ontology with the optimal topology, (b) the potentially most powerful and expressive ontology language, and (c) the perfect set of steps to follow and requirements to fulfill in order to achieve the best modeled ontology. All these canons or ideal models of topologies, languages, and methodologies are weakly defined since they may vary across evaluations and across the evaluators who defined them.

- *Topology-based:* The frame of reference is defined by the minimum or maximum possible values of the topology evaluation metrics among ontologies within the network, or among ontology entities within the same ontology. Topology metrics automatically assess *ontology quality* features as well as *adoption and use* features, by measuring the interlinking structure of ontologies across the network (Ding et al. 2005).
- *Language-based:* The frame of reference is defined by the representational capabilities of the language used to construct the ontology.
- *Methodology-based:* The frame of reference is defined by the different quality factors of the selected ontology-development methodology.

Task 4. Applying the selected evaluation approach. Applying the selected evaluation approach requires a proper setup for the evaluation experiments and implementation of software tools to compute the evaluation metrics, and/or engage the human experts in stimulating sessions to collect their evaluations. We advise ontology developers to refer to the relevant scientific publications cited in this chapter for example evaluation setups and best practices. Evaluation approaches that rely on human judgment (Guarino and Welty 2004; Lozano-Tello and Gómez-Pérez 2004) are generally more time consuming and sophisticated than those which compare numeric values derived by automatic measures (Sindice, Watson), although they often offer more valuable insight into the evaluation process. We advise using parallel evaluation with multiple human experts to account for cross-evaluator disagreements.

Task 5. Combining and presenting individual evaluation results. This task highlights the weakest spots in the ontology network by considering individual evaluation results and how they affect the rest of the network. The evaluation results derived for individual components are combined to reach a global understanding of the network's quality. The final task is to present the results of the evaluation in an appropriate form for possible repair (corrections, additions), improvements, and future evolution of the ontology network.

9.4 Examples of Ontology Evaluation

Since ontology network evaluation is not a widespread activity as yet, in this section, we present examples of various ontology evaluation studies and show how their stages map to the tasks prescribed by our guidelines. The examples cover all the key evaluation goals described in Task 2: domain coverage (Sect. 9.4.3), quality of modeling (Sects. 9.4.1 and 9.4.2), suitability for an application (Sects. 9.4.3 and 9.4.4), and adoption (Sect. 9.4.5).

9.4.1 Evaluation of an Individual Ontology

In this example, we describe the evaluation of YAGO (Suchanek et al. 2008), a large, lightweight, general-purpose ontology, automatically derived from Wikipedia and WordNet. YAGO has 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. YAGO's evaluation follows the main tasks of our methodology.

[*Task 2*] Since the evaluation was performed in an ontology development scenario, the authors' goal was to assess the *quality of modeling* of YAGO, namely its precision with respect to the data sets from where it has been derived. The approach was that of evaluating the precision by using human expert opinion.

[*Task 3*] 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 to be used as a *frame of reference*, the authors relied on manual evaluations against Wikipedia content, which was the frame of reference.

[*Task 4*] During the evaluation, human judges rated as "correct," "incorrect," or "don't know" facts that were randomly selected from YAGO. 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

Table 9.3 Precision of some YAGO facts

Relation	No. of evaluation	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%
...		
88 hasGDPPPP	75	91.22189% ± 5.897%
89 hasGini	62	91.00750% ± 6.455%
90 discovered	84	90.98286% ± 5.702%

with the problem of Wikipedia containing some false information). Thirteen judges evaluated a total of 5,200 facts (ground relations between YAGO entities).

[*Task 5*] The authors use a tabular format (Table 9.3) to present the evaluation results in the decreasing order of the obtained precision (we only show the most and least precise relations). To make sure that the findings are significant, the Wilson confidence interval for $\alpha = 5\%$ was computed. A confidence interval of 0% means that the facts have been evaluated exhaustively. The evaluation shows very high quality results as 74 relations have a precision of over 95%.

This tabular presentation helps identifying the least precise relations and fosters the analysis of such cases. It can be concluded, for example, that a key 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., persons born in 1802 are in the *1805 Births* Wikipedia category). For facts with literals (such as *hasHeight*), many errors stem from a nonstandard format of the numbers (e.g., height is considered 1.6 km, just because the infobox says 1,632 m instead of 1.632 m). Occasionally, the data in Wikipedia was updated between the time of extraction and the time of the evaluation. This explains many errors for frequently changing properties such as *hasGDPPPP* and *hasGini*.

9.4.2 Pattern-Based Ontology Evaluation

In this section, we show how ontology design patterns, specifically content design patterns (CPs), are used to evaluate an ontology. The example does not cover the complete evaluation of the ontology, but presents one specific case where a CP assisted in finding potential problems and additionally suggested a solution. The example is set within the fishery domain, and the evaluated ontology is version 0.3 of the “fishing areas” ontology, modeling the division of water areas into divisions and subdivisions. An example is the FAO major fishing area 51, Western Indian Ocean, and its subareas numbered from 1 to 8, where 1 corresponds to the Red Sea and 2 to the Persian Gulf, but where the subdivisions of these subareas are only numerically identified.

[*Task 2*] The goal of the evaluation was assessing the *quality of modeling*, and the chosen approach was manual evaluation by an ontology pattern expert.

[*Task 3*] The expert used the pattern catalog available in the ontology design pattern portal[5] as a "gold standard" of modeling to which the modeling solutions in the evaluated ontology were compared. CPs introduce best practices for solving particular modeling problems, but by introducing those solutions, the pattern catalog can also be seen as a catalog of modeling issues.

[*Task 4*] The ontology used a locally defined, transitive, "part-of" relation to model the division of subareas and further levels of divisions and subdivisions, thus using the same modeling approach as the "part-of" content pattern. This modeling solution, however, is not suitable for certain contexts, because, when using reasoning, it is not possible to distinguish between the direct and the indirect subparts of an area. For example, if the hierarchical structure of the partitioning of the areas should be reconstructed, for example, for browsing the ontology in a graphical interface, or when answering "what are the divisions of the Red Sea?," only the direct subareas of the Red Sea are of interest rather than all the inferable parts.

The "componency pattern" provides a modeling alternative using two inverse object properties: "hasComponent" and "isComponentOf." These are nontransitive properties that can be used in combination with the "part-of pattern" to both register general partitioning but also the nontransitive property of a "proper part," i.e., a direct component of something. When using these two patterns as "gold standards" for modeling, the ontology evaluator can discover the potential problem of a missing nontransitive property to distinguish the different "levels" of area decomposition and propose an appropriate solution.

9.4.3 Multiple Evaluations of an Ontology

An example of how various types of evaluations shed light on different aspects of an ontology is provided in (Sabou et al. 2005). Similar to this, when evaluating ontology networks, one needs to combine evaluation results for various network components. The authors of (Sabou et al. 2005) report on the multifaceted evaluation of an ontology that was automatically extracted from a corpus of textual web service descriptions in the bioinformatics domain. The various stages of this evaluation are graphically depicted in Fig. 9.3. The aim of the *extracted ontology* is to support the semantic description of web services. The *myGrid*[6] project provided a good context to evaluate this ontology as a bioinformatics expert has previously built a *gold standard* ontology for describing the same set of web

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

[6] http://www.mygrid.org.uk

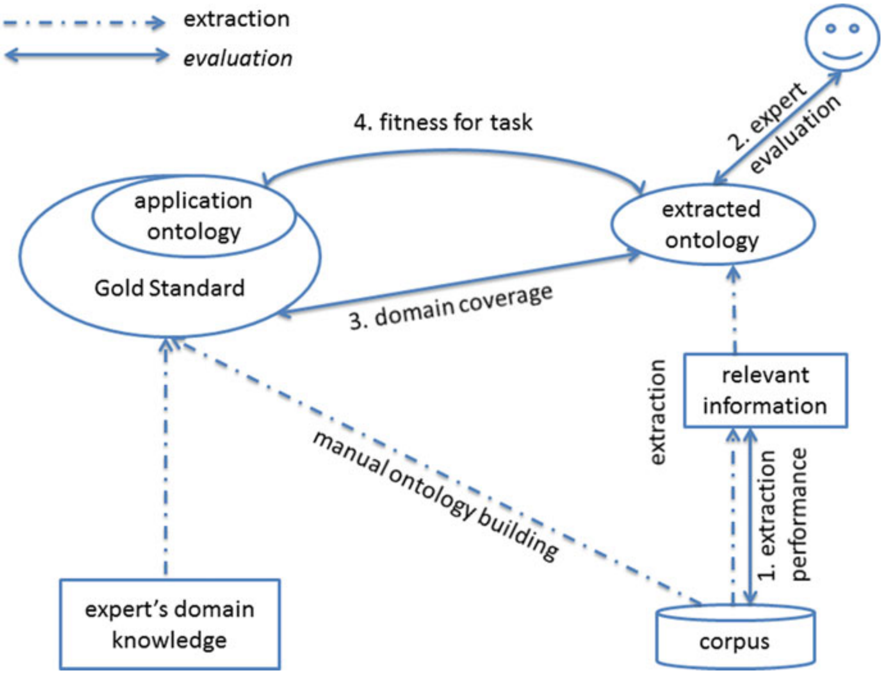


Fig. 9.3 Overview of various evaluations of an ontology (Sabou et al. 2005)

services. The domain expert has relied on his *domain knowledge* to build the ontology rather than on the description of web services (*corpus*), which were used as the main input for the automatic extraction algorithm. A part of the gold standard ontology, referred to as the *application ontology*, provides concepts for annotating web service descriptions in a form-based annotation tool and is subsequently used at web service discovery time to power the search.

[*Task 2*] In this ontology development scenario, the evaluations had several complementary goals. First, the authors aimed to assess whether the extracted ontology would be a good starting point for building an ontology and relied on an expert evaluation approach for this (shown as evaluation 2 in Fig. 9.3). Second, they wanted to evaluate *domain coverage* by comparison to the gold standard ontology (shown as evaluation 3 in Fig. 9.3). Third, the authors got an insight into how well the ontology would *support an application* by comparing it with the application ontology.

[*Task 3*] The authors made use of the following frames of references and metrics. For evaluation 2, the frame of reference consisted in the expert's knowledge of the domain as he was asked to review and rate the extracted concepts as either *correct* or *spurious* or *new*. A precision value was then computed as a ratio of the correct and new concepts over all extracted concepts. For evaluation 3, the authors used the gold standard ontology as a frame of reference and computed metrics such as *lexical overlap* (LO – the ratio of overlapping concepts), *ontological improvement*

Table 9.4 Results for domain coverage and task fitness from (Sabou et al. 2005)

Concepts	Gold standard	Application ontology
All	549	125
Correct	39	25
All_{missed}	510	100
$Missed_{corpus}$	3 (0.6%)	0 (0%)
$Missed_{external}$	360 (70.6%)	88 (88%)
$Missed_{abstract}$	101 (19.8%)	6 (6%)
$Missed_{composed}$	46 (9%)	6 (6%)
New	306	27
LO	7%	20%
OL	93%	80%
OI	56%	21.5%

(OI – the ratio of new concepts that were not in the gold standard but were domain relevant), and *ontological loss* (OL – the ratio of gold standard concepts which were not extracted). For evaluation 4, the application ontology was used as a frame of reference and compared to the extracted ontology using the metrics defined for evaluation 3.

[*Task 4*] Task 4 consisted in the evaluation performed by the domain expert as well as the computation of the various ontology comparison metrics.

[*Task 5*] The authors sum up the results of the various evaluations in tabular form and perform a subsequent analysis of these results. For example, Table 9.4 sums up the results when assessing domain coverage and suitability for a task by comparing the extracted ontology to the gold standard and application ontologies. The results show that although the overlap with the gold standard is low (7%), the extracted ontology contains a significant number of new, domain-relevant concepts (56%) that were identified in the automatically analyzed corpus but missed by the domain expert, which relied exclusively on his domain knowledge. A detailed analysis of all the missed concepts when comparing to the gold standard ontology shows that 70.6% of these terms did actually not appear in the corpus (but could be acquired if the corpus would be enlarged) and 19.8% referred to abstract concepts introduced by the domain expert to structure the ontology and which again were not in the corpus. It turns out that extraction algorithm–related issues only account for only 10% of the missed concepts.

9.4.4 Task-Based Ontology Evaluation

The authors of (Strasunskas and Tomassen 2008) investigate which ontology features influence the web search task. In their study, they consider different types of search tasks (fact-finding, exploratory search, comprehensive search), identify ontology features important for each task, and then introduce new evaluation metrics that measure these features respectively (e.g., fact-finding fitness

(FFF), exploratory search task fitness (EXF)). Such metrics can support ontology selection for search. Their theoretical considerations are experimentally verified, by correlating the values of the metrics for different ontology versions with the search performance obtained in the context of the WebOdIR web search application (Strasunskas and Tomassen 2008). Core to their study is therefore a task-based evaluation of ontologies.

[*Task 2*] The goal is to understand the *suitability for a task*, and the approach consists in exploiting ontologies to support web search and measuring the improvement in terms of search precision obtained in an experimental setting.

[*Task 3*] The frame of reference is defined by the *performance scores obtained in a web search task* with an original version of the ontology. The metrics used measure ontology features important for certain search tasks (e.g., FFF, EXF).

[*Task 4*] The experimental setup consists of relying on two groups of users to perform web search using WebOdIR within four different domains (two search tasks per domain, i.e., eight tasks in total). WebOdIR exploited a set of ontologies for one group and the extended version of the same ontologies for the second group. The performance score of the search task is computed and compared across the two versions of the ontologies as well as correlated with the computed values of the newly introduced metrics.

[*Task 5*] The authors present these correlations in both tabular and graphical form and conclude on the influence of ontology features on various search tasks. For example, they found that more instances and object properties improve fact finding, while the addition of disjoint and equivalent concepts is beneficial for explanatory and comprehensive search tasks.

9.4.5 Evaluating Ontology Adoption and Use

The work of Cantador and colleagues (Cantador et al. 2007) presents a tool for collaborative ontology evaluation and reuse (WebCORE) focused on evaluating *domain coverage* and *adoption and usage*. The goal of this tool is to help experts and practitioners to select the most appropriate ontologies from a repository. The tool has three main components. The first one helps the user to semiautomatically generate a gold standard representing the domain of interest. The second component evaluates the domain coverage of the ontologies by comparing them against the previously generated gold standard by means of lexical and taxonomical evaluation measures. The third component exploits previous users' judgments of those ontologies to automatically recommend the best ones.

[*Task 2*] Two main evaluation goals are considered when selecting the optimal ontology: (a) the *domain coverage* and (b) the *adoption and use* of the ontology.

[*Task 3*] To evaluate *domain coverage*, authors select a gold standard as a frame of reference. This gold standard is a representation of the domain of interest and is semiautomatically generated by the user with the support of the tool. To generate it, the user (a) introduces an initial set of terms or selects a textual source from which a set of terms representing the domain of interest can be extracted, (b) complements this set of terms by selecting additional terms from a ranked list, automatically generated by the system by considering previous user-generated gold standards, and (c) extends this set of terms by selecting suggested hypernym, hyponym, and synonym relations from WordNet. To evaluate the *adoption and use* of the ontologies, this work relies on an *assessment by humans*' frame of reference. Users share their own experiences by evaluating the used ontologies according to five criteria: correctness, readability, flexibility, level of formality (highly informal, semi-informal, semiformal, and rigorously formal), and type of model (upper-level, core-ontology, domain-ontology, task-ontology, and application-ontology).

[*Task 4*] The tool evaluates the ontologies in two phases. First, the ontologies are evaluated according to their domain coverage by comparing them against the semiautomatically generated gold standard using lexical and taxonomical similarity measures. Second, the ontologies with sufficient domain coverage are assessed on their level of adoption and use with the help of a collaborative filtering algorithm (Adomavicius and Tuzhilin 2005) that explores the manual evaluations of the ontologies stored into the system. This algorithm takes into account not only previous users' experiences (usage) but also the number of times the ontologies were selected (adoption).

[*Task 5*] The representation of the results differs for the two types of evaluations. For domain coverage, the tool presents a ranked list of ontologies including their individual scores for the lexical and taxonomical evaluation measures, as well as a combined evaluation score. After the adoption and usage evaluation, the list of ontologies is reranked, and the collaborative ontology evaluation score is added to the previous scores. In addition, the system allows the user to provide her own judgment of the ontology so that her assessment can be exploited for future ontology evaluations and selections.

9.5 Relevant NeOn Toolkit Plugins

Given the complexity of the ontology evaluation task in terms of the variety of approaches and metrics, the NeOn Toolkit does not provide an evaluation plugin per se. However, various plugins exist that can support different evaluation approaches. We provide a brief description of these plugins here.

The *RaDON* plugin[7] supports the automatic detection of logical inconsistency and incoherence in an ontology or an ontology network. The plugin does not only detect these modeling errors but can also repair them automatically or support the user to manually solve these issues. As such, RaDON can support users whose goal is to assess the *quality of modeling* in their ontology.

[7] http://www.neon-toolkit.org/wiki/2.3.1/RaDON

The *XDTools* plugin[8] contains a suite of tools that support design pattern–based ontology development. One of the tools, XD Analyzer, provides suggestions and feedback to the user with respect to how good practices in ontology design have been followed, according to the eXtreme Design (XD) method (for instance, missing labels and comments, isolated entities, unused imported ontologies). Chapter 3 provides more information about the XD method. Similarly to RaDON, this plugin can also be used when checking the *quality of modeling*; however, the focus here is the quality of the domain conceptualization rather than logical correctness.

The *Watson for knowledge reuse*[9] plugin primarily supports knowledge reuse by allowing an ontology developer to search the Watson ontology search engine for relevant knowledge statements directly from within the NeOn Toolkit and then reuse those statements. The plugin also interfaces with the Cupboard ontology publication environment that allows users to rate various characteristics of the ontologies that they reused (e.g., reusability, correctness, completeness, domain coverage, modeling style). Individual ratings are aggregated into an overall score and can support other people when reusing ontologies. This plugin supports the evaluation of ontologies in terms of their *adoption and use* providing also reviews written by previous adopters.

9.6 Summary

Ontology evaluation is an important and complex ontology engineering activity. Its complexity stems both by its applicability in a variety of scenarios (Sect. 9.1) as well as the abundant number of existing approaches and metrics. In this chapter, we aimed at providing practitioners with the right balance of generic guidelines and specific techniques that they could use from the wide landscape of works in this area (Sect. 9.2). We hope that the five diverse evaluation examples in Sect. 9.3 will serve as useful material for exemplifying the proposed guidelines.

Although ontology networks contain both ontologies and their links in terms of alignments, we have mostly focused on ontology evaluation. Readers interested in ontology alignment evaluation should also consult Chap. 12. Finally, Chaps. 10 and 11 describe other ontology engineering activities that can benefit from ontology evaluation, namely ontology modularization and evolution.

[8] http://www.neon-toolkit.org/wiki/2.3.1/XDTools

[9] http://www.neon-toolkit.org/wiki/2.3.1/Watson_for_Knowledge_Reuse

References

Adomavicius G, Tuzhilin A (2005) Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Trans Knowl Data Eng 17(6):734–749

Alani H, Brewster C, Shadbolt N (2006) Ranking ontologies with AKTiveRank. In: 5th international Semantic Web Conference (ISWC 2006), Athens, GA, USA, pp 1–15

Brank J, Grobelnik M, Mladenić D (2005) A survey of ontology evaluation techniques. In: Conference on Data Mining and Data Warehouses (SiKDD 2005), Ljubljana, Slovenia, pp 166–170

Brewster C, Alani H, Dasmahapatra S, Wilks Y (2004) Data driven ontology evaluation. In: 4th international conference on Language Resources and Evaluation (LREC 2004), Lisbon, Portugal, pp 164–169

Burton-Jones A, Storey VC, Sugumaran V, Ahluwalia P (2005) A semiotic metrics suite for assessing the quality of ontologies. Data & knowledge engineering – Special issue: Natural Language and Database and Information Systems: NLDB 2003, pp 84–102

Cantador I, Fernandez M, Castells P (2007) Improving ontology recommendation and reuse in WebCORE by Collaborative Assessments. In: Workshop on social and collaborative construction of structured knowledge at the 16th international World Wide Web conference (WWW 2007), Banff, Canada

Caracciolo C, Heguiabehere J (2009) NeOn deliverable D7.2.3. Initial network of fisheries ontologies. NeOn project

Cerbah F (2008) Learning highly structured semantic repositories from relational databases – RDBtoOnto tool. In: 5th European Semantic Web Conference (ESWC 2008), Tenerife, Spain, pp 777–781

Cimiano P, Völker J (2005) Text2Onto – a framework for ontology learning and data-driven change discovery. In: 10th international conference on applications of Natural Language to Information Systems (NLDB-2005), Alicante, Spain, pp 227–238

d’Aquin M, Motta E, Sabou M, Angeletou S, Gridinoc L, Lopez V, Guidi D (2008) Towards a new generation of semantic web applications. IEEE Intell Syst 23(3):20–28

d’Aquin M, Euzenat J, Duc C, Lewen H (2009) Sharing and reusing aligned ontologies with cupboard. Demo at international conference on Knowledge Capture (K-CAP 2009), Redondo Beach, CA, USA

Ding L, Pan R, Finin T, Joshi A, Peng Y, Kolari P (2005) Finding and ranking knowledge on the semantic web. In: 4th international Semantic Web Conference (ISWC 2005), Galway, Ireland, pp 156–170

Euzenat J (2007). Semantic precision and recall for ontology alignment evaluation. In: 20th international Joint Conference on Artificial Intelligence (IJCAI-2007), Hyderabad, India, pp 348–353

Euzenat J, Shvaiko P (2007) Ontology matching. Springer, Heidelberg

Fernandez M, Cantador I, Castells P (2006) CORE: a tool for collaborative ontology reuse and evaluation. In: 4th international workshop on evaluation of ontologies for the web at the 15th international World Wide Web conference (WWW 2006), Edinburgh, Scotland

Fernandez M, Overbeeke C, Sabou M, Motta E (2009) What makes a good ontology? A case-study in fine-grained knowledge reuse. In: 4th Asian Semantic Web Conference (ASWC 2009), Shanghai, China, pp 61–75

Guarino N, Welty C (2004) An overview of OntoClean. In: Handbook on ontologies. Springer, Berlin, pp 151–172

Horridge M, Parsia B, Sattler U (2009) Explaining inconsistencies in OWL ontologies. In: Scalable uncertainty management. Springer, Berlin/Heidelberg, pp 124–137

Jones M, Alani H (2006) Content-based ontology ranking. In: 9th international protege conference, Stanford, CA

Lewen H, Supekar K, Noy N, Musen M (2006) Topic-specific trust and open rating systems: an approach for ontology evaluation. In: 4th international workshop on Evaluation of Ontologies for the Web (EON2006) at the 15th international World Wide Web conference (WWW 2006), Edinburgh, Scotland

Lopez V, Nikolov A, Fernandez M, Sabou M, Uren V, Motta E (2009) Merging and ranking answers in the semantic web: the wisdom of crowds. In: 4th Asian Semantic Web Conference (ASWC 2009), Shanghai, China, pp 135–152

Lopez V, Nikolov A, Sabou M, Uren V, Motta E (2010) Scaling up question-answering to linked data. In: Knowledge Engineering and Knowledge Management by the Masses (EKAW-2010), Lisbon, Portugal, pp 193–210

Lozano-Tello A, Gómez-Pérez A (2004) Ontometric: a method to choose the appropriate ontology. J Database Manage 15(2):1–18

Maedche M, Staab S (2002) Measuring similarity between ontologies. In: 13th international conference on Knowledge Engineering and Knowledge Management (EKAW 2002), Siguenza, Spain, pp 251–263

Patel C, Supekar K, Lee Y, Park E (2003) OntoKhoj: a semantic web Portal for ontology searching, ranking, and classification. In: 5th international workshop on Web Information and Data Management (WIDM 2003). In conjunction with the 12th international conference on Information and Knowledge Management (CIKM 2003), New Orleans, LA, USA

Porzel R, Malaka R (2004) A task-based approach for ontology evaluation. In: Proceeding of ECAI 2004 workshop on ontology learning and population, Valencia, Spain

Poveda-Villalón M, Suárez-Figueroa MC, Gómez-Pérez A (2009) Common pitfalls in ontology development. In: 13th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2009), Sevilla, Spain, pp 91–100

Sabou M, Wroe C, Goble C, Mishne G (2005) Learning domain ontologies for web service descriptions: an experiment in bioinformatics. In: 14th international World Wide Web conference (WWW 2005), Chiba, Japan, pp 190–198

Sabou M, d'Aquin M, Motta E (2008) Exploring the semantic web as background knowledge for ontology matching. J Data Semant 11:156–190

Strasunskas D, Tomassen S (2008) Empirical insights on a value of ontology quality in ontology-driven web search. OnTheMove 2008 confederated international conferences (OTM 2008), Monterrey, Mexico, pp 1319–1337

Suárez-Figueroa MC, Gómez-Pérez A (2008) First attempt towards a standard glossary of ontology engineering terminology. In: 8th international conference on Terminology and Knowledge Engineering (TKE 2008), Copenhagen, Demark, pp 1–15

Suchanek FM, Kasneci G, Weikum G (2008) YAGO: a large ontology from Wikipedia and WordNet. J Web Semant 6(3):203–217

Tartir S, Arpinar I, Moore M, Sheth A, Aleman-Meza B (2005) OntoQA: metric-based ontology quality analysis. In: IEEE workshop on knowledge acquisition from distributed, autonomous, semantically heterogeneous data and knowledge sources, Houston, TX

Van Hage W, Isaac A, Aleksovski Z (2007). Sample evaluation of ontology matching systems. In: 5th international workshop on Evaluation of Ontologies and Ontology-based tools (EON 2007) Located at the 6th international Semantic Web Conference (ISWC 2007), Busan, Korea

Zablith F, d'Aquin M, Sabou M, Motta E (2010) Using ontological contexts to assess the relevance of statements in ontology evolution. In: 17th conference on Knowledge Engineering and Knowledge Management by the Masses (EKAW 2010), Lisbon, Portugal, pp 226–240

Chapter 10
Modularizing Ontologies

Mathieu d'Aquin

Abstract As large monolithic ontologies are difficult to handle and maintain, the activity of modularizing an ontology consists in identifying components (modules) of this ontology that can be considered separately while they are interlinked with other modules. The end benefit of modularizing an ontology can be, depending on the particular application or scenario, (a) to improve performance by enabling the distribution or targeted processing, (b) to facilitate the development and maintenance of the ontology by dividing it in loosely coupled, self-contained components or (c) to facilitate the reuse of parts of the ontology. In this chapter, we present a brief introduction to the field of ontology modularization. We detail the approach taken as a guideline to modularize existing ontologies and the tools available in order to carry out this activity.

10.1 Motivation

In complex domains such as medicine, ontologies can contain thousands of concepts. Examples of such large ontologies are the NCI (National Cancer Institute) Thesaurus[1] with about 27,500 and the Gene Ontology[2] with about 22,000 concepts. However, problems with large monolithical ontologies in terms of reusability, scalability, and maintenance have led to an increasing interest in techniques for dividing ontologies into sets of cohesive, self-contained modules; for extracting modules from ontologies relevant to a sub-domain or a task; as well as for

[1] http://ncit.nci.nih.gov/

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

M. d'Aquin (✉)
Knowledge Media Institute (KMi), The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK
e-mail: m.daquin@open.ac.uk

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_10, © Springer-Verlag Berlin Heidelberg 2012

combining and manipulating ontology modules. We observe however that there is no universal way to modularize an ontology and that the choice of a particular technique or approach should be guided by the requirements of the application or scenario relying on modularization.

In particular, ontologies that contain thousands of concepts cannot be created and maintained by a single person. The broad coverage of such large ontologies normally requires a team of experts. In many cases, these experts will be located in different organizations and will work on the same ontology in parallel. In other situations, large ontologies are mostly created to provide a standard model of a domain to be used by developers of individual solutions within that domain. While existing large ontologies often cover a complete domain, the providers of individual solutions are often only interested in a specific part of the overall domain.

Also, the nature of ontologies as reference models for a domain requires a high degree of quality of the respective model. Representing a consensus model, it is also important to have proposed models validated by different experts. In the case of large ontologies, it is often difficult, if not impossible, to understand the model as a whole.

On a technical level, very large ontologies cause serious scalability problems. The complexity of reasoning about ontologies is well known to be critical even for smaller ontologies. In the presence of ontologies like the NCI Thesaurus, not only reasoning engines but also modelling and visualization tools reach their limits. Currently, there is no modelling tool that can provide convenient modelling support for ontologies of the size of the NCI Thesaurus.

All these problems are a result of the fact that a large ontology is treated as a single monolithic model. Most problems would disappear if the overall model consists of a set of coherent modules about a certain sub-topic that can be used independently of the other modules while still containing information about its relation to these other modules.

In the next sections, we describe a general guideline to the modularization of ontologies and tools that can be used to support this activity. We identify three approaches which can be involved in realizing the modularization of an ontology: ontology partitioning, ontology module extraction and ontology module composition.

10.2 Ontology Modularization

We consider an ontology O as a set of axioms (sub-class, equivalence, instantiation, etc.) and the signature Sig(O) of an ontology O as the set of entity names occurring in the axioms of O, that is, its vocabulary. As described in the NeOn Glossary (Suárez-Figueroa 2010), ontology modularization refers to the activity of identifying one or more modules in an ontology. A module is considered to be a significant and self-contained sub-part of an ontology. Therefore, a module $M_i(O)$ of an ontology O is also a set of axioms (an ontology), with the minimal constraint that $Sig(M_i(O)) \subseteq Sig(O)$. Note that, while it may often be desirable, it is not always the case that $M_i(O) \subseteq O$.

10.2.1 Ontology Partitioning

The activity of partitioning an ontology consists of splitting up the set of axioms into a set of modules $\{M_1, \cdots, Mk\}$ such that each M_i is an ontology, and the union of all modules is semantically equivalent to the original ontology O (see Fig. 10.1). Note that some approaches being labelled as partitioning methods do not actually create partitions, as the resulting modules may overlap. There are several methods for ontology partitioning that have been developed for different purposes.

The method of MacCartney et al. (2003) 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 method of Cuenca Grau et al. (2005) 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 every module can be exploited independently of the others.

A tool that produces sparsely connected modules of reduced size was presented in Stuckenschmidt and Klein (2004). The goal of this method 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.

Later in this chapter, we describe a method for ontology partitioning based on enforcing good properties in the dependency graph between the resulting modules.

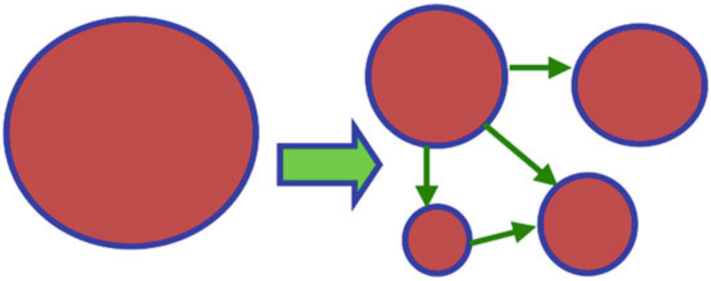


Fig. 10.1 Ontology partitioning

10.2.2 Ontology Module Extraction

Ontology module extraction consists in reducing an ontology to the sub-part, the module, that covers a particular sub-vocabulary. This activity has been called segmentation in Seidenberg and Rector (2006) and traversal view extraction in Noy and Musen (2004). More precisely, given an ontology O and a set SV $\subseteq$ Sig(O) of terms from the ontology, a module extraction mechanism returns a module M_{SV}, supposed to be the relevant part of O that covers the sub-vocabulary SV (Sig(M_{SV}) $\supseteq$ SV, see Fig. 10.2). Techniques for module extraction often rely on the so-called traversal method: 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 (Noy and Musen 2004), to be used in the Protégé environment. This method recursively follows the properties around a selected class of the ontology until a given distance is reached. The user can exclude certain properties in order to adapt the result to the needs of the application.

The mechanism presented in Seidenberg and Rector (2006) 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 Stuckenschmidt (2006), the author defines a viewpoint as being a sub-part 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, d'Aquin et al. (2006) define 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, 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.

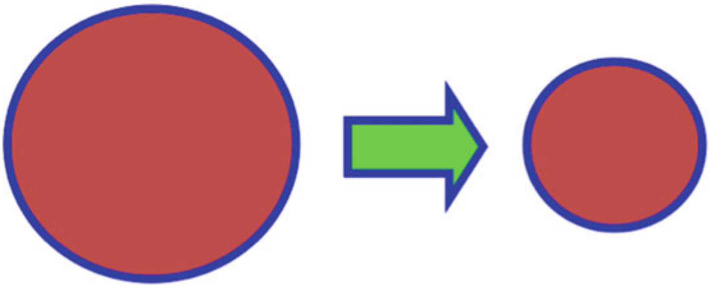


Fig. 10.2 Ontology module extraction

Finally, the technique described in Doran et al. (2007) is focussed 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.

One important issue related to ontology module extraction is that different scenarios and applications require different ways to modularize ontologies (d'Aquin et al. 2007b). To facilitate the selection, combination and adaptation of the various existing module extraction techniques, d'Aquin et al. (2007a) describe a parametric approach for module extraction. The principle is to describe module extraction techniques under a common framework that can be parameterized according to the modularization technique that is most 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 be 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 (d'Aquin et al. 2007a) 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 d'Aquin et al. (2007a) are at the basis of another approach for parametric modularization (Doran et al. 2008) which, instead of a graph transformation framework, employs 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 queries that represent modularization techniques. In the same line of ideas, we describe later in this chapter a tool that relies on a set of specific extraction operators that can be combined to extract modules from ontologies in a way that is customized to the application at hand.

10.2.3 Ontology Module Composition

In Wiederhold (1994), Wiederhold defines a simple ontology algebra, with the main purpose of facilitating ontology-based software composition. He defines a set of operators applying set-related operations on the entities described in the input ontologies and relying on equality mappings (=) between these entities. More precisely, the three operators are defined as shown in Table 10.1.

In the same line of ideas, but in a more formalized and sophisticated way, Melnik et al. (2004) describe a set of operators for model management, as defined in the Rondo platform (Melnik et al. 2003). The goal of model management is to facilitate and automatize the development of metadata-intensive applications by relying on the abstract and generic notion of model of the data, as well as on the idea of mappings between these models. An essential part of a platform for model

Table 10.1 Set of operators

Intersection(O_1, O2) → O	Creates an ontology O containing the common (mapped) entities in O_1 and O_2
Union(O_1, O_2) → O	Creates an ontology O containing the entities of O_1 and O_2 and merging the common ones
Difference(O_1, O_2) → O	Creates an ontology O containing only the entities in O_1 that are not mapped to entities in O_2

management is a set of operators to manipulate and combine these models and mappings. Melnik et al. (2004) focus on formalizing a core set of operators: Match, Compose, Merge, Extract, Difference and Confluence. Match is particular in this set. It takes two models as an input and returns a mapping between these models. It inherently does not have a formal semantics as it depends on the technique used for matching, as well as on the concrete formalism used to describe the models and mappings. Merge intuitively corresponds to the Union operator in Wiederhold (1994): it takes two models and a mapping and creates a new model that contains the information from both input models, relying on the input mapping. It also creates two mappings from the created model to the two original ones. Extract creates the sub-model of a model that is involved in a mapping, and Difference, the sub-model that is not involved in a mapping. Finally, compose and confluence are mapping manipulation operators, creating mappings by merging or composing other mappings.

Kaushik et al. (2006) define operators for combining ontologies created by different members of a community and written in RDF. This paper first provides a formalization of RDF to describe set-related operators such as Intersection, Union and Difference. It also adds other kinds of operators, such as the quotient of two ontologies O_1 and O_2 (collapsing O_2 into one entity and pointing all the properties of O_1 to entities of O_2 to this particular entity) and the product of two ontologies (inversely, extending the properties from O_1 to O_2 to all the entities of O_2). It is worth mentioning that such operators can be related to the ones of relational algebras used in relational database systems.

Note finally that the OWL tools[3] that are part of the KAON2[4] framework include operators such as Difference, Merge and Filter, working at the level of ontology axioms. For example, merge creates an ontology as the union of the axioms contained in the two input ontologies. The NeOn Toolkit plugin for ontology module composition presented in Sect. 10.5.3 relies on similar simple operators and is integrated with the other tools for module extraction and ontology partitioning.

[3] http://km.aifb.kit.edu/projects/owltools/

[4] http://kaon2.semanticweb.org/

10.3 A General Approach to Modularizing Ontologies

As we mentioned in Sect. 10.1, 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. Especially due to the large number of different techniques that can be used and combined to achieve these goals, there is a need for methodological guidelines to help ontology developers in selecting and applying the appropriate techniques for modularization, depending on the goal of modularization.

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.

Generalizing and clarifying the description above, we specify the definition of ontology modularization, as provided by the NeOn Glossary (Suárez-Figueroa 2010), as the activity that takes as an input an ontology and that has for goal to identify a set of modules for this ontology, effectively creating a modular version of it, for the purpose of supporting maintenance and reuse (see Fig. 10.3). Modularization offers a way to cut down potentially large ontologies into smaller, more manageable modules. It is generally realized by the ontology engineer or the ontology engineering team, preferably with the help of domain experts.

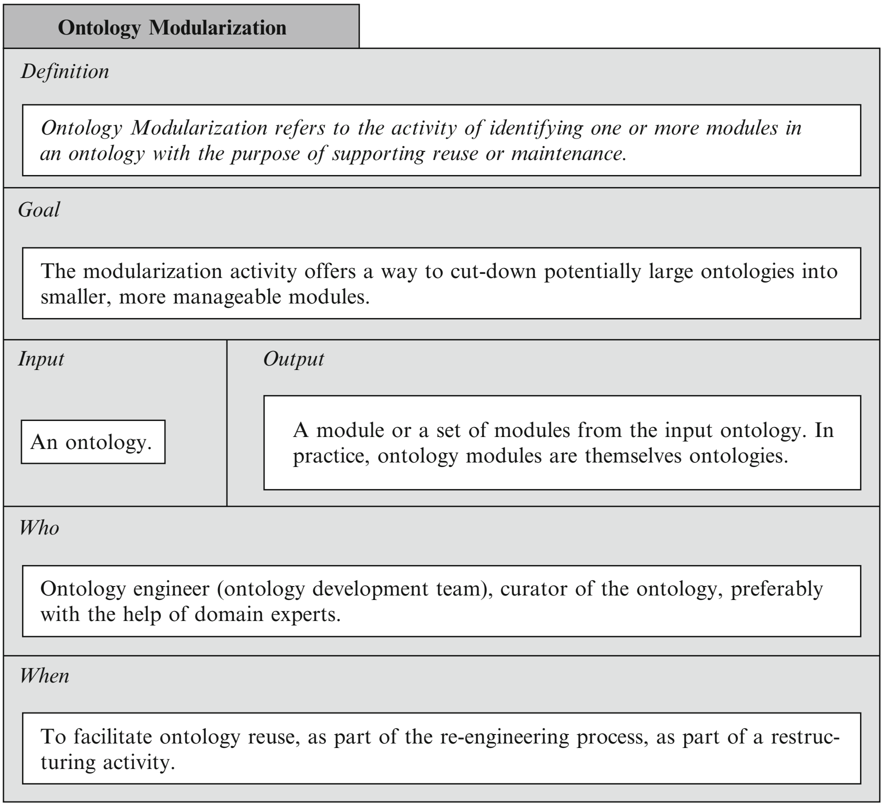
Figure 10.4 shows the workflow and the tasks for carrying out the ontology modularization activity. As can be seen in this figure, we see this activity as an iterative process, potentially combining different methods and techniques for module extraction and partitioning, and combining their results through the use of module composition operators.

Task 1. Identifying the Purpose of Modularization

As discussed earlier, the modularization of an ontology strongly depends on the application relying on the modularization and the context in which the ontology is developed. It is therefore crucial to start by identifying the reasons for modularizing the ontologies and the expected benefits, to guide the rest of the process.

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 Suntisrivaraporn et al. (2008) 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 reused
- *Customizing ontologies* by application developers to flexibly extract and combine modules relevant to a particular application or to provide different modules



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

To facilitate ontology reuse, as part of the re-engineering process, as part of a restructuring activity.

Fig. 10.3 Ontology modularization filling card

to different groups of users (see Lopez et al. (2009) 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. Selecting a Modularization Approach

As explained at the beginning of this chapter, there are two main approaches to obtain modules from ontologies: ontology partitioning and ontology module extraction. It is generally easy to decide which one to choose according to 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.

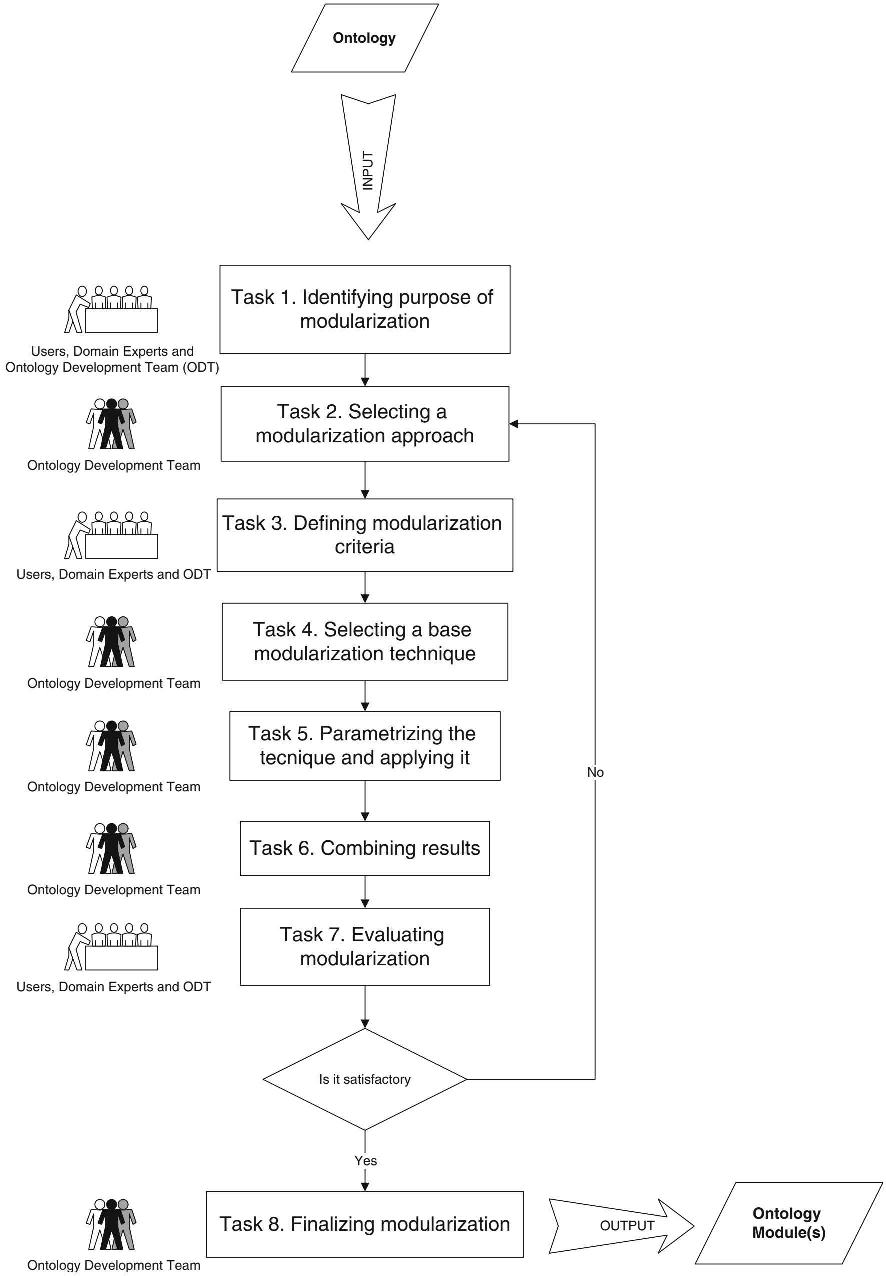

Ontology
INPUT
Task 1. Identifying purpose of modularization
Users, Domain Experts and Ontology Development Team (ODT)
Task 2. Selecting a modularization approach
Ontology Development Team
Task 3. Defining modularization criteria
Users, Domain Experts and ODT
Task 4. Selecting a base modularization technique
Ontology Development Team
Task 5. Parametrizing the tecnique and applying it
Ontology Development Team
Task 6. Combining results
Ontology Development Team
Task 7. Evaluating modularization
Users, Domain Experts and ODT
Is it satisfactory
No
Yes
Task 8. Finalizing modularization
Ontology Development Team
OUTPUT
Ontology Module(s)

Fig. 10.4 Tasks for modularizing ontologies

Of course, this needs to be considered in the context of the overall iterative process that constitutes ontology modularization. In general, when the purpose is to obtain a set of modules to cover the entire ontology, in a first iteration, partitioning should be considered. In subsequent iterations, intermediary modules might need to be further partitioned, or specific modules be extracted.

Task 3. Defining Modularization Criteria

The modularization criteria define the basic characteristics that the resulting modules should have, that is, what should go into a module. In d'Aquin et al. (2009), 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). The criteria to emphasize should be decided, depending on the purpose of modularization (as defined in Task 1). For example, if the goal is to improve the reasoning procedure, logical criteria should be favoured. In d'Aquin et al. (2009), we showed that 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 d'Aquin et al. (2009) 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. Selecting a Base Modularization Technique

As mentioned in previous sections, there is a great variety of techniques and tools for ontology modularization. In d'Aquin et al. (2007b, 2009), 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 d'Aquin et al. (2009) provide a description of the major techniques and experiments, demonstrating how they realize some possible criteria.

Task 5. Parametrizing the Technique and Applying It

Depending on the technique that has been selected by Task 4, 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. In such cases, 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. Combining 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, everytime a new (set of) module(s) is produced, it is necessary to integrate it – that is, 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 be merged.
- If modules from a previous iteration were too big because the employed technique did not 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 should be employed here to derive new modules from the results of partitioning or extraction techniques, or from different iterations for the process. The three common operators 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 or more modules have been produced that are complementary in the sense that one should be narrowed down so that it does not overlap with the other

Task 7. Evaluating Modularization

The evaluation of the result of the modularization (meaning the complete set of generated modules to be included in the modular ontology) is a crucial part of the iterative process. 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 are 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 obtained modularization actually realizes 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. Another example could be when the goal is to better support an application; in these cases, further guidelines about how to perform an application-based ontology evaluation can be found in Chap. 9.

There can be three 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 Sect. 10.4.

Task 8. Finalizing Modularization

Once the produced modularization is judged satisfactory, an additional step can be required for it to be deployed and exploited in an application. For example, it is 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.

10.4 Example

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 too many different modules for a particular management task
- The relations between modules, which should favour a well-structured organization in the dependency of the modules

Considering both criteria above, it is decided to apply the NeOn Toolkit plugin for ontology partitioning (see Sect. 10.5), which works on the dependency graph of modules and intends to provide good structures for this dependency graph (Task 4). The only parameter for this technique is the minimum size of a module (Task 5), which is chosen according to the size of the initial ontology. The resulting partition

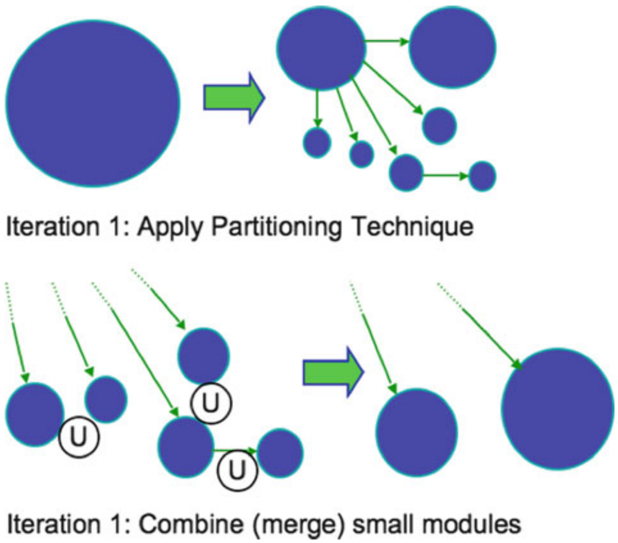

Iteration 1: Apply Partitioning Technique
U
U
U
Iteration 1: Combine (merge) small modules

Fig. 10.5 First iteration in our example modularization process

is described in Fig. 10.5. 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 plugin for module combination and employing the Union operator. This is depicted for our example in Fig. 10.5.

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

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 this particular topic (Task 3). A specific ontology module extraction technique is selected for this (Task 4) and used to generate relevant modules on the basis of a set of core terms defining the topic (Task 5). The result is depicted in Fig. 10.6. 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 plugin of the NeOn Toolkit (see Fig. 10.6). 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.

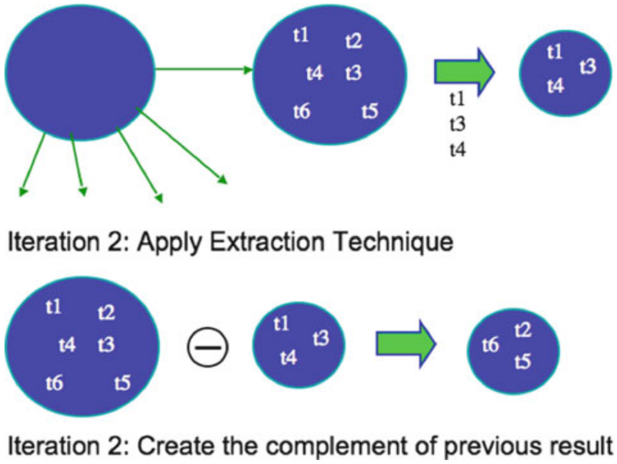

t1
t2
t4
t3
t6
t5
t1
t3
t4
t1
t3
t4
Iteration 2: Apply Extraction Technique
t1
t2
t4
t3
t6
t5
⊖
t1
t3
t4
t6
t2
t5
Iteration 2: Create the complement of previous result

Fig. 10.6 Second iteration in our example modularization process

10.5 Tool Support

The abstract example presented above provides an illustration of the overall activity of modularizing an existing ontology, using the iterative method we propose, based on different modularization approaches, and combining results from different techniques. Ideally, the tools necessary to achieve this activity of modularizing should be integrated within the same ontology engineering environment in which the ontologies are developed. Here, we present the tools integrated in the NeOn Toolkit in order to realize ontology partitioning, ontology module extraction and ontology module composition. Together with the NeOn Toolkit, these tools represent an integrated environment for creating and manipulating ontology modules.

10.5.1 Ontology Partitioning

Our method for ontology partitioning is based on basic requirements concerning the resulting modularization and its structure. We consider that the result of the partitioning process should not only be a bag of modules but should also provide the relations between them in terms of dependency. In addition, some good properties for this structure should be enforced in order to facilitate the manipulation and maintenance of the modularization.

As our approach is based on the dependency structure of modules, we need to define this relation of dependency. We consider a module M_1 to be dependent on a module M_2 if there is at least one entity in M_1 whose definition or description depends on at least one entity in M_2. The definition or the description of an entity

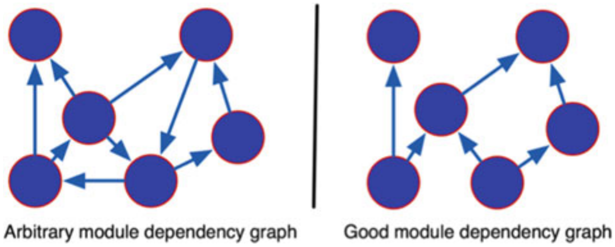

Arbitrary module dependency graph
Good module dependency graph

Fig. 10.7 Graphs illustrating dependency structures between ontology modules

A depends on an entity B whenever B participates in the axioms defining or describing A.

From this definition, we can see that if a module M_1 depends on a module M_2, it means that M_1 should import M_2. The main particularity of our approach is that we want the dependency structure of the resulting modularization to have good properties in order to be efficient in facilitating further engineering of the obtained modular ontology. In other terms, as shown in Fig. 10.7, we do not want this structure to be any arbitrary (directed) graph, but to respect two major rules:

1. *Rule 1 (no cycle)*: There should not be any cycle in the dependency graph of the resulting modularization. The rationale for this rule is that we are trying to reproduce the natural situation where modules would be reused. Creating bidirectional interdependencies between reused modules is a bad practice as it introduces additional difficulties in case of an update of one of the modules or when distributing modules (Parnas 1978).
2. *Rule 2 (no transitive dependency)*: If a module reuses another one, it should not directly or indirectly reuse a module on which the reused one is dependent. Indeed, when this situation arises, it means that the organization of modules into layers has not been enforced, so that a module is reusing other modules at different levels of the same branch of the dependency graph. Besides producing unnecessary redundancies in the dependency structure, this could also cause difficulties for the evolution and distribution of the module by creating 'concurrent propagation paths,' leading to the same module.

In addition, in order to ensure not only that the structure of the modularization respects good properties but also that individual modules are easy to manage and to handle, we add two rules on the characteristics of each module:

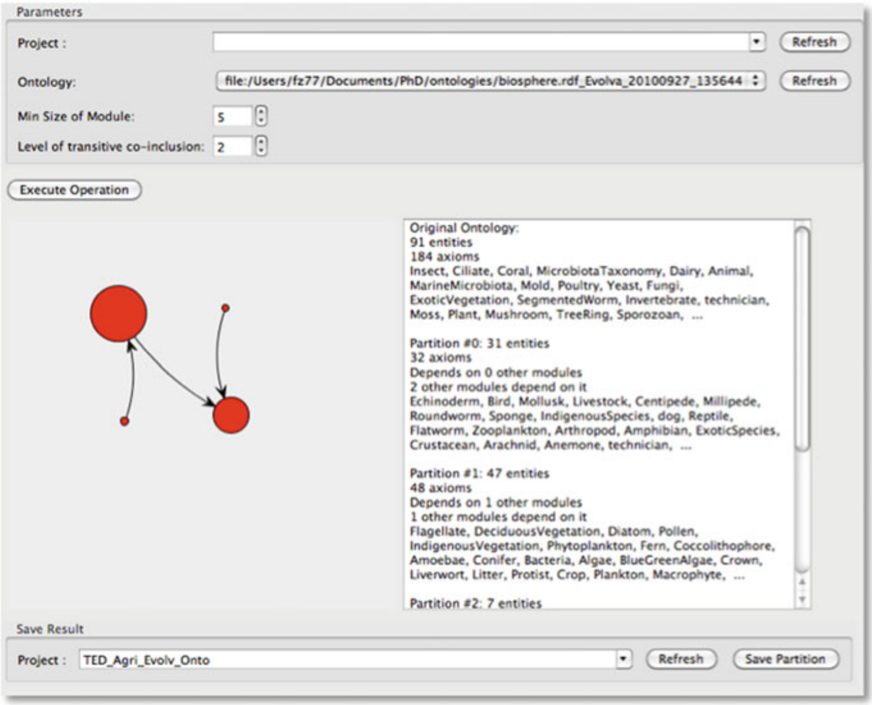
1. *Rule 3 (size of the modules)*: A module should not be smaller than a given threshold. Indeed, initial experiments have shown that applying only the two rules above can result in very small modules. Too small modules can be hard to manage, as it can result in having to consider too many different modules for a given task (e.g. update) (d'Aquin et al. 2007b). Note that, even if it could sometimes be useful, a rule based on the maximum size of a module would not be applicable, as it would contradict rules 1 or 2. In this case, it would be

recommended to use the extraction techniques described in Sect. 10.5.2 to reduce the size of the modules considered too big.

2. *Rule 4 (intra-connectedness)*: Entities within a module should be connected with each other. This is a very simple and natural rule to follow. Indeed, there is no reason for entities that are completely disconnected, directly or indirectly, to end up in the same module.

Having the above rules defined, our algorithm for partitioning ontologies is reasonably straightforward. It basically consists in starting from an initial modularization with as many modules as entities in the ontology. From this initial modularization, the algorithm iteratively enforces rules 1 and 2, merging modules when necessary. At the end of this step, a modularization that respects rules 1, 2 and 4 is obtained. The last task consists in merging modules that are too small according to the given threshold, ensuring that this merging ends up in modules that respect both rules 3 and 4.

Figure 10.8 shows a screenshot of the ontology partitioning plugin integrated with the NeOn Toolkit, which relies on the technique described above. Concretely, this plugin takes the form of a view which allows the user to select the ontology to modularize, specify the threshold for the minimum size of the modules, and execute the algorithm. The result of the algorithm is then presented as a graph, with each



Project :

Ontology: file:/Users/fz77/Documents/PhD/ontologies/biosphere.rdf_Evolva_20100927_135644

Min Size of Module: 5

Level of transitive co-inclusion: 2

```
Original Ontology:
91 entities
184 axioms
Insect, Ciliate, Coral, MicrobiotaTaxonomy, Dairy, Animal,
MarineMicrobiota, Mold, Poultry, Yeast, Fungi,
ExoticVegetation, SegmentedWorm, Invertebrate, technician,
Moss, Plant, Mushroom, TreeRing, Sporozoan, ...

Partition #0: 31 entities
32 axioms
Depends on 0 other modules
2 other modules depend on it
Echinoderm, Bird, Mollusk, Livestock, Centipede, Millipede,
Roundworm, Sponge, IndigenousSpecies, dog, Reptile,
Flatworm, Zooplankton, Arthropod, Amphibian, ExoticSpecies,
Crustacean, Arachnid, Anemone, technician, ...

Partition #1: 47 entities
48 axioms
Depends on 1 other modules
1 other modules depend on it
Flagellate, DeciduousVegetation, Diatom, Pollen,
IndigenousVegetation, Phytoplankton, Fern, Coccolithophore,
Amoebae, Conifer, Bacteria, Algae, BlueGreenAlgae, Crown,
Liverwort, Litter, Protist, Crop, Plankton, Macrophyte, ...

Partition #2: 7 entities
```

Project : TED_Agri_Evolv_Onto

Fig. 10.8 Screenshot of the ontology partitioning plugin of the NeOn Toolkit

node corresponding to a created module (details of the module are shown when selecting the corresponding node). The plugin allows the user to save and integrate to the current ontology project each module individually.

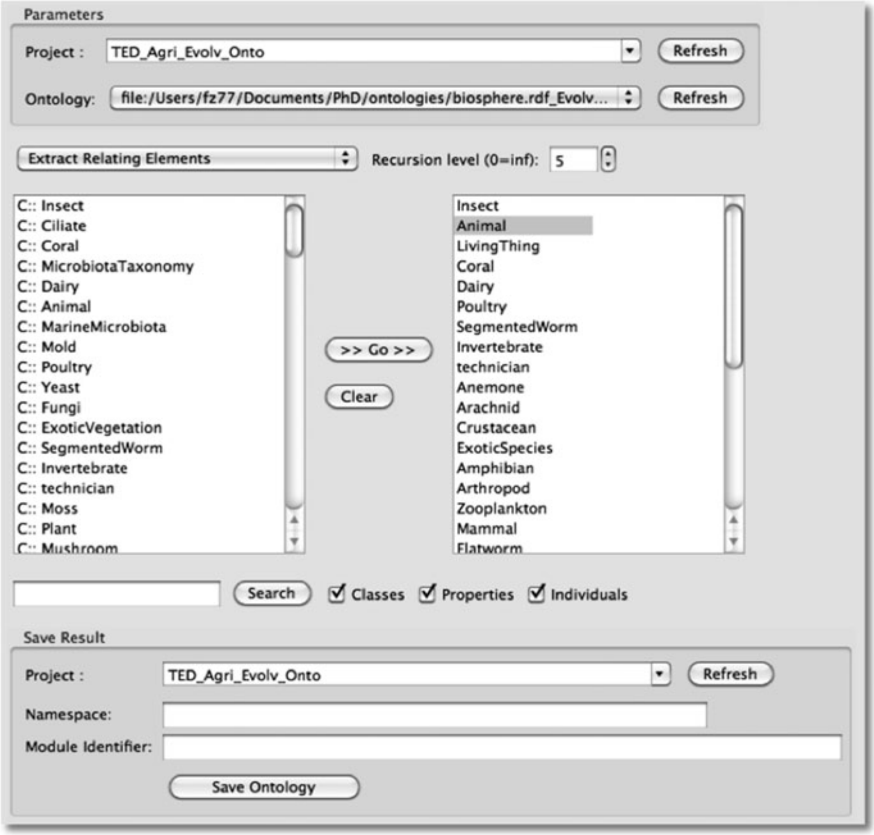
An interesting aspect of the implementation within the NeOn Toolkit is that it allows a very flexible and customizable modularization process. Indeed, it is possible to re-run the algorithm with different parameters, save only the modules that are relevant according to the ontology engineer, and use the module composition plugin presented below to manipulate and customize the modularization until a satisfactory, well-suited modularization is obtained.

10.5.2 Ontology Module Extraction

In d'Aquin et al. (2007a, and 2007b), we have shown through a number of experiments that extracting a module from an ontology is an ill-defined task: the criteria used to decide what should go in a module and what is a good, relevant module are highly dependent on the specificity of the application scenario. In other terms, there is no universal, generic module extraction approach. This appeared also very clearly in the different use cases described in d'Aquin et al. (2008), where different users, in different contexts, provided completely different perspectives about what should go in a module. In general, what appeared from these use cases is that:

1. Users have different, more or less well-defined ideas about what module extraction should do, varying from very elementary cases (e.g. extract a branch) to complex, abstract requirements (should extract everything that helps in interpreting a particular entity). Hence, each of the scenarios we encountered would require a different approach for module extraction.
2. Users want to keep in control of the way the module is created. It is required to support the parameterization of the module extraction for the user to be able to really 'choose' what goes into the module.

For these reasons, we implemented a plugin for the NeOn Toolkit to realize module extraction, providing an interactive and iterative approach to this activity. This plugin integrates a number of different 'operators' for module extraction, most of them being relatively elementary: based on an initial set of entities, extract the super-/sub-classes, entities they depend or that depend on them, common super-/ sub-classes, sub-/super-properties, all classes of instances, or all instances of classes. The interface for this plugin (Fig. 10.9) allows the user to easily combine these different elementary operators in an interactive way. An initial module can be created, using particular parameters (here the recursion level), obtaining an initial set of entities to be included. Then another operator can be used, on other entities and other parameters, to refine the module and extend it with other entities until an appropriate module is created. At any point of the process, previous operations can be undone and the module cleared.



Project : TED_Agri_Evolv_Onto

Ontology: file:/Users/fz77/Documents/PhD/ontologies/biosphere.rdf_Evolv...

Extract Relating Elements

Recursion level (0=inf): 5

C:: Insect
C:: Ciliate
C:: Coral
C:: MicrobiotaTaxonomy
C:: Dairy
C:: Animal
C:: MarineMicrobiota
C:: Mold
C:: Poultry
C:: Yeast
C:: Fungi
C:: ExoticVegetation
C:: SegmentedWorm
C:: Invertebrate
C:: technician
C:: Moss
C:: Plant
C:: Mushroom

Insect
Animal
LivingThing
Coral
Dairy
Poultry
SegmentedWorm
Invertebrate
technician
Anemone
Arachnid
Crustacean
ExoticSpecies
Amphibian
Arthropod
Zooplankton
Mammal
Flatworm

☑ Classes ☑ Properties ☑ Individuals

Project : TED_Agri_Evolv_Onto

Namespace:

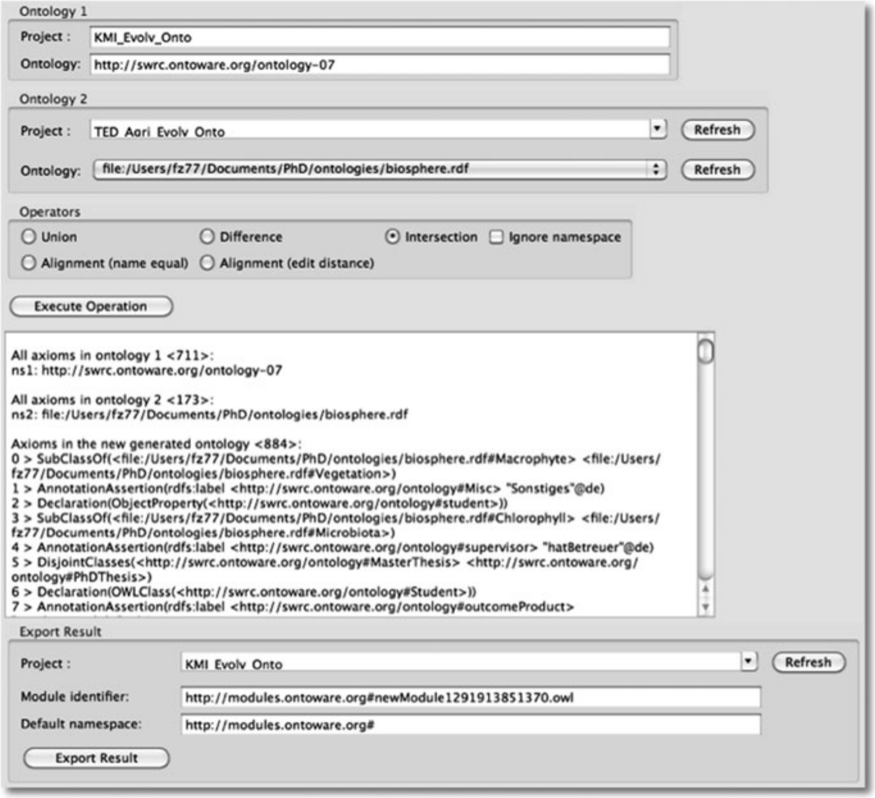
Module Identifier:

Fig. 10.9 Screenshot of the ontology module extraction plugin of the NeOn Toolkit

In addition, the plugin provides straightforward functions to facilitate the selection of the entities to consider for module extraction. This includes restricting the visualization to classes, properties or individuals and searching for entities matching a specific string. Once a module is created, it can simply be saved as part of the current ontology project and become itself processable as an ontology (module) to be composed or partitioned using the other modularization plugins.

10.5.3 Ontology Module Composition

A simple module algebra (including operators for Intersection, Union and Difference of module) is implemented in a dedicated plugin, which is realized as a new NeOn Toolkit view. As shown in Fig. 10.10, in this view, the user selects the two ontologies that serve as input for the operators. In the field between the two



Ontology 1

Project : KMI_Evolv_Onto

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

Ontology 2

Project : TED Agri Evolv Onto

Ontology: file:/Users/fz77/Documents/PhD/ontologies/biosphere.rdf

Operators

○ Union ○ Difference ● Intersection □ Ignore namespace

○ Alignment (name equal) ○ Alignment (edit distance)

```
All axioms in ontology 1 <711>:
ns1: http://swrc.ontoware.org/ontology-07

All axioms in ontology 2 <173>:
ns2: file:/Users/fz77/Documents/PhD/ontologies/biosphere.rdf

Axioms in the new generated ontology <884>:
0 > SubClassOf(<file:/Users/fz77/Documents/PhD/ontologies/biosphere.rdf#Macrophyte> <file:/Users/
fz77/Documents/PhD/ontologies/biosphere.rdf#Vegetation>)
1 > AnnotationAssertion(rdfs:label <http://swrc.ontoware.org/ontology#Misc> "Sonstiges"@de)
2 > Declaration(ObjectProperty(<http://swrc.ontoware.org/ontology#student>))
3 > SubClassOf(<file:/Users/fz77/Documents/PhD/ontologies/biosphere.rdf#Chlorophyll> <file:/Users/
fz77/Documents/PhD/ontologies/biosphere.rdf#Microbiota>)
4 > AnnotationAssertion(rdfs:label <http://swrc.ontoware.org/ontology#supervisor> "hatBetreuer"@de)
5 > DisjointClasses(<http://swrc.ontoware.org/ontology#MasterThesis> <http://swrc.ontoware.org/
ontology#PhDThesis>)
6 > Declaration(OWLClass(<http://swrc.ontoware.org/ontology#Student>))
7 > AnnotationAssertion(rdfs:label <http://swrc.ontoware.org/ontology#outcomeProduct>
```

Export Result

Project : KMI Evolv Onto

Module identifier: http://modules.ontoware.org#newModule1291913851370.owl

Default namespace: http://modules.ontoware.org#

Fig. 10.10 Screenshot of the ontology module composition plugin of the NeOn Toolkit

ontologies, the user selects the operator to be applied. In addition to the combination operators, the plugin also supports alignment as an operator, which allows relating modules via mappings. Depending on the operator chosen, the result will be either a new module (for Union, Difference, Intersection) or an alignment (for align).

Finally, the user can specify whether the application of the operators should be sensitive to differences in the namespace. If not, the operators only consider local names. This is for example relevant for the Difference operator applied to two versions of the same ontology – as often, the namespace changes from one version to another (and thus all elements in the ontology), a difference based on the fully qualified names would not be very meaningful.

10.6 Conclusion

In this chapter, we motivated and gave an overview of the activity of ontology modularization. We described a general approach for modularizing ontologies and the tools that have been developed for the NeOn Toolkit ontology engineering environment to support this approach. 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 described a simple 'abstract' example of ontology modularization. Further to this work, the empirical analysis of existing modular ontologies and of the process of modularizing existing ontologies could give us further insight into the broad notion of ontology modularity.

References

Cuenca Grau B, Parsia B, Sirin E, Kalyanpur A (2005) Automatic partitioning of owl ontologies using E-connections. In: Description logics, DL2005, Edinburgh

d'Aquin M, Sabou M, Motta E (2006) Modularization: a key for the dynamic selection of relevant knowledge components. In: Workshop on modular ontologies, WoMO 2006, Athens

d'Aquin M, Doran P, Motta E, Tamma V (2007a) Towards a parametric ontology modularization framework based on graph transformation. In: International workshop on modular ontologies, K-CAP 2007, Whistler

d'Aquin M, Schlicht A, Stuckenschmidt H, Sabou M (2007b) Ontology modularization for knowledge selection: experiments and evaluations. In: Database and expert systems applications, 18th international conference, DEXA 2007. Springer, Berlin/Heidelberg/New York

d'Aquin M, Haase P, Rudolph S, Euzenat J, Zimmermann A, Dzbor M, Iglesias M, Jacques Y, Caracciolo C, Buil Aranda C, Gomez, JM (2008) D1.1.3 NeOn formalisms for modularization: syntax, semantics, algebra. NeOn deliverable 1.1.3. NeOn project

d'Aquin M, Schlicht A, Stuckenschmidt H, Sabou M (2009) Criteria and evaluation for ontology modularization technique criteria and evaluation for ontology modularization technique. In: Stuckenschmidt H, Parent C, Spaccapietra S (eds) Modular ontologies: concepts, theories and techniques for knowledge modularization. Springer, Berlin/Heidelberg/New York

Doran P, Tamma V, Iannone L (2007) Ontology module extraction for ontology reuse: an ontology engineering perspective. In: Proceedings of the 2007 ACM CIKM international conference on information and knowledge management, Lisbon

Doran P, Palmisano I, Tamma V (2008) SOMET: algorithm and tool for SPARQL based ontology module extraction. In: International workshop on ontologies: reasoning and modularity (WORM-08), ESWC 2008, Tenerife

Kaushik S, Farkas C, Wijesekera D, Ammann P (2006) An algebra for composing ontologies. In: Formal ontology in information systems, FOIS 2006, Baltimore

Lopez V, Motta E, Dzbor M, d'Aquin M, Peroni S, Guidi D (2009) Final version of the question answering system. Deliverable 8.6 of the OpenKnowledge project

MacCartney B, McIlraith S, Amir E, Uribe TE (2003) Practical partition-based theorem proving for large knowledge bases. In: Proceedings of the international joint conference on artificial intelligence, IJCAI 2003, Acapulco

Melnik S, Rahm E, Bernstein PA (2003) Rondo: a programming platform for generic model management. In: Proceedings of the SIGMOD 2003, San Diego, pp 193–204

Melnik S, Bernstein PA, Halevy AY, Rahm E (2004) A semantics for model management operators. Microsoft technical report

Noy NF, Musen MA (2004) Specifying ontology views by traversal. In: Proceedings of the international semantic web conference, ISWC 2004, Hiroshima

Parnas DL (1978) Designing software for ease of extension and contraction. In: Proceedings of the 3rd international conference on software engineering

Seidenberg J, Rector A (2006) Web ontology segmentation: analysis, classification and use. In: Proceedings of the world wide web conference, WWW 2006, Edinburgh

Stuckenschmidt H (2006) Toward multi-viewpoint reasoning with OWL ontologies. In: Proceedings of the European semantic web conference, ESWC 2006, Budva

Stuckenschmidt H, Klein M (2004) Structure-based partitioning of large concept hierarchies. In: International semantic web conference, ISWC 2004, Hiroshima

Suárez-Figueroa MC (2010) NeOn Methodology for building ontology networks: specification, scheduling and reuse. PhD thesis, Universidad Politécnica de Madrid, España. Available at http://oa.upm.es/3879/

Suntisrivaraporn B, Guilin Q, Ji Q, Haase P (2008) A modularization-based approach to finding all justifications for OWL DL entailments. In: Asian semantic web conference, ASWC 2008, Bangkok

Wiederhold G (1994) An algebra for ontology composition. In: Monterey workshop on formal methods, Monterey

Chapter 11
Ontology Evolution

Raúl Palma, Fouad Zablith, Peter Haase, and Oscar Corcho

Abstract Ontologies are dynamic entities that evolve over time. There are several challenges associated with the management of ontology dynamics, from the adequate control of ontology changes to the identification and administration of ontology versions. Moreover, ontologies are increasingly becoming part of a network of complex relationships and dependencies, where they reuse and extend other ontologies, have associated metadata in order to ease sharing and reuse, are used to integrate heterogeneous knowledge bases, etc. Under these circumstances, a change in an ontology does not only affect the ontology itself but may also have consequences in all its related artifacts. In this chapter, we propose methodological guidelines for carrying out the ontology evolution activity. We target different scenarios, supporting users in the process of ontology evolution from a generic perspective and on how to use tools that semiautomatically assist them in discovering, evaluating, and integrating domain changes to evolve ontologies. To illustrate their applicability, we describe how such guidelines have been used in real example applications.

R. Palma (✉)
Poznan Supercomputing and Networking Center, ul Dabrowskiego 79a, 60-529 Poznan, Poland
e-mail: rpalma@man.poznan.pl

F. Zablith
Knowledge Media Institute (KMi), The Open University, Walton Hall, Milton Keynes
MK7 6AA, UK
e-mail: f.zablith@open.ac.uk

P. Haase
fluid Operations AG, Altrottstr. 31, 69190 Walldorf, Germany
e-mail: peter.haase@fluidops.com

O. Corcho
Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid,
Campus de Montegancedo sn, 28660 Boadilla del Monte, Madrid, Spain
e-mail: ocorcho@fi.upm.es

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_11, © Springer-Verlag Berlin Heidelberg 2012

11.1 Motivation

Ontologies are fundamental building blocks of the Semantic Web and are often used as the knowledge backbones of advanced information systems. As such, the growing use and application of ontologies in many different areas during the last years has led to an increasing interest of both researchers and industry in the construction of ontologies and the reuse of existing ones. Reusing existing ontologies instead of creating new ones from scratch has many benefits: it lowers the time and cost of developing new ontologies, avoids duplicate efforts, eases interoperability, etc. As a consequence, complex networks of ontologies are being created where each ontology may depend on several others and may also be related to other artifacts (e.g., individuals, mappings, applications, and metadata).

Nevertheless, this situation also brings about new issues. Ontologies (like many other system components) are dynamic entities. An ontology, defined as a formal, explicit specification of a shared conceptualization (Studer et al. 1998), may change whenever any of the elements of this definition changes. For instance, domains are not static or fixed: they may evolve when non-existing elements become part of the domain or when some elements become obsolete, among others. Additionally, ontologies need to be kept up to date in order to reflect the changes that affect the life cycle of the underlying systems (e.g., changes in the underlying data sets, need for new functionalities, etc.). A similar situation occurs with shared conceptualizations, which may change, for example, when the domain experts involved in modeling acquire additional knowledge about the domain. Finally, the formal specification may change because new ontology languages or new versions of the existing ones become available, for example.

The management of ontology dynamics raises many challenges such as the identification and administration of different ontology versions or the flow control of ontology changes (i.e., when and how an ontology can change). Moreover, dealing with ontology changes involves the execution of many related tasks. Most of these tasks are already identified in the context of the ontology evolution process, defined in (Stojanovic 2004) as the timely adaptation of an ontology to the arisen changes and the consistent management of these changes. For example, 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 distributed nature of a network of ontologies where complex relations can exist between ontologies and other artifacts demands the necessity to propagate ontology changes to the distributed ontology-dependent artifacts (e.g., related ontologies, ontology individuals, mappings, and metadata). For instance, a change in a wine ontology (e.g., add a new class for a type of wine) may require one or more updates in its related metadata (e.g., increase the number of classes by one, add an additional key class, add an additional contributor, and update the date of the last modification) or its mappings to other similar ontologies (e.g., create a new correspondence between the new class and another class representing the same type of wine in another ontology). Moreover, the ontology and its related artifacts may be distributed in different places across the web.

While it seems necessary to apply the ontology evolution 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. 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.

It is worth noting that both ontology evolution and ontology versioning deal with the management of ontology changes. However, they differ in their focus: ontology evolution focuses on the modification of an ontology, possibly preserving its consistency, whereas ontology versioning focuses on creating and managing different versions of the ontology.

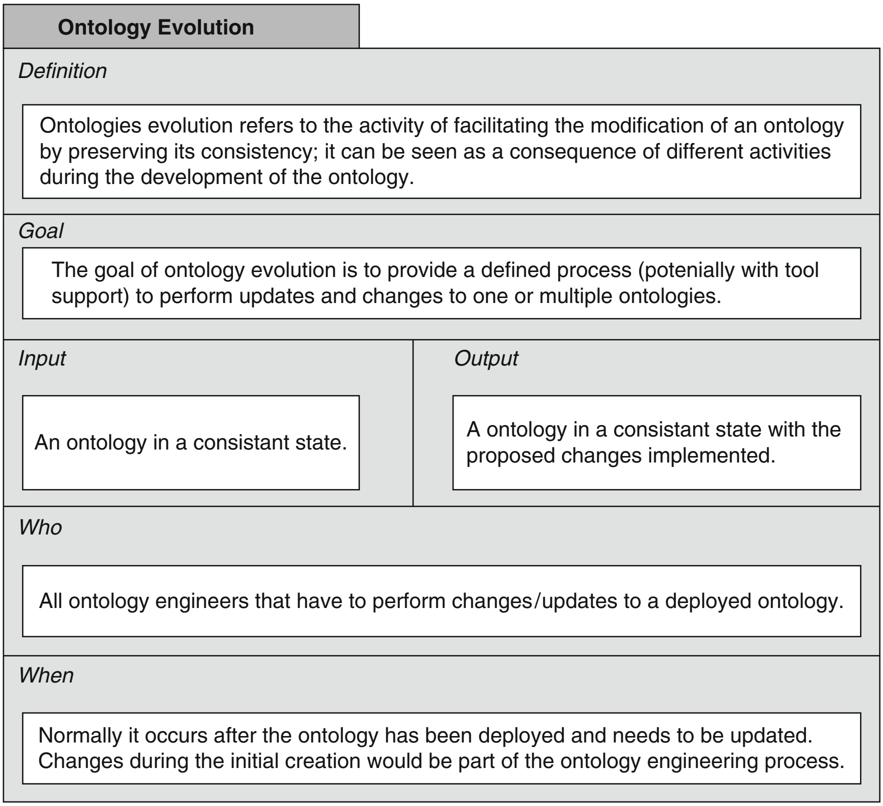
We argue that in order to provide a comprehensive support for ontology evolution, targeted at users and ontology engineers, we need two types of guidelines: one that guides users in the process of ontology evolution from a generic perspective and another that provides guidelines on how to use tools that semiautomatically assist users in discovering, evaluating, and integrating domain changes to evolve ontologies.

The remainder of this chapter is organized as follows: First, we introduce high-level guidelines for carrying out the ontology evolution activity. This would give an overall picture of the required tasks and possible options to handle each one, supported by an example in the fishery domain of FAO. Second, we provide guidelines for how to support users in exploiting and customizing tools that support users in evolving ontologies from external domain data using semiautomatic techniques. This is also supported by an applied example in the academic domain.

11.2 Guidelines for Ontology Evolution

In this section, we present the guidelines set out to help ontology developers in the ontology evolution activity. Such guidelines have been created in the context of the NeOn Methodology for building ontology networks. This methodology takes into account the existence of multiple ontologies in ontology networks, the collaborative ontology development, the dynamic dimension, and the reuse and reengineering of knowledge-aware resources.

According to the NeOn Glossary of Processes and Activities (Suárez-Figueroa and Gómez-Pérez 2008), ontology evolution refers to the activity of facilitating the modification of an ontology by preserving its consistency; it can be seen as a consequence of different activities during the development of the ontology.



Ontology Evolution

Definition

Ontologies evolution refers to the activity of facilitating the modification of an ontology by preserving its consistency; it 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 (potenially with tool support) to perform updates and changes to one or multiple ontologies.

Input

An ontology in a consistant state.

Output

A ontology in a consistant state with the proposed changes implemented.

Who

All ontology engineers that have to perform changes/updates to a deployed ontology.

When

Normally it occurs after the ontology has been deployed and needs to be updated. Changes during the initial creation would be part of the ontology engineering process.

Fig. 11.1 Ontology evolution filling card

Thus, in the framework of the NeOn Methodology we propose the filling card for the ontology evolution, presented in Fig. 11.1, which includes the definition, goal, input, output, who carries out the activity, and when the activity should be carried out.

11.2.1 Ontology Evolution Tasks

Figure 11.2 illustrates the methodological guidelines for carrying out the ontology evolution activity, showing the main tasks involved, their inputs, outputs, and actors. The tasks shown in the figure are explained below. They are based on the generic activities discussed in (Bennett and Rajlich 2000) for the process of making changes to any type of artifact that is subject to changes, customized to the case of ontology evolution (e.g., Leenheer and Mens 2007) in the context of ontology networks.

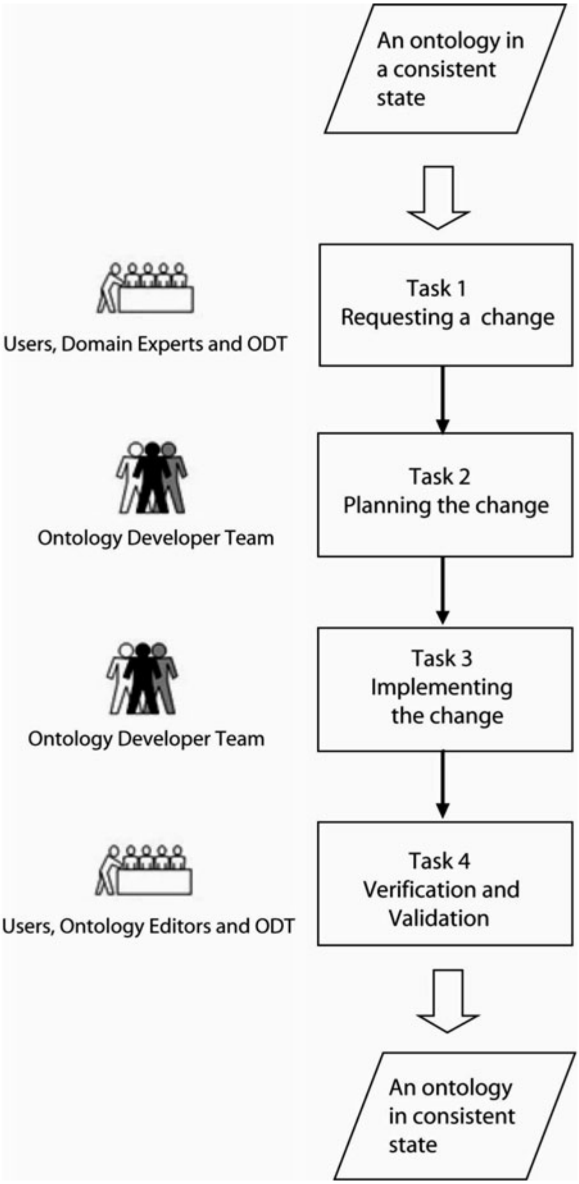

An ontology in a consistent state
Users, Domain Experts and ODT
Task 1 Requesting a change
Ontology Developer Team
Task 2 Planning the change
Ontology Developer Team
Task 3 Implementing the change
Users, Ontology Editors and ODT
Task 4 Verification and Validation
An ontology in consistent state

Fig. 11.2 Tasks for ontology evolution activity

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, see Chap. 17) 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 artifacts. This requirement is also in accordance to existing ontology evolution approaches (e.g., Stojanovic (2004)). 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 correctly models 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 preprocessing 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 (Stojanovic 2004), change discovery is distinguished into top-down and bottom-up change discovery. Top-down (deductive/explicit) changes are often 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 typically 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., Stojanovic 2004; Klein and Noy 2003; Noy et al. 2006; Palma et al. 2009). 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, a layered approach for the representation of ontology changes was proposed (Palma et al. 2007, 2009), 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, this task may include the use of a well-defined process (a workflow) for coordinating change proposals (see Palma et al. 2008a, b). This process is responsible for determining who (depending on the user permissions) can do what (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).

Tool Support in the NeOn Toolkit

- RaDON plugin is an ontology diagnosis and repair tool that can be used before starting the evolution activity, i.e., before applying changes.
- Tools supporting the request/discovery of changes:
  - The workflow feature supports the process that coordinates the proposal of changes in a collaborative environment. It supports a top-down/explicit discovery method, i.e., when changes are requested by users/developers.
  - The Evolva plugin supports the discovery of changes from external data sources (e.g., text, folksonomies, or RSS feeds). Changes are integrated and evaluated by relying on background knowledge such as online ontologies. In the next section, we present in details the guidelines for how to exploit such tool to apply the identified changes on the ontology and produce a new ontology version.

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 (Arnold 1996; Bohner 1996). As we noted in the introduction, ontologies may depend on several others and may also be related to other artifacts (e.g., individuals, 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 artifacts should be presented to the ontology engineer (Plessers 2006).

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 artifact (Plessers 2006).

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.

Tool Support in the NeOn Toolkit

- The NeOn Toolkit provides simple support when deciding whether to make 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, which permits 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 (Chikofsky and Cross 1990). 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 (Proper and Halpin 2004), the authors distinguish three reasons to apply transformation: (1) to select an alternative conceptual schema which is regarded as a better representation of the domain, (2) to enrich the schema with derivable parts creating diverse alternative views on the same conceptual schema as a part of the original schema, and (3) to optimize 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, the inconsistencies have to be identified and resolved, possibly using some tools as it was mentioned in the introduction. In the literature, this activity has been introduced in (Stojanovic 2004) as the semantics of the change (originally proposed in the area of data schema evolution in Banerjee et al. 1987) 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, the author introduces 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 can be found. For instance, in (Klein and Fensel 2001), the authors propose to use an URI for ontology identification with a two-level numbering scheme: major 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 it 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 Klein and Fensel 2001). For instance, in (Palma et al. 2008c), 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 artifacts (if the ontology engineer decided to do it in the previous task based on the analysis of the cost and impact). In (Stojanovic 2004), the author discusses the propagation of changes to dependent ontologies, individuals, and applications and elaborates on the propagation to dependent ontologies using a combination of push and pull mechanism. For the propagation to ontology individuals, several mechanisms can be applied from the research in the area of databases. For instance, in (Parsia et al. 2005), the authors discuss how changes can be propagated to the individuals 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, the propagation of changes has also been considered to (1) distributed copies of the same ontology and (2) ontology metadata (Palma 2009; Palma et al. 2007, 2008b).

Tool Support in the NeOn Toolkit

- NeOn Toolkit ontology editor allows the manual application of changes to ontologies.
- The change capturing plugin supports the logging of changes automatically from the NeOn ontology editor. It also supports the application of logs generated by other systems. Additionally, it is also in charge of propagating changes to the distributed copies of the same ontology.
- RaDON plugin 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 artifacts related to the ontology (as mentioned above) to ensure that they were not changed in a wrong way or they have an unexpected behavior. The verification and validation step can include the following activities:

- 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 localizing and repairing errors found during the verification or testing (usually performed by an ontology engineer) (for example Haase et al. (2006))
- Quality assurance, which typically concerns non-functional qualities, like reusability, adaptability, interoperability, etc.
- Justification of the changes, (for example Stojanovic 2004)
- Relevance of the changes with respect to the ontology under evolution (Zablith et al. 2010)

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 activities (e.g., approve/reject) derived from the well-defined process (e.g., workflow) that coordinates the change proposals (see Palma et al. 2008a, b). 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.

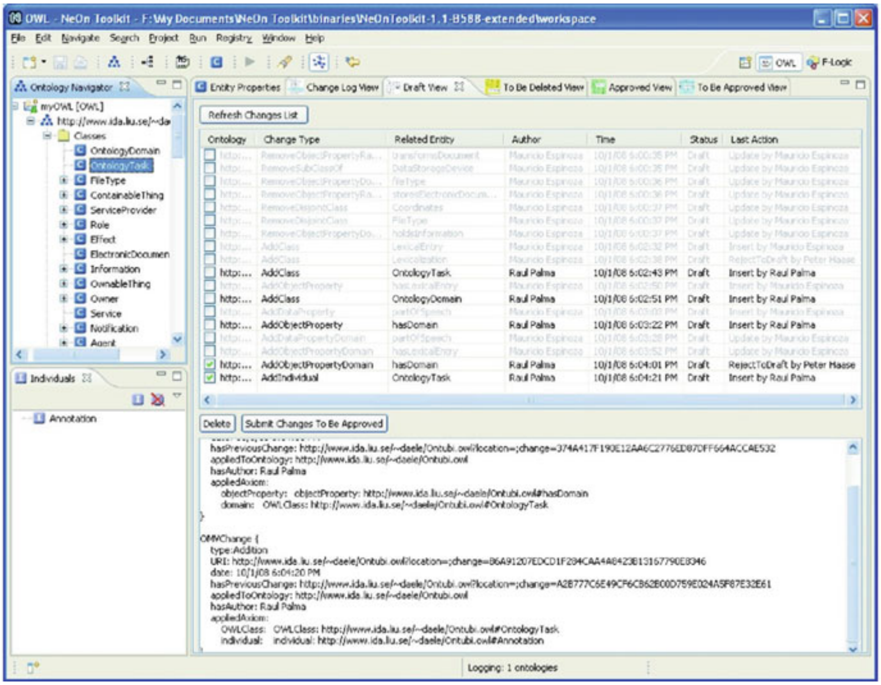
Tool Support in the NeOn Toolkit

- The Cicero plugin supports the justification of changes.
- The workflow feature supports the refining of activities (see Fig. 11.3).
- The Evolva plugin checks the relevance of a change with respect to an ontology by relying on the analysis of ontological contexts and a set of identified relevance patterns supported by a confidence-based ranking (Zablith et al. 2010).

Working with Networked Ontologies

The NeOn project deals with networks of ontologies and networked ontologies (Haase et al. 2006), defined as a collection of ontologies related together via a variety of different relationships such as mapping, modularization, version, and dependency relationships.

Hence, it is worth remarking that the process described above can be applied to networked ontologies since such a process takes into account the existing ontology dependencies with other related artifacts, such as individuals, mappings, applications, and metadata, as we noted in each step. In a nutshell, such dependencies are first 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 artifacts 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 artifacts to ensure that



	Ontology	Change Type	Related Entity	Author	Time	Status	Last Action
☐	http:...	RemoveObjectPropertyRa...	transformsDocument	Mauricio Espinoza	10/1/08 6:00:35 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveSubClassOf	DataStorageDevice	Mauricio Espinoza	10/1/08 6:00:35 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveObjectPropertyDo...	fileType	Mauricio Espinoza	10/1/08 6:00:36 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveObjectPropertyRa...	storesElectronicDocum...	Mauricio Espinoza	10/1/08 6:00:36 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveDisjointClass	Coordinates	Mauricio Espinoza	10/1/08 6:00:37 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveDisjointClass	FileType	Mauricio Espinoza	10/1/08 6:00:37 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveObjectPropertyDo...	holdsInformation	Mauricio Espinoza	10/1/08 6:00:37 PM	Draft	Update by Mauricio Espinoza
☐	http:...	AddClass	LexicalEntry	Mauricio Espinoza	10/1/08 6:02:32 PM	Draft	Insert by Mauricio Espinoza
☐	http:...	AddClass	Lexicalization	Mauricio Espinoza	10/1/08 6:02:38 PM	Draft	RejectToDraft by Peter Haase
☐	http:...	AddClass	OntologyTask	Raul Palma	10/1/08 6:02:43 PM	Draft	Insert by Raul Palma
☐	http:...	AddObjectProperty	hasLexicalEntry	Mauricio Espinoza	10/1/08 6:02:50 PM	Draft	Insert by Mauricio Espinoza
☐	http:...	AddClass	OntologyDomain	Raul Palma	10/1/08 6:02:51 PM	Draft	Insert by Raul Palma
☐	http:...	AddDataProperty	partOfSpeech	Mauricio Espinoza	10/1/08 6:03:03 PM	Draft	Insert by Mauricio Espinoza
☐	http:...	AddObjectProperty	hasDomain	Raul Palma	10/1/08 6:03:22 PM	Draft	Insert by Raul Palma
☐	http:...	AddDataPropertyDomain	partOfSpeech	Mauricio Espinoza	10/1/08 6:03:28 PM	Draft	Update by Mauricio Espinoza
☐	http:...	AddObjectPropertyDomain	hasLexicalEntry	Mauricio Espinoza	10/1/08 6:03:52 PM	Draft	Update by Mauricio Espinoza
☑	http:...	AddObjectPropertyDomain	hasDomain	Raul Palma	10/1/08 6:04:01 PM	Draft	RejectToDraft by Peter Haase
☑	http:...	AddIndividual	OntologyTask	Raul Palma	10/1/08 6:04:21 PM	Draft	Insert by Raul Palma

```
  hasPreviousChange: http://www.ida.liu.se/~daele/Ontubi.owl?location=;change=374A417F190E12AA6C2776ED87DFF664ACCAE532
  appliedToOntology: http://www.ida.liu.se/~daele/Ontubi.owl
  hasAuthor: Raul Palma
  appliedAxiom:
    objectProperty:  objectProperty: http://www.ida.liu.se/~daele/Ontubi.owl#hasDomain
    domain:   OWLClass: http://www.ida.liu.se/~daele/Ontubi.owl#OntologyTask
}

OMVChange {
  type:Addition
  URI: http://www.ida.liu.se/~daele/Ontubi.owl?location=;change=B6A91207EDCD1F284CAA4A8423B13167790E8346
  date: 10/1/08 6:04:20 PM
  hasPreviousChange: http://www.ida.liu.se/~daele/Ontubi.owl?location=;change=A2B777C6E49CF6CB62B00D759E024A5F87E32E61
  appliedToOntology: http://www.ida.liu.se/~daele/Ontubi.owl
  hasAuthor: Raul Palma
  appliedAxiom:
    OWLClass:   OWLClass: http://www.ida.liu.se/~daele/Ontubi.owl#OntologyTask
    individual:   individual: http://www.ida.liu.se/~daele/Ontubi.owl#Annotation
```

Fig. 11.3 Collaborative editorial workflow support in NeOn Toolkit

the whole network of ontologies is behaving as expected, i.e., it is consistent. So, any conflict that may arise can be caught at an earlier stage of the ontology evolution process, affecting, for instance, the decision of whether or not a change should be implemented.

11.2.2 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 stable version of one 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 Palma (2009).

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[1] (v1.0 at the time the experiment was conducted) – the ontology models a taxonomic classification of biological entities, including classes such as Family, Group, Order, and Species. In this case, changes were discovered using a top-down/explicit method as the knowledge came directly from human experts. A total of 34 changes were requested using real information according to the experts (see Palma 2009). Examples of those changes are: to add Individual 31005–10001 (Species); to add Individual 31005–10001 DataProperty hasNameScientific, value: Pterodroma wrong macroptera, type: string; to add Root Class Speciation; and to add ObjectProperty hasScientificNameAuthor.

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

Each of the proposed changes was represented as an individual of the change ontology proposed in Palma et al. (2009) – 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 change capturing plugin of NeOn Toolkit.

Furthermore, in this scenario, the ontology editors were following a well-defined process (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 (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 did not perform 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, no restructuring of the change(s) was necessary, 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 plugin of NeOn Toolkit) took care of logging automatically all of the proposed changes (change logging), maintaining the chronological history of the events.

[1] Available at http://aims.fao.org/en/website/Fisheries-ontologies-/sub2#species

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 plugin 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 plugins 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 plugins of the NeOn Toolkit. Finally, at the moment of the experiment, there was no support for the justification of changes.

11.3 Guidelines for Exploiting Tools in Ontology Evolution

In this section, we propose a methodological guideline for supporting users in identifying new and relevant domain changes from external data sources. Such guidelines aim to facilitate the process on evolving ontologies to reflect the latest changes in certain domains by analyzing various data sources. This guideline complements the tool-based support provided by the Evolva ontology evolution framework (discussed next), with concrete guidance on how to realize the various tasks of the evolution activity, using semiautomatic techniques in an efficient way.

11.3.1 The Evolva Framework

The Evolva ontology evolution framework (Zablith 2009) relies on the hypothesis that various forms of data corpus (texts, folksonomies, RSS feeds, etc.) can be used to detect the need for an evolution and initiate it (see Fig. 11.4). 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 sources, which can be supplied by online ontologies, lexical resources (e.g., WordNet, Fellbaum 1998), or the web. An additional use of background knowledge comes at the level of online ontologies used to assess the relevance of statements with respect to the ontology in focus.

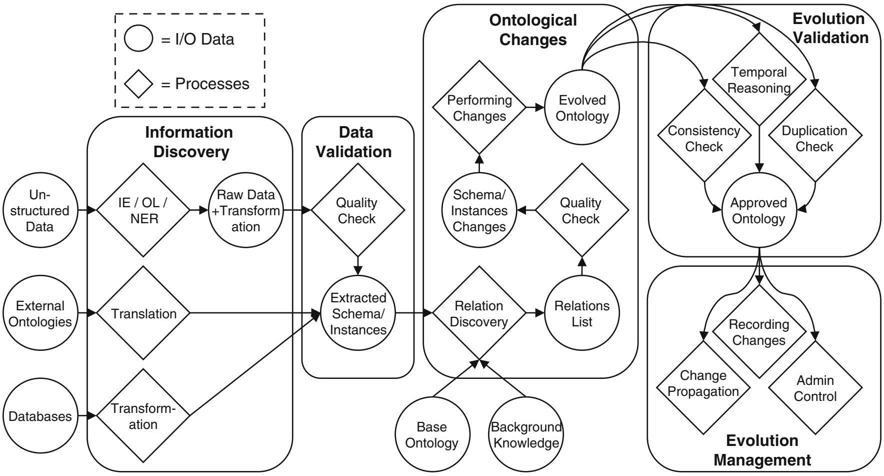


Fig. 11.4 Evolva's ontology evolution framework

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 of 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.

11.3.2 Tasks

The tasks for performing a semiautomatic ontology evolution can be seen in the workflow shown in Fig. 11.5. In this context, the starting point is an existing ontology (depicted as V1 in Fig. 11.5 and *base ontology* in Fig. 11.4, see Sect. 11.3.1), which the user aims to evolve based on available domain data sources. The selection of the appropriate sources from which new ontology entities are identified depends on the evolution use case and the availability of such sources in the domain in focus. In the rest of this section, we present the details of the tasks involved in semiautomatically evolving the ontology.

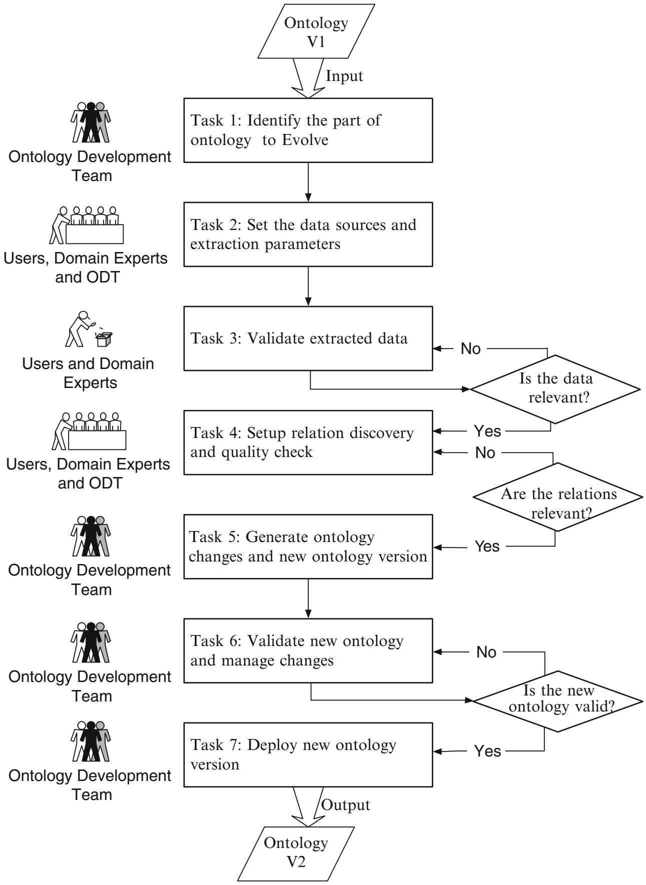

Ontology V1
Input
Ontology Development Team
Task 1: Identify the part of ontology to Evolve
Users, Domain Experts and ODT
Task 2: Set the data sources and extraction parameters
Users and Domain Experts
Task 3: Validate extracted data
No
Is the data relevant?
Users, Domain Experts and ODT
Task 4: Setup 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
Output
Ontology V2

Fig. 11.5 Tasks for ontology evolution supported by semiautomatic tools

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, individuals, or both, the extraction should be customized accordingly. For example, in the case of schema evolution, the user may choose to extract concepts for the data sources, without dealing with individuals. While in the case of individuals, the evolution process could omit the extraction of schema elements. Choosing between schema and individuals evolution could be biased by the ontology functionalities and domain nature, i.e., 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 is 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 settings related to the automatic relevance checking, or specify the maximum depth to explore. In addition to the supplied automatic quality checking methods, for example, in terms or relevance, domain experts should additionally 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.

11.3.3 Example

In this part, we highlight an example of ontology evolution scenario using the Evolva plugin for the NeOn Toolkit, following the guidelines presented in the previous section. We run our example in an environment where the NeOn Toolkit and Evolva plugin[2] are operational.

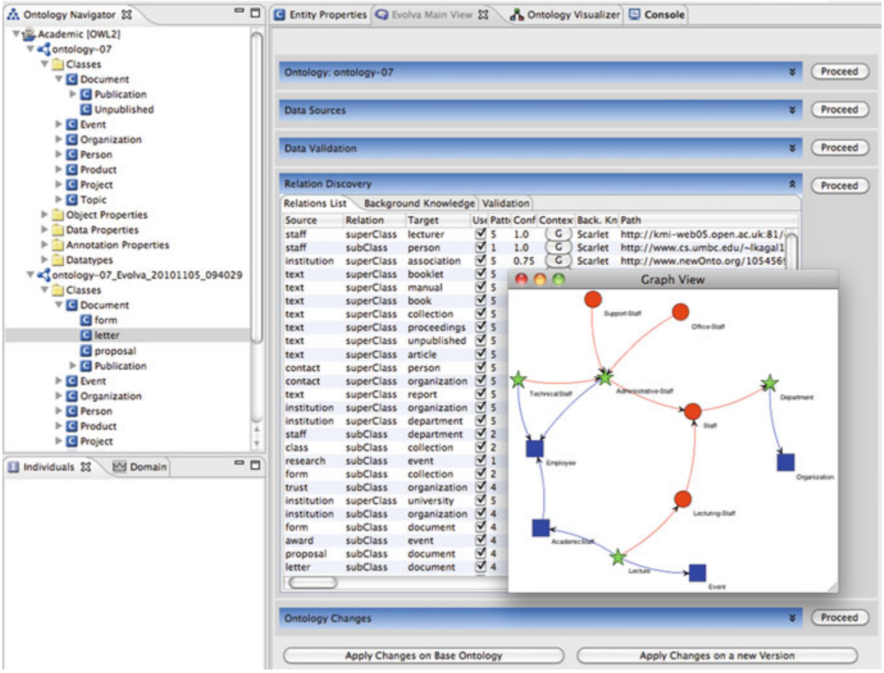
Consider the case of evolving the latest version of the SWRC ontology[3] in the academic context. We first load the ontology in the NeOn Toolkit and start Evolva. In our simple case where the ontology has a limited number of concepts and time is not an issue, we choose to evolve all the ontology (Task 1). This choice can be specified in the first step of the process (called *Ontology*) in Fig. 11.6.

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). This is implemented in *Data Sources* step of Fig. 11.6. A relevant source of information we found was on the Leverhulme website[4] 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 would not 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. Evolva includes extraction of

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

[3] The SWRC ontology can be downloaded from: http://ontoware.org/swrc/swrc/SWRCOWL/swrc_updated_v0.7.1.owl

[4] http://www.leverhulme.ac.uk/



Source	Relation	Target	Use	Patt	Conf	Contex	Back. Kn	Path
staff	superClass	lecturer	☑	5	1.0	G	Scarlet	http://kmi-web05.open.ac.uk:81/
staff	subClass	person	☑	1	1.0	G	Scarlet	http://www.cs.umbc.edu/~lkagal1
institution	superClass	association	☑	5	0.75	G	Scarlet	http://www.newOnto.org/105456
text	superClass	booklet	☑	5				
text	superClass	manual	☑	5				
text	superClass	book	☑	5				
text	superClass	collection	☑	5				
text	superClass	proceedings	☑	5				
text	superClass	unpublished	☑	5				
text	superClass	article	☑	5				
contact	superClass	person	☑	5				
contact	superClass	organization	☑	5				
text	superClass	report	☑	5				
institution	superClass	organization	☑	5				
institution	superClass	department	☑	5				
staff	subClass	department	☑	2				
class	subClass	collection	☑	2				
research	subClass	event	☑	1				
form	subClass	collection	☑	2				
trust	subClass	organization	☑	4				
institution	superClass	university	☑	5				
institution	subClass	organization	☑	4				
form	subClass	document	☑	4				
award	subClass	event	☑	4				
proposal	subClass	document	☑	4				
letter	subClass	document	☑	4				

Fig. 11.6 Screenshot of the Evolva plugin

concepts from text documents and RSS feeds, as well as a list of raw terms. The validation parameters incorporate term existence checking feature (based on a similarity value) and a term length checker for removing terms under a specified 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 rerun 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), implemented in the *Data Validation* step in Fig. 11.6. 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 extent, 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 online ontologies using Scarlet (Sabou et al. 2008) (a Semantic Web–based relation discovery engine) and WordNet.

Evolva automatically harvests the chosen background knowledge sources and identifies how extracted concepts should be integrated in the ontology. If needed, Evolva also provides the option to discover relations between new concepts, before being integrated in the ontology. This has been implemented in the *Relation Discovery* step in Fig. 11.6. To illustrate how background knowledge sources integrate new concepts, *Applicant* and *Website* are two concepts extracted from the Leverhulme text document. WordNet links *Applicant* as a *subclass* of *Person*, an existing concept in the SWRC ontology, while online ontologies link *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 rerun the process again.

After the relations linking new concepts to existing concepts in the ontology, Evolva relies on online ontologies from where the relation is identified to assess the relevance of the relation with respect to the ontology. Using identified relevance patterns, with pattern-specific confidence, relations are returned ranked to the user with the highly relevant relations placed on top (Zablith et al. 2010). The user is supplied with a customizable graphical visualization of the ontological contexts (shown on top of Fig. 11.6), as a validation of the relevance calculation. In addition to the visualization parameters, it is possible to change the weight of relevance patterns, hence affecting the overall ranking of relations.

Once all relations are approved and relevant, they are used to generate the ontology changes (last step in Fig. 11.6). 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, it is possible to apply the changes on the source ontology, or a new version with the evolution date appended to the name of the new ontology version. Changes are applied automatically within the NeOn Toolkit, and the user will instantly see the updates in the ontology navigator of the toolkit (on the left of Fig. 11.6).

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 (Palma et al. 2008b) 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 that it is still accessible by doing some checks such as running queries and validating the results.

11.4 Conclusion

Ontology evolution is a tedious and time-consuming task. To successfully keep the ontology up to date with domain changes, ontology engineers should be supplied with the right guidelines and tool usage to make this task easier. For that, we presented in this chapter guidelines for ontology evolution covering two aspects: a high-level ontology evolution process and tool-oriented guidelines to semiautomatically identify, evaluate, and apply domain changes to ontologies.

The first aspect describes the tasks involved in the ontology evolution process from a generic perspective and discusses guidelines in possible ways to achieve each task. The second aspect aims to facilitate the process of identifying ontology changes from external domain data, checking their quality, and integrating them in the ontology, by using semiautomatic techniques. The guidelines in this case include how to use and parameterize the involved tools to achieve the optimal new ontology version.

The two aspects work together to enable ontology engineers to understand the complete picture and tasks involved in ontology evolution, to successfully move from an existing ontology state to a new one with the appropriate representation of domain changes that arise.

References

Arnold RS (1996) Software change impact analysis. IEEE Computer Society Press, Los Alamitos

Banerjee J, Kim W, Kim HJ, Korth HF (1987) Semantics and implementation of schema evolution in object-oriented databases. SIGMOD Rec 16(3):311–322

Bennett KH, Rajlich V (2000) Software maintenance and evolution: a roadmap. In: ICSE - future of SE track, ACM, New York, pp. 73–87

Berners-Lee T, Fielding R, Masinter L (2005) RFC 3986, Uniform Resource Identifier (URI): Generic syntax. Available at http://tools.ietf.org/html/rfc3986

Bohner SA (1996) Software change impact analysis for design evolution. In: Software change impact analysis. IEEE Computer Society Press, Los Alamitos, pp 67–81

Chikofsky EJ, Cross JG (1990) Reverse engineering and design recovery: a taxonomy. IEEE Softw 7(1):13–17

Fellbaum C (1998) Wordnet: an electronic lexical database. MIT Press, Cambridge

Haase P, Rudolph S, Wang Y, Brockmans S, Palma R, Euzenat J, d'Aquin M (2006) NeOn deliverable D1.1.1. Networked ontology model. Available at http://www.neon-project.org/

Klein M, Fensel D (2001) Ontology versioning for the semantic web. In: Proceedings of the international semantic web working symposium (SWWS'01), Stanford University, Stanford, CA, USA

Klein M, Noy N (2003) A component-based framework for ontology evolution. In: Proceedings of the IJCAI'03 workshop: ontologies and distributed systems, Acapulco, Mexico

Leenheer PD, Mens T (2007) Ontology management. Semantic web, semantic web services, and business applications. In: Ontology evolution. State-of the-art and future directions. Springer, New York/London

Noy N, Chugh A, Liu W, Musen M (2006) A framework for ontology evolution in collaborative environments. In: International semantic web conference, Athens, pp 544–558

Palma R (2009) Ontology metadata management in distributed environments. PhD thesis, Departamento de Inteligencia Artificial, Facultad de Informatica, Universidad Politecnica de Madrid

Palma R, Haase P, Wang Y, d'Aquin M (2007) D1.3.1 propagation models and strategies. Technical report D1.3.1, UPM; NeOn deliverable. Available at http://www.neon-project.org/

Palma R, Haase P, Corcho O, Gómez-Pérez A (2008a) An editorial workflow approach for collaborative ontology development. In: ASWC'08. Springer, Berlin

Palma R, Haase P, Jiu Q (2008b) D1.3.2 Evaluation of propagation models and strategies. Technical report D1.3.2; NeOn deliverable

Palma R, Hartmann J, Haase P (2008c) OMV – ontology metadata vocabulary for the semantic web. v. 2.4. Available at http://omv.ontoware.org/

Palma R, Haase P, Corcho O, Gómez-Pérez A (2009) Change representation for OWL 2 ontologies. In: Proceedings of the fifth international workshop OWL: experiences and directions. In ISWC09, Chantilly, VA, USA

Parsia B, Sirin E, Kalyanpur A (2005) Debugging OWL ontologies. In: Proceedings of the 14th international conference on world wide web. ACM Press, New York, pp 633–640

Plessers P (2006) An Approach to Web-based Ontology Evolution. PhD thesis, Department of Computer Science, Vrije Universiteit Brussel, Brussel

Proper HA, Halpin TA (2004) Conceptual schema optimisation – database optimization before sliding down the waterfall. Technical report, Department of Computer Science, University of Queensland

Sabou M, d'Aquin M, Motta E (2008) Exploring the semantic web as background knowledge for ontology matching. J Data Semant XI:156–190

Stojanovic L (2004) Methods and tools for ontology evolution. PhD thesis, University of Karlsruhe (TH)

Studer R, Benjamins VR, Fensel D (1998) Knowledge engineering: principles and methods. Data Knowl Eng 25(1–2):161–197

Suárez-Figueroa MC, Gómez-Pérez A (2008) First attempt towards a standard glossary of ontology engineering terminology. In: Proceedings of 8th international conference on terminology and knowledge engineering (TKE'08) Copenhagen, DENMARK, pp 1–15

Zablith F (2009) Evolva: a comprehensive approach to ontology evolution. In: Proceedings of ESWC 2009: the semantic web: research and applications – PhD symposium, Heraklion, pp 944–948

Zablith F, Sabou M, d'Aquin M, Motta E (2010) Using ontological contexts to assess the relevance of statements in ontology evolution. In: Proceedings of the 17th conference on knowledge engineering and knowledge management by the masses (EKAW), Lisbon, Portugal. Springer, Berlin

Chapter 12
Methodological Guidelines for Matching Ontologies

Jérôme Euzenat and Chan Le Duc

Abstract Finding alignments between ontologies is a very important operation for ontology engineering. It allows for establishing links between ontologies, either to integrate them in an application or to relate developed ontologies to context. It is even more critical for networked ontologies. Incorrect alignments may lead to unwanted consequences throughout the whole network, and incomplete alignments may fail to provide the expected consequences. Yet, there is no well-established methodology available for matching ontologies. We propose methodological guidelines that build on previously disconnected results and experiences.

12.1 Motivation

Ontology matching is the activity of establishing correspondences between ontologies. Correspondences express relationships supposed to hold between entities in ontologies, for instance, that a ‘district’ in one ontology is the same as a ‘kreis’ in another one or that ‘fishery’ in an ontology is a subclass of ‘company’ in another one. An alignment may be used to link an ontology with its background, i.e. set it in a more general context: This is typically what is achieved by providing an alignment with an upper-level ontology. An alignment can also be used to link an ontology with its previous versions or alternative ontologies in other applications.

We use interchangeably the terms matching operation (focussing on the input and result), matching task (focussing on the goal and the insertion of the task in a wider context) and matching activity (focussing on the internal processing).

J. Euzenat (✉)
INRIA & LIG, F-38330 Montbonnot Saint-Martin, France
e-mail: Jerome.Euzenat@inria.fr

C. Le Duc
Université Paris 8, 93200 Saint-Denis, France
e-mail: Chan.Leduc@iut.univ-paris8.fr

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_12, © Springer-Verlag Berlin Heidelberg 2012

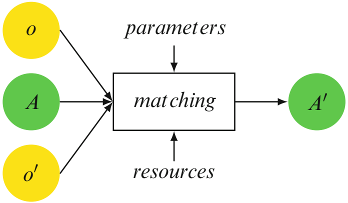


Fig. 12.1 The matching activity (From Euzenat and Shvaiko 2007)

The ontology matching process may be summarised as in Fig. 12.1 by a process taking two ontologies (o and o') as input and returning an alignment (A'), i.e. a set of correspondences. In addition, this process can take as input an initial alignment and various parameters. Ontology matching can be used with more than two ontologies. However, in this chapter, we restrict ourselves to matching two ontologies. As simple as it seems, ontology matching is an unsolved problem and a delicate activity which requires care (Euzenat and Shvaiko 2007). Many matching methods exist, and not one fits all needs.

Ontology matching is a very important operation in modern ontology engineering because of the networked environment in which ontologies are engineered and supposed to work. Methodologically, it is worthwhile to express relations between ontologies because this allows (1) for working with small and self-sufficient modules instead of monolithic ontologies, (2) for expressing the links between two versions of the same ontology and thus to upgrade data from one ontology to another or (3) for putting back an ontology in the context of an upper-level ontology, allowing it to play better with other ontologies. Networked ontologies are sets of ontologies together with alignments relating the entities of these ontologies. These ontologies may be related because they are complementary, two independent domain ontologies, e.g. sales and tyres, refinement, a domain ontology specialising a top-level ontology, or supplementary, a version replacing another version or two ontologies about the same domain. In networked ontologies, alignments are as important as the ontologies themselves because relationships between ontologies are the basis of networks.

Hence, methodological guidance for ontology matching is particularly required and needs to be supported for helping ontology engineers to develop semantic applications. Contrary to ontology building which is an open-ended (design) activity, ontology matching is an inductive activity bounded by the ontology to be matched. Hence, it requires a more focussed methodology.

Yet, very little support exists for such an activity at the methodological or at the tool level. Even in the database field, where similar but simpler problems have existed for years, there is no consensus methodology on how schema matching can be conducted. This chapter provides guidance for matching ontologies based on existing partial guidelines and overall experience collected so far in the field.

We do not consider ontology matching as an independent activity. On the contrary, we consider it as related to ontology management: When ontologies evolve, alignments must follow this evolution. Moreover, as proposed in the work

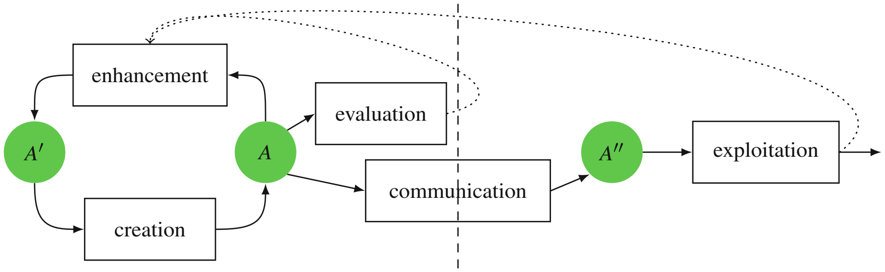


Fig. 12.2 The ontology alignment life cycle (Adapted from Euzenat et al. 2008)

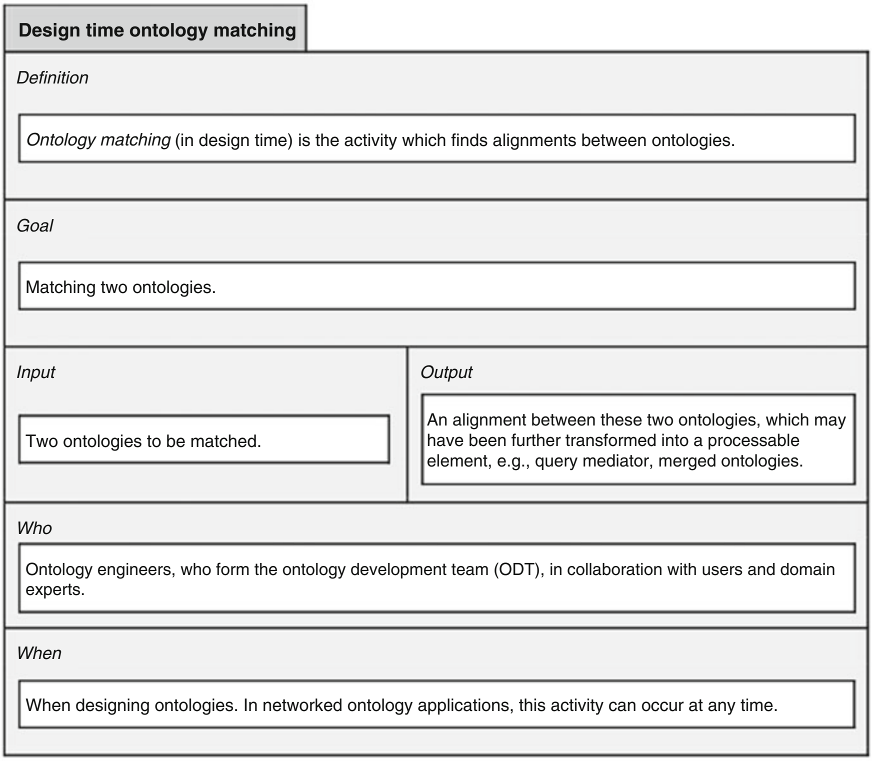
of Euzenat et al. (2008), ontology matching should be considered in a dynamic perspective in which the result of matching has its own life cycle and will have to be maintained and evolved. This is illustrated by Fig. 12.2, representing the alignment life cycle. This life cycle takes into account the evolution of alignments as well as the importance of considering alignment as first-class objects which can be shared. As such, alignments can be manipulated to better suit the needs of users. We consider this ontology alignment life cycle and further investigate the methodological guidelines for supporting it. In the spirit of NeOn (see Chap. 1), these guidelines put the emphasis on networks of ontologies as well as reusing ontologies and alignments.

In what follows, we first introduce synthetic descriptions of the ontology matching activity (Sect. 12.2). Then, we discuss the issue of the format in which alignments have to be delivered in order to support reusable matching (Sect. 12.3) before considering step by step the proposed methodological guidelines (Sect. 12.4). Then we present support offered by tools for the proposed methodological guidelines (Sect. 12.5). Finally, examples are given (Sect. 12.6) before concluding.

12.2 Ontology Matching Filling Cards

We present below two different ontology matching activities. These depend on the time at which ontology matching is supposed to take place. If ontology matching is supposed to occur at design time, then its goal is to match two ontologies for connecting them in a network; if it is to occur at runtime, then the goal of the activity is to generate a matching process that achieves ontology matching at runtime.

This distinction between runtime and design time ontology matching is very important in practice because the output of the two operations is not the same. At design time, the resulting alignment is used for relating the different ontologies which will be used at runtime, for instance, for transforming queries. At runtime,



Design time ontology matching

Definition

Ontology matching (in design time) is the activity which finds alignments between ontologies.

Goal

Matching two ontologies.

Input

Two ontologies to be matched.

Output

An alignment between these two ontologies, which may have been further transformed into a processable element, e.g., query mediator, merged ontologies.

Who

Ontology engineers, who form the ontology development team (ODT), in collaboration with users and domain experts.

When

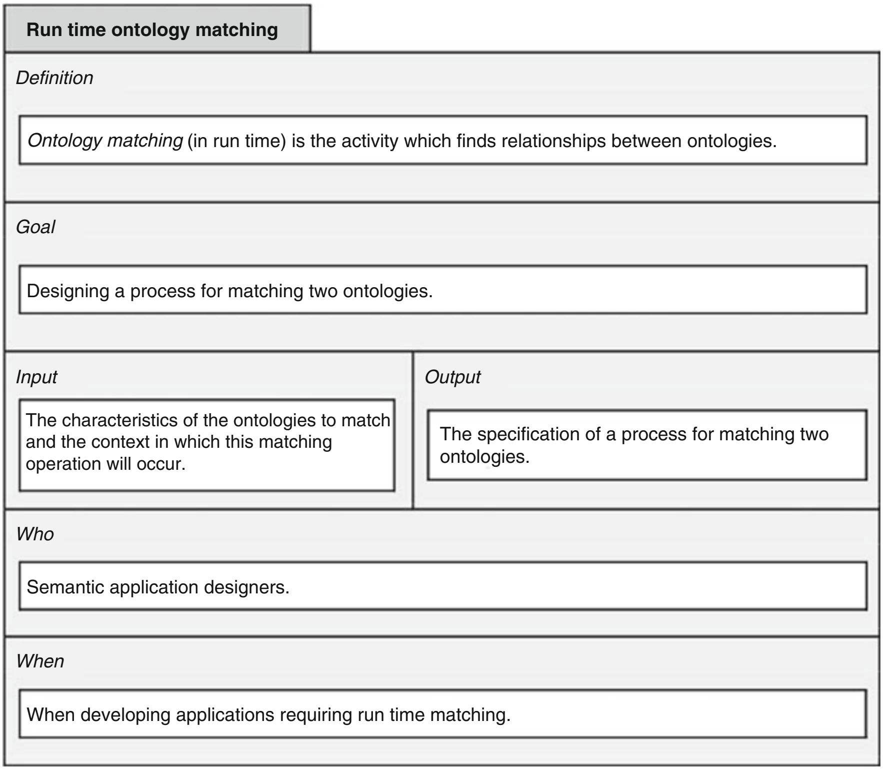
When designing ontologies. In networked ontology applications, this activity can occur at any time.

Fig. 12.3 Design time ontology matching

ontology matching is used for finding alignments between ontologies which were not known at design time. This could be for composing semantic web services using different ontologies, for instance.

When ontology matching is performed at design time (see Fig. 12.3), only the resulting alignment is available at runtime: no more matching is necessary. So, there is no runtime constraint on matching. When it is performed at runtime, no design time alignment is available; the matching will occur at runtime. So, the goal of the designer is to design a matching process instead of an alignment (see Fig. 12.4). In this case, runtime constraints (speed, memory) may apply to matching.

However, functionally, these two operations can also be seen as the same since they, in practice, generate an ontology matching process which is executed at different moments. Hence, the guidelines that we apply are the same in both cases because it consists of choosing software components which are applied at different time.



Run time ontology matching

Definition

Ontology matching (in run time) is the activity which finds relationships between ontologies.

Goal

Designing a process for matching two ontologies.

Input

The characteristics of the ontologies to match and the context in which this matching operation will occur.

Output

The specification of a process for matching two ontologies.

Who

Semantic application designers.

When

When developing applications requiring run time matching.

Fig. 12.4 Runtime ontology matching

12.3 Alignments and Formats

Although formats should not be a main concern for methodology, it is here very important because the input and output of most of the tasks is an alignment. Hence, choosing a common alignment representation makes tasks interoperable and allows for better sharing and reusing the product of the ontology matching activity.

Ontology alignments are sets of relationships between ontology entities. Alignments may be expressed in various languages. For instance, the two relations mentioned in the introduction can be expressed in OWL (Horrocks et al. 2003) through `owl:equivalentClass` and `rdfs:subClassOf`, but they can also be expressed in SKOS (Miles and Bechhofer 2009) through `skos:exactMatch` and `skos:broaderMatch`. Other applications may mandate a different form like views in databases, mediators in web services frameworks or even merged ontologies. The advantage of such representations is that they can be processed.

However, application-specific output is not particularly interoperable. It is not easy to transform a database view into OWL axioms or SKOS statements into ODEDialect (Corcho and Gómez-Pérez 2007). Indeed, when the alignment is expressed in OWL, its only possible use is to 'merge' two OWL ontologies. It cannot easily be used to import data from one ontology to another or to translate queries. Moreover, such formats are not easy to share and retrieve (see Sect. 12.4.7) or to manipulate (see Sect. 12.4.6), e.g. for merging the results of several matchers if they do not use a format that supports such manipulations.

Hence, in order to avoid early commitment to a particular type of usage, it is to be preferred to keep the alignments in a declarative language. Such a language allows for manipulating and composing alignments as well as for generating the required representation (OWL, SKOS and others) when necessary.

Using a neutral and declarative representation (Euzenat 2004) provides the opportunity to distribute and share alignments among applications. This is why, in the remainder, only 'alignments' are considered.

12.4 Detailed Guidelines

Ontology matching has been the focus of a lot of attention in the recent years. However, little work has been carried out on the methodological support for finding alignments. We provide here the outline for such methodological guidelines. It can be summarised by the workflow of Fig. 12.5. Each task of this workflow will be described in subsections.

12.4.1 Identifying Ontologies and Characterising Needs

The first task in finding alignments is to identify the ontologies to be matched and to characterise the need. Indeed, the type of required alignment will be different if the

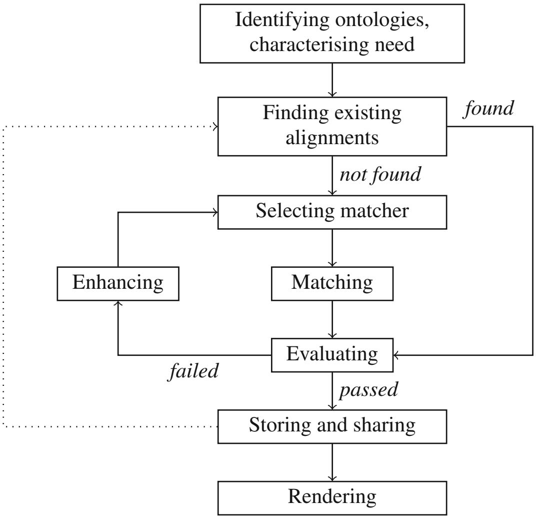


Fig. 12.5 The matching methodology workflow. It goes step by step through characterising the problem, selecting existing alignments, selecting appropriate matchers, running the matchers, evaluating the results and correcting the choices made before (matchers, parameters), documenting and publishing good results and finally using them

goal is to merge two ontologies in a knowledge-based system or to add yet another data source to a query mediator. In the former case, the alignment will have to be strictly correct, otherwise the system may draw incorrect inferences, but the relationships can be diverse: subsumption and disjointness assertions can be very useful. In the latter case, lack of completeness is not a problem since other sources may return the missing answers, but relations other than equivalence are not straightforwardly used in query mediation. This first task is similar to Activity 3 of the work of Corcho (2005), called ‘design of [a] translation system’, which specifies how to characterise source and target ontologies for ontology translation.

It is also useful to characterise the kind of ontologies: Are they labelled in the same natural language? What is their expressiveness? Are individuals related to the ontologies available?

Characterising the situation in which matching will be performed should not be neglected. It will determine the choice of matchers or alignments as well as the parameters to care for. Euzenat and Shvaiko (2007) identified several parameters:

- Are instances available at match time?
- Is matching performed under time constraints?
- Has matching to be performed automatically?
- Must the alignment be correct?
- Must the alignment be complete?
- What type of operation (merging ontologies, transforming queries) is to be performed?

These characteristics of the situation are requirements for the ontology matching process. There has been research attempting to refine such requirements. Mochol (2009) gave a very precise description of the type of ontologies to be matched depending on their size, expressiveness, language and role, e.g. domain ontology or upper-level ontology.

12.4.2 Finding Existing Alignments

Finding existing alignments which satisfy the need of the application is the second task. Alignments may be found on the web or through specialised directories. Reusing existing alignments should be privileged because of the cost of generating such alignments. For that purpose, the task of sharing (see Sect. 12.4.7) prepares alignment retrieving.

Ideally, alignments should come with annotations characterising their level of trustworthiness, the purpose for which they have been built and the type of relations they use.

These alignments must concern the ontologies to be matched, and they have to satisfy the constraints related to the alignment established in Sect. 12.4.1. In particular, correctness and completeness are criteria to use for selecting among various alignments.

These criteria may be assessed manually, on a sample, or can be inferred through the properties of their generation methods. In particular, one can use metadata attached to such alignments. They can reveal the method used for matching the ontologies (in particular, if these are automatic or manually generated alignments), they can cover manual assessments about the alignment (people publishing them can annotate the alignments to tell what they are good for) or they may contain indications of their intended use which can be matched with that of the current situation.

So, practically, selecting an alignment requires:

- Finding alignment repositories
- Finding those alignments between the ontologies to match
- Assessing the capacity of these alignments to address the needs previously identified, either based on metadata, or on the content of the alignments
- Deciding for one alignment based on this assessment

If apparently suitable alignments are available, the user can directly go to the validation task (Sect. 12.4.5). Otherwise, it is necessary to create a new alignment from the ontologies, as is explained in Sect. 12.4.3.

12.4.3 Selecting a Matcher

In order to build a new alignment, a suitable matcher has to be found. Many matchers have been developed over the years, and they provide different results on different types of data sets and matching contexts. Hence, the criteria elicited in the characterisation phase (Sect. 12.4.1) are also used for selecting the most appropriate matcher.

There have been several studies about how to choose a matcher depending on the characteristics of the ontologies and those of the expected alignments. They are worth taking into account.

Euzenat et al. (2006) provide a simple method for weighting matcher capabilities (speed, automaticity, precision and recall as measured in matcher evaluations) against the application requirements defined as the answers to the questions of Sect. 12.4.1.

The work of Mochol (2009) uses a deep classification of matchers and the matching context in order to assess which matcher will be more adapted to a particular context. This assessment is made using the Analytic Hierarchy Process (AHP) which guides the decision process of choosing a matcher. It can work on automatic or manual mode.

The OntoMas system (Huza et al. 2006) has been developed for helping and teaching how to carry out matching. For choosing a matcher, it processes a set of symbolic rules over a classification of tools and a characterisation of tasks.

The problem of such methods is that they require extensive information about available matchers which is not always available or always accurate when the assessment comes from the matcher developers. An important source of information is the result of the various matcher evaluation campaigns that have been run. The most important one is the Ontology Alignment Evaluation Initiative (OAEI) campaigns[1]. They have evaluated many matchers in a variety of situations. So their results can be taken into account when choosing a matcher. They are currently further developed in the context of the SEALS project[2].

So, in practice, choosing a matcher can be achieved by:

- Finding available matchers
- Assessing their capacity to generate alignments that fill the identified requirements by reading their documentation or comparing their performances in similar tasks during evaluations
- Deciding for one matcher based on this assessment

Other works try to automate this task, or the choice of matcher parameters, on the fly (Sayyadian et al. 2005). Such work can be used in runtime matching processes.

12.4.4 Matching Ontologies

The next task consists of running the matcher against the ontologies and collecting the resulting alignment. It may seem like the simplest task, methodologically speaking, because matchers have been designed exactly for this purpose.

But the user should not hesitate to run the matcher several times or to run several matchers, trying different sets of parameters and different thresholds. It is also useful to process matching incrementally by curating the returned alignment and feeding it again to the matcher for improving it.

In fact, all the procedure that can be applied at the enhancing phase (see Sect. 12.4.6) can also be directly applied during the matching phase without any prior evaluation.

Hence, this task can be further decomposed into a more complex sub-workflow (see Fig. 12.6). Section 12.4.6 provides some refinements of the matching workflow.

12.4.5 Evaluating Alignments

Once an alignment has been obtained, it is necessary to perform a final screening and validation. Evaluation can be applied on alignments that have been retrieved as

[1] http://oaei.ontologymatching.org

[2] http://www.seals-project.eu

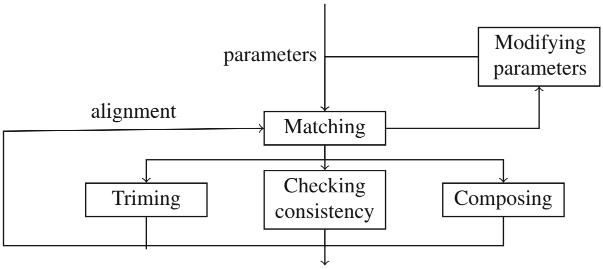


Fig. 12.6 The sub-workflow of fine-tuning matchers (all tasks but matching are optional). After matching, it is possible to apply automatically some alignment manipulation that can trim the alignment under a threshold, check and restore the consistency of an alignment or compose the alignment with another alignment. The result of these manipulations can be fed back as input to the matching operation or can be the final result of the workflow. Alternatively, it is possible to modify the parameters of the matcher and to run it again. These operations can be triggered manually or automatically

well. This task corresponds to the *evaluation* task of Fig. 12.2. Very precise methodological guidelines for evaluation of ontology networks are provided in Chap. 9 which may be applied here as well (note that the 'identify evaluation criteria and frame of reference' task corresponds to our 'identifying ontologies and characterising needs' task, see Sect. 12.4.1). We consider here what is specific to alignment evaluation during the matching activity. The evaluate/enhance loop in Fig. 12.5 corresponds to the feedback after evaluation in Fig. 12.2.

Evaluation consists of assessing properties of the obtained alignment. It can be performed either manually or automatically. Manual evaluation can be achieved by running a dry test of the final application or by asking an independent expert to assess the quality of the alignment and perform some manual assessment. For that purpose, graphical tools which allow to navigate quickly both in the alignment and in the ontologies are invaluable.

An often overlooked functionality of matching algorithms is their ability to give explanation for the provided alignments. Explanations can be obtained by interacting with the matcher or by accessing metadata about a stored alignment.

Automated quantitative evaluation can be performed by using techniques for evaluating alignments used in matcher evaluation campaigns such as OAEI[1] or SEALS[2]. These would require to extract samples from the results and computing measures like precision and recall which would provide an approximation of correctness and completeness.

There is no definitive answer as to what is a good result for evaluation. The evaluation must be performed so as to assess evaluation criteria. The characterisation of the problem (Sect. 12.4.1) aims at defining such success criteria. For some applications, high recall is required, while for some others, recall is not important. Moreover, the meaning of 'high' is not the same for all applications: A critical application which can break if some correspondence is missing will require 100% recall while a non critical application may be satisfied with 98%.

If the evaluation results are positive, i.e. the alignment satisfies these success criteria, then the obtained alignment can go through the next task, storing and sharing (Sect. 12.4.7); otherwise, the alignment can be improved (Sect. 12.4.6) before being input to the matcher and/or another matcher and/or parameters can be chosen.

12.4.6 Enhancing Alignments

Enhancement can be obtained either through manual modification of the alignment, e.g. with the help of an alignment editor, or the application of refinement procedures, e.g. selecting correspondences by applying thresholds. This enhancing task can lead to:

- The selection of another matcher, as in Fig. 12.5, by going back to Sect. 12.4.3
- The selection of another set of parameters to use with the same matcher, as in Fig. 12.6
- The manipulation of the alignment through trimming under a particular threshold or combining several alignments, as in Fig. 12.6

Among these procedures, the most straightforward one consists of trimming the alignment under some thresholds. There are many different ways to apply automatic thresholds (Euzenat and Shvaiko 2007). Some work has introduced double thresholding: Above the upper threshold, correspondences are selected, under the lower threshold, they are discarded, and the remaining correspondences are brought to the attention of the user (Lambrix and Liu 2009).

It may also restore consistency when the resulting alignment has been detected inconsistent in the evaluation (Sect. 12.4.5). By consistency checking, we do not necessarily mean logical consistency checking, but rather that the result does not violate particular constraints which may be:

- Acyclicity
- Syntactic anti-patterns (Roussey et al. 2009)
- Full logical consistency

Enhancing may then consist of selecting a subset of the correspondences in an alignment which satisfies the constraints. Algorithms developed in (Meilicke and Stuckenschmidt 2009) are particularly suited for that purpose.

Alignments obtained from various sources, such as other matchers or alignment libraries, may be composed in a single alignment through various operators: composition, meet, join and union.

12.4.7 Storing and Sharing

An extra task is to save and share the obtained alignment in a declarative format and to give it proper annotations to record its provenance and purpose. This task is very often overlooked but it is vital if one wants to find alignments in the corresponding task (Sect. 12.4.2): carefully annotating alignments will help others to reuse them. This task corresponds to the *communication* task of Fig. 12.2, and the dotted arrow in Fig. 12.5 corresponds to the availability of stored alignments after communication.

When an alignment is deemed worth publishing, then it can be annotated, stored and communicated to other parties interested in such an alignment.

Annotations of alignments should record the information that is useful for the ‘finding existing alignment’ task (Sect. 12.4.2). In particular, what is the purpose of this alignment, what is the assessment of its quality? Noy et al. (2008) discuss various kinds of metadata that are useful to record with correspondences. There are a few normalised vocabularies for doing this, and in particular the ontology metadata vocabulary (Hartmann et al. 2005). Other useful information like the algorithm used for computing it, the time it took or the source alignments and the date of matching can generally be recorded automatically.

Below is a sample of metadata associated with an alignment in the Alignment API:

dc:date	2009/10/23
align:method	fr.inrialpes.exmo.align.impl.methods.StringDistAlignment
align:time	421
omwg:purpose	Query mediation
align:creator	JohnDoe

while correspondence annotations can be:

align:measure	.7768
align:note	"manualy validated"

Storing an alignment requires some type of persistent storage. This is usually achieved through the use of a database management system, but a web site based on a file system may be sufficient. However, alignments must be properly indexed to retrieve them when necessary on various characteristics (one ontology, pairs of ontologies, arity, etc.). Indexing can be direct through a URI identifying alignments or indirect through queries looking for alignments based on their metadata. In general, it is preferable that both access modes be available.

Finally, these alignments may be shared by interested communities. For that purpose, they should be accessible on the web through HTML interfaces or web services.

There are several software supporting sharing alignments on the web. The Alignment server[3] and Cupboard (d’Aquin et al. 2009) are general-purpose servers providing alignments in the Alignment format. BioPortal[4] is specialised in biomedical ontologies and provides individual correspondences (called mappings in this system).

[3] http://alignapi.gforge.inria.fr

[4] http://bioportal.bioontology.org

12.4.8 Rendering Alignments

Finally, the alignment is transformed into another form or interpreted for performing actions like mediation or merging.

This task corresponds to the *exploitation* task of Fig. 12.2. It is the natural outcome of matching. The exploitation of the alignment may be denoted by a different activity name, e.g. merging or query translating, taking directly the alignment as input. However, it may happen that ontology matching is considered as an activity in itself in which case it will deliver its output in an appropriate format for another task. This is what is called ‘rendering’.

Rendering may deliver the alignment as such in RDF in order to be processed by an interpreter such as a query mediator. But it also can be transformed, as discussed in Sect. 12.3, into OWL axioms, SKOS relations or sets of `owl:sameAs` statements.

The dotted arrow on Fig. 12.2 expresses the feedback after using the alignment which may contribute to enhance it.

12.5 Support for Matching Ontologies

We think that methodological guidelines are more useful and better accepted if they are supported by tools rather than delivered as rules to be applied. So far, existing support is available in the alignment manipulation part rather than the requirement analysis part.

12.5.1 Independent Tools

Some tools offer alignment manipulation that can be used for alignment enhancement (Sect. 12.4.6).

Foam (Ehrig 2007) is a framework in which matching algorithms can be integrated. It mostly offers matching and processor generation. It does not offer online services or alignment editing, but it is available as a Protégé plugin and has been integrated in the KAON2 ontology management environment. COMA++ (Aumüller et al. 2005) and Harmony (Mork et al. 2008) are stand-alone (schema) matching work benches that allow for integrating and composing matching algorithms. They support matching, evaluating, editing, storing and, for COMA++, processing alignments.

The Alignment server, associated with the Alignment API[5] (David et al. 2011), offers matching ontologies, manipulating, evaluating, storing and sharing alignments as well as processor generation. It can be accessed by clients through an API, web services, agent communication languages or HTTP. It does not support editing.

Most of the software developed for editing alignments are candidates for design time matching. The same alignment editor can be used for manipulating more precisely the obtained alignments. They should provide a convenient display of the currently edited alignments and the opportunity to discard, modify or add correspondences. Ideally, each design time function should be available from an alignment editor. Since ontologies and alignments can be very large, it is very challenging to offer intuitive alignment editing support. There exists such editor prototype such as OnaGui[6] or iMerge (El Jerroudi and Ziegler 2008).

12.5.2 Integrated Tools

Model management has been promoted in databases for dealing with data integration in a generic way. It offers a high-level view to the operations applied to databases and their relations. Rondo[7] is such a system (Melnik et al. 2003). It offers operators for generating the alignments, composing them and applying them as data transformation. It is a stand-alone programme with no editing functions.

Integrated tools integrate alignment management to ontology management.

The Web Service Modeling Toolkit (WSMT) (Kerrigan et al. 2007) is an Integrated Development Environment (IDE) for Semantic Web services which also provides ontology engineering capabilities. Among other capabilities, WSMT proposes a set of tools for manually creating, editing and storing ontology alignments. It offers a set of methods and techniques that assist ontology engineers in their work, such as different graphical perspectives over the ontologies, suggestions of the most related entities from the source and target ontology and guidance throughout the matching process (Mocan et al. 2006). WSMT and the ontology engineer work together in an iterative process which involves cycles consisting of suggestions from the tool side and validation and creation of correspondences from the user side.

Protégé is an ontology edition environment which offers design time support for matching. In particular, it features Prompt[8] (Noy and Musen 2003), an environment that provides some matching methods and alignment visualisation. Prompt allows to match, compare and merge ontologies. Since alignments are expressed in an

[5] http://alignapi.gforge.inria.fr

[6] http://sourceforge.net/projects/onagui/

[7] http://infolab.stanford.edu/modman/rondo/

[8] http://protege.stanford.edu/plugins/prompt/prompt.html

ontology, they can be stored and shared through the Protégé server mode. Prompt can be extended through a plugin mechanism.

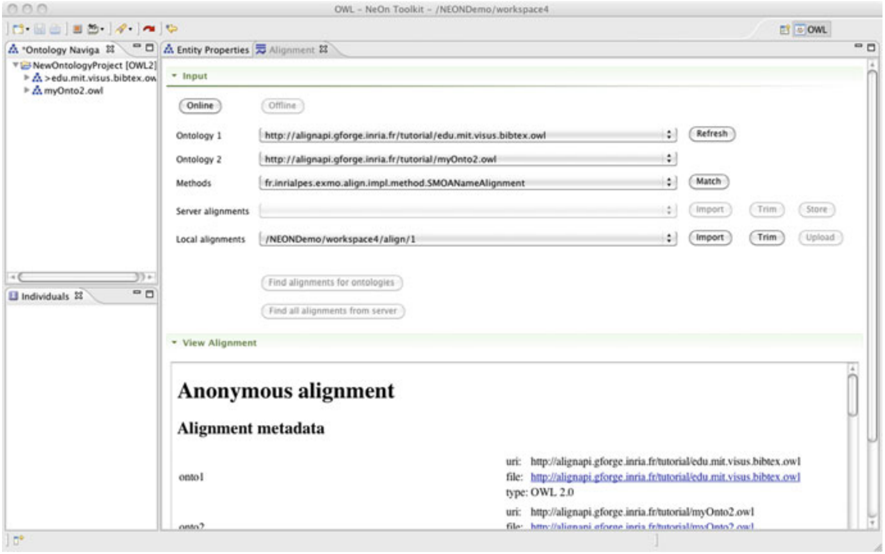
12.5.3 The NeOn Toolkit Alignment Plugin

The NeOn[9] project produced a toolkit for ontology management (see Chap. 13) which features runtime and design time ontology alignment support.

NeOn supports ontology alignments in both the NeOn Toolkit and the Cupboard ontology server.

The NeOn Toolkit Alignment plugin works in two modes: an offline mode in which the user can work locally on the alignments. The user can run the matchers which are embedded in the toolkit against ontologies in the NeOn Toolkit and manipulate alignments which are in the local environment. Figure 12.7 shows two selected ontologies and a matching method that can be applied to them. It also shows a local alignment between these two ontologies to which operations such as trimming under a threshold or rendering in OWL ('import') can be applied.

The online mode connects the NeOn Toolkit to an alignment server allowing to share ontologies and to apply these operations on alignments stored on the server.



Ontology 1 http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl

Ontology 2 http://alignapi.gforge.inria.fr/tutorial/myOnto2.owl

Methods fr.inrialpes.exmo.align.impl.method.SMOANameAlignment

Server alignments

Local alignments /NEONDemo/workspace4/align/1

Anonymous alignment

Alignment metadata

onto1
uri: http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
file: http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
type: OWL 2.0

onto2
uri: http://alignapi.gforge.inria.fr/tutorial/myOnto2.owl
file: http://alignapi.gforge.inria.fr/tutorial/myOnto2.owl

Fig. 12.7 The NeOn Toolkit Alignment plugin interface

[9] http://www.neon-project.org

Of course, alignments can move back and forth between the server and the local environment.

Both online and offline modes provide the functions of the Alignment API: retrieving alignments, matching ontologies, trimming alignments under various thresholds, storing them in permanent stores and rendering them in numerous output formats. These operations support the whole alignment life cycle (Fig. 12.5).

The Alignment plugin allows one to automatically compute and manage ontology alignments. More precisely, it offers the following functionalities:

- Find alignments between ontologies or those available on the server
- Match ontologies
- Trim alignments by applying thresholds to existing alignments
- Retrieve and render alignments in a particular format
- Upload and store an alignment permanently on the server

Alignments stored in the server can be further shared through the Cupboard ontology server. It allows for indexing alignments available from alignment servers. Hence, these alignments can be available to each Cupboard user to be stored in her cupboard and, as for ontologies, be rated and annotated. Cupboard provides direct access to alignments as well as indirect access to the Alignment server to generate new alignments when they are missing.

12.6 Examples

In this section, we consider a user having to connect an ontology designed for drug and prescription to existing ontologies outside. These examples are closely related to the application presented in Chap. 20.

12.6.1 Identifying Needs

More precisely, the newly proposed Semantic Nomenclature ontology (presented in Chap. 20), designed from schemas of pharmacological firm databases, has to be matched to ontologies available on the web. This could help searching for literature about concerned drugs or exporting drug interactions as linked data for other applications to take them into account.

The requirements for this matching activity allow it to be performed offline, without time constraints, so the use of the NeOn Toolkit and user supervision is perfectly suited. The ontologies, having been developed independently and for different purposes, are not expected to match exactly. Correct correspondences are expected, completeness is secondary. The type of operation to be performed with the resulting alignments is data export (for exposing linked data) and query translation (for connecting to the literature).

Table 12.1 Characteristics of the considered ontologies

Organisation	Ontology	Lang.	Form.	Classes	Relations	Properties
ATOS	Semantic Nomenclature	English	OWL	67	20	26
UMLS	Semantic Network	English	OWL	199	105	0
LDIS	DrugOnt	English	OWL	28	26	32
NLM	RX nomenclature	English	OWL	10	16	0

12.6.2 Identifying Ontologies

Watson (d'Aquin and Motta 2011) allows for finding further ontologies that may be useful. These are:

- The LDIS Drug ontology[10] which has been designed for prescription application in hospitals (related with electronic patient record)
- The UMLS Semantic Network ontology[11] which is well used for literature indexing because of its extensive coverage
- The RxNorm ontology[12] which is used for classifying drugs and is suited to search the literature

A quick study of these ontologies shows the characteristics displayed in Table 12.1.

More ontologies on this topic are available, and a comparison can be found in Herrero Cárcel and Pariente (2009).

The ontologies are relatively homogeneous being in English (with some Spanish comments in Semantic Nomenclature) and OWL. They have comparable sizes with the notable exception of UMLS.

12.6.3 Finding Existing Alignments

Finding available alignments may be achieved by using an alignment server. In the present case, there is no alignment available between these ontologies.

12.6.4 Selecting a Matcher

The user then proceeds by selecting a matcher suited to match these ontologies. In this case, given that ontologies are about a very close and normalised domain, they

[10] http://lsdis.cs.uga.edu/projects/asdoc/DrugOnt_schema.owl

[11] http://swpatho.ag-nbi.de/owldata/umlssn.owl

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

are written in the same natural language; the user may select very simple matchers based on the strings naming entities. There are several matchers available, either under the NeOn Toolkit or the Alignment server; the best way is to try them and to see the results (see Sect. 12.6.5).

In a second iteration, tests have been performed with more elaborate matchers such as a simple use of WordNet which would use synonyms to match terms and distance in the hypernym graph. Or it can use the Aroma matcher which will attempt at determining association rules between concepts before extracting an alignment between them (David et al. 2007).

12.6.5 Matching

The simple StringDistAlignment method with different string distances is run, and results are displayed in Table 12.2.

The user first ran the method with Levenshtein measure (edit distance) and SMOA measure which tries to better interpret the way people label things, e.g. by using syntactic variations. The threshold has been put to .75 so as to avoid considering far-reaching similarity between strings. Later, a threshold of .85 has been applied in order to further ensure correctness (because a higher threshold will eliminate unlikely matches).

12.6.6 Evaluating

There is no automatic way to evaluate these results. They have to be manually looked into by the user to assess their quality (they can be displayed by alignments editors).

Concerning the drug ontology, the small returns with the Levenshtein distance are obviously correct. The use of SMOA provides mostly new correct matches, such as interacts/hasInteraction. The only non–fully correct matches are the matching of DrugInteraction and OtherInteraction to interaction. Using SMOA with a .75 threshold provides reasonable results. Some more matches, such as

Table 12.2 Number of matched entities in Semantic Nomenclature depending on matching method and threshold. The example has used two iterations displayed by the horizontal line

Method	Threshold	DrugOnto	RxNorm	UMLSSN
Levenshtein	.85	3	3	9
–	.75	3	4	11
SMOA	.85	8	8	20
–	.75	11	11	34
WordNet	.75	5	6	12
Aroma		8	5	30

isIndicated/has_indication_text, could have been found, but not many. The small number of matches can be explained as follows: Semantic Nomenclature is more oriented towards the drug production and commerce processes, while the drug ontology is targeting the consumption process.

Concerning UMLSSN and RxNorm, Levenshtein was better than SMOA which was returning quite a lot of unwanted matches, such as isProducedBy/produces or Clinical_Finding/Clinical_Drug. After a closer examination, there is no real reason to find more correspondences than those provided by Levenshtein, so the user may want to use these. Especially with UMLSSN, it seems that the labels have been chosen so that they correspond to those of UMLS so they exactly match.

Using the more elaborate matchers has confirmed this. They have only returned plausible but not necessarily valid correspondences, such as Physical_Entity/ Physical_Object.

12.6.7 Enhancing

Enhancement may be achieved by two means: either by manual edition of the resulting alignment or by running a new matcher, using new parameters or applying different threshold to the results. This is what has been done by using different thresholds and testing the more elaborate matchers, i.e. starting back to Sect. 12.6.4. In the end, once the SMOA alignment with the drug ontology has been found acceptable with respect to other results, this alignment is manually edited and selected.

Both means can be interleaved: It is possible to edit an alignment and to use it as further input for a matcher.

12.6.8 Storing and Sharing

Once an alignment of sufficient quality is established, especially if it has been curated by hand, it must be better documented, for instance, by adding metadata explaining how it has been obtained, who has curated it and what is the reached confidence in each correspondence. This is illustrated in Sect. 12.4.7. Then, it can be uploaded to an Alignment server so that it would be visible to other people (in the previous step of Sect. 12.6.3).

12.6.9 Rendering

Finally, the obtained alignments have to be used. We have considered that the data expressed in the Semantic Nomenclature ontology could be converted in the drug ontology so as to communicate critical information about interaction. This may be achieved either by generating an XSLT transformation applying to the data expressed in XML for obtaining the interactions under the drug ontology or a more elaborate process may take advantage of the alignment to generate links between instances of both ontologies.

On the other side, if RxNorm or UMLSSN is used to query bibliographical databases, the alignments may be used for translating queries expressed with respect to the Semantic Nomenclature into queries expressed in the two other ontologies and eventually evaluate them in parallel.

12.7 Conclusions

Establishing relations between ontology entities is part of modern ontology engineering and a very important activity for networked ontology engineering. This activity remains difficult though there are many solutions for carrying it out. We proposed methodological guidelines for ontology matching which integrates with the alignment life cycle and can cooperate with ontology engineering methodologies. In particular, we paid a particular attention to alignment sharing and reuse. These guidelines are based on research work on particular tasks: Some of these have been investigated in depth and others have not. Similarly, some tools cover parts of these guidelines, but none is able to support them entirely.

Hence, more work is necessary to achieve a fully instrumented ontology matching methodological support, and no doubt it will raise some demands for improvement in the proposed methodological guidelines.

12.8 Further Readings

There are few methodological accounts of ontology matching. Mochol (2009) is the exception: a whole thesis dedicated to matcher selection. Corcho (2005) has considered more specifically the methodology for designing an ontology translation method, including a matcher. Euzenat and Shvaiko (2007) covers many facets of ontology matching, but not extensively methodology. It provides insights of most of the tasks of the above methodological path. Euzenat et al. (2008) is more methodological but not focussed on the individual act of matching.

Acknowledgements We thank Pavel Shvaiko for his comments on a previous version of this chapter. This work has been partly supported by the European Commission IST project NeOn (IST-2006-027595).

References

Aumüller D, Do H-H, Maßmann S, Rahm E (2005) Schema and ontology matching with COMA++. In: Proceedings of the 24th international conference on management of data (SIGMOD), software demonstration, Baltimore, MD, USA, pp 906–908

Corcho Ó (2005) A layered declarative approach to ontology translation with knowledge preservation. Ios Press, Amsterdam

Corcho Ó, Gómez-Pérez A (2007) ODEDialect: a set of declarative languages for implementing ontology translation systems. J Univers Comput Sci 13(12):1805–1834

d'Aquin M, Motta E (2011) Watson, more than a semantic web search engine. Semant Web J 2:55–63

d'Aquin M, Euzenat J, Le Duc C, Lewen H (2009) Sharing and reusing aligned ontologies with cupboard. In: Proceedings of 5th ACM KCap poster session, Redondo Beach, CA, USA, pp 179–180. URL ftp://ftp.inrialpes.fr/pub/exmo/publications/daquin2009a.pdf

David J, Guillet F, Briand H (2007) Association rule ontology matching approach. Int J Semant Web Inf Syst 3(2):27–49

David J, Euzenat J, Scharffe F, Trojahn dos Santos C (2011) The Alignment API 4.0. Semant Web J 2(1):3–10. URL http://iospress.metapress.com/content/4164891n48p5v826/

Ehrig M (2007) Semantic web and beyond: computing for human experience. In: Ontology alignment: bridging the semantic gap. Springer, New York. Acitrezza, Italy, ISBN 0–387–32805-X

El Jerroudi Z, Ziegler J (2008) iMERGE: interactive ontology merging. In: Proceedings of the 16th EKAW demonstration track, Acitrezza, Italy, pp 52–56

Euzenat J (2004) An API for ontology alignment. In: Proceedings of 3rd international semantic web conference (ISWC), Hiroshima, Japan, Lecture notes in computer science, vol 3298. Springer, Berlin/Heidelberg, pp 698–712

Euzenat J, Shvaiko P (2007) Ontology matching. Springer, Heidelberg

Euzenat J, Ehrig M, Jentzsch A, Mochol M, Shvaiko P (2006) Case-based recommendation of matching tools and techniques. Deliverable 1.2.2.2.1, knowledge web. URL ftp://ftp.inrialpes.fr/pub/exmo/reports/kweb-126.pdf

Euzenat J, Mocan A, Scharffe F (2008) Ontology alignment: an ontology management perspective. In: Hepp M, De Leenheer P, De Moor A, Sure Y (eds) Ontology management: semantic web, semantic web services, and business applications. Springer, New York, pp 177–206

Hartmann J, Palma R, Sure Y, Haase P, Suárez-Figueroa MC, Haase P, Gómez-Pérez A, Studer R (2005) Ontology metadata vocabulary and applications. In: Meersman R, Tari Z, Herrero P et al (eds) Proceedings of the International conference on ontologies, databases and applications of semantics (ODBASE-2005), Lecture notes in computer science, vol 3762. Springer, Berlin/Heidelberg/New York, pp 906–915

Herrero Cárcel G, Pariente T (2009) Revision of ontologies for semantic nomenclature: pharmaceutical networked ontologies. Deliverable 8.3.2, NeOn project

Horrocks I, Patel-Schneider P, van Harmelen F (2003) From SHIQ and RDFto OWL: the making of a web ontology language. J Web Semant 1(1):7–26

Huza M, Harzallah M, Trichet F (2006) OntoMas: a tutoring system dedicated to ontology matching. In: Proceedings of the 1st ISWC international workshop on ontology matching (OM), Athens, GA, USA, pp 228–323

Kerrigan M, Mocan A, Tanler M, Fensel D (2007) The web service modeling toolkit – an integrated development environment for semantic web services. In: Proceedings of the 4th European semantic web conference (ESWC) system description track, Innsbruck, Austria, pp 303–317

Meilicke C, Stuckenschmidt H (2009) An efficient method for computing alignment diagnoses. In: Proceedings of the 3rd international conference on web reasoning and rule systems (RR-2009), Chantilly, VA, USA, pp 182–196

Melnik S, Rahm E, Bernstein P (2003) Rondo: a programming platform for model management. In: Proceedings of the 22nd international conference on management of data (SIGMOD), San Diego, CA, USA, pp 193–204

Miles A, Bechhofer S (2009) SKOS simple knowledge organization system: reference. Recommendation, W3C. URL http://www.w3.org/TR/skosreference

Mocan A, Cimpian E, Kerrigan M (2006) Formal model for ontology mapping creation. In: Proceedings of the 5th international semantic web conference (ISWC), Athens, GA, USA, Lecture notes in computer science, vol 4273. Springer, Berlin/Heidelberg/New York, pp 459–472

Mochol M (2009) The methodology for finding suitable ontology matching approaches. PhD thesis, Freie Universität Berlin. URL http://www.diss.fuberlin.de/diss/receive/FUDISS_thesis_000000008124

Mork Peter, Seligman Len, Rosenthal Arnon, Korb Joel, Wolf Chris (2008) The harmony integration workbench. J Data Semant XI:65–93

Noy N, Musen M (2003) The PROMPT suite: interactive tools for ontology merging and mapping. Int J Hum-Comput Stud 59(6):983–1024. ISSN: 1071–5819. doi:http://dx.doi.org/10.1016/j.ijhcs.2003.08.002

Noy N, Griffith N, Musen M (2008) Collecting community-based mappings in an ontology repository. In: Proceedings of the 7th international semantic web conference (ISWC), Karlsruhe, Germany, pp 371–386

Patrick Lambrix, Qiang Liu (2009) Using partial reference alignments to align ontologies. In: Proceedings of the 6th European semantic web conference (ESWC 2009), Heraklion, Germany, Lecture notes in computer science, vol 5554. Springer, Berlin/Heidelberg/New York, pp 188–202

Roussey C, Corcho Ó, Vilches Blázquez LM (2009) A catalogue of owl ontology antipatterns. In: Proceedings of the 5th international conference on knowledge capture (KCap-2009), Redondo Beach, CA, USA, pp 205–206

Sayyadian M, Lee Y, Doan A-H, Rosenthal A (2005) Tuning schema matching software using synthetic scenarios. In: Proceedings of the 31st international conference on very large data bases (VLDB), Trondheim, Norway, pp 994–1005

Part III
The NeOn Toolkit

Chapter 13
Overview of the NeOn Toolkit

Michael Erdmann and Walter Waterfeld

Abstract The NeOn Toolkit is one of the major results of the NeOn project. It is a state-of-the-art, open-source, multiplatform ontology engineering environment, which provides comprehensive support for the ontology engineering life cycle of networked ontologies. It is based on an open and modular plugin architecture that allows adding additional plugins realizing more advanced features supporting more complex ontology engineering activities. A substantial number of plugins have been developed within and outside the NeOn consortium and are available at the NeOn Toolkit homepage. The NeOn Toolkit supports the Web Ontology Language OWL 2, the ontology language specified by the W3C, and features basic editing and visualization functionality. Its user interface, especially the presentation of class restrictions, makes the NeOn Toolkit accessible to users that do not have long experience with ontologies but instead know the object-oriented modeling paradigm. In the chapter, we will present the feature set of the NeOn Toolkit and how to use it. A second part explains some architecture and implementation background and how new plugins can be integrated into the common platform.

13.1 Introduction to the NeOn Toolkit

The NeOn Toolkit is a state-of-the-art, open-source, multiplatform ontology engineering environment, which provides comprehensive support for the ontology engineering life cycle of networked ontologies (see Chap. 1).

M. Erdmann (✉)
ontoprise GmbH, An der RaumFabrik 33a, 76227 Karlsruhe, Germany
e-mail: michael.erdmann@ontoprise.de

W. Waterfeld
Software AG, Uhlandstraße 12, 64297 Darmstadt, Germany
e-mail: Walter.Waterfeld@softwareag.com

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_13, © Springer-Verlag Berlin Heidelberg 2012

In order to support such broad ontology modeling functionality, it has an open and modular plugin architecture. The NeOn Toolkit core provides the framework and some basic functionality. More advanced features supporting more complex ontology engineering activities (see part "Ontology Engineering Activities" in this book) are provided by *plugins*. A substantial number of plugins have been developed within and outside the NeOn consortium and are available at the NeOn Toolkit homepage[1].

The NeOn Toolkit supports the Web Ontology Language OWL 2, the ontology language specified by the World Wide Web Consortium (Motik et al. 2009). Neither this chapter nor this book is intended to be an OWL tutorial. Thus, we assume a basic understanding of the concepts of this ontology language, that is, the different kinds of axioms and entities that OWL 2 provides[2]. The NeOn Toolkit provides editing support for all entities which are defined in OWL (e.g., classes) and also supports all but a few of the OWL axioms (e.g., the assertion that there is no relation between two individuals cannot yet be expressed in the NeOn Toolkit).

This chapter consists of two main parts. In the first part, we will discuss the basic features of the NeOn Toolkit and describe how to use them. Since the NeOn Toolkit is an open and extensible platform, we will look under the hood of the toolkit and we will discuss the building blocks and standards that are used to define the toolkit and which can be exploited by developers to create additional functionalities for the NeOn Toolkit in the form of plugins. Thus, we mean here developers of ontology functionalities and not developers of ontologies, where the latter we consider as users of the NeOn Toolkit.

13.2 The NeOn Toolkit: For Users

In this section, we demonstrate the features of the core NeOn Toolkit and how to use them.

13.2.1 General Features

13.2.1.1 The Screen Layout

The NeOn Toolkit operates on *ontologies* (Gruber 1993) that are stored locally in a so-called *workspace*. The workspace is the organizational unit, where users can organize all the artifacts of their current work. At any time, a user works only in

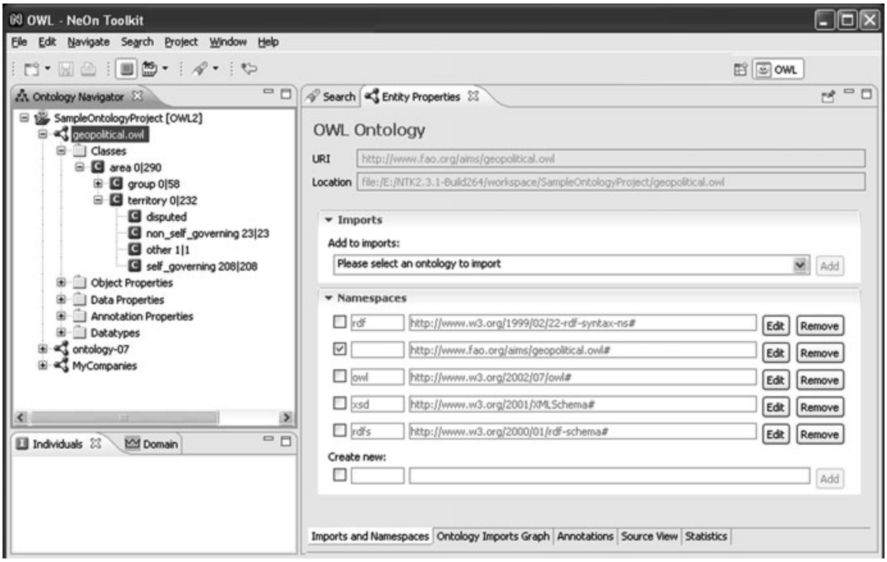
[1] http://NeOn-Toolkit.org

[2] A good introduction to the OWL language can be found in Hitzler et al. (2009).

exactly one workspace, which is organized into independent *projects*. Each project can contain multiple ontologies. The ontologies within a project can refer to each other via the *owl:import* statement. In this way, multiple ontologies can be viewed and edited at the same time. Different versions of the same ontology can be stored in different projects or in different workspaces.

The workspace and project metaphor is visualized in the NeOn Toolkit by the layout of the basic OWL *perspective*. A perspective is an Eclipse mechanism to describe the composition and layout of different *views* in a window. A view is a subwindow displaying certain information and/or allowing user input. The OWL perspective initially contains three main views, which represent the basic functionality of the NeOn Toolkit, as shown in Fig. 13.1.

- *Ontology navigator*: This view in the top left of the screen displays all ontology projects of the current workspace. Each project can hold multiple ontologies, and each ontology contains folders for classes, properties (object, data, and annotation properties), and data types. Each of these folders displays a hierarchical presentation of the classes, properties, and data types of the respective ontology.
- *Individuals:* When you select a class in the *Ontology Navigator*, a list of all its individuals is listed in this view.
- *Entity properties*: The NeOn Toolkit displays details of each selected entity in this view, which takes the most space of the screen and is located on the right-hand side. The content of this view adapts, depending on the type of the currently selected entity (ontology, class, property, individual, etc.). Since most entities (can) have a lot of relevant information, which would not necessarily fit on a single screen, the entity property view uses several tabs (at the bottom of the view) where users can switch between different aspects of the entity.



URI http://www.fao.org/aims/geopolitical.owl

Location file:/E:/NTK2.3.1-Build264/workspace/SampleOntologyProject/geopolitical.owl

Add to imports: Please select an ontology to import

□ rdf http://www.w3.org/1999/02/22-rdf-syntax-ns#

☑ http://www.fao.org/aims/geopolitical.owl#

□ owl http://www.w3.org/2002/07/owl#

□ xsd http://www.w3.org/2001/XMLSchema#

□ rdfs http://www.w3.org/2000/01/rdf-schema#

Create new:

□

Fig. 13.1 Screenshot of the NeOn Toolkit

The basic operations users perform with the NeOn Toolkit are (1) the creation of new objects (projects, ontologies, and other entities) and (2) the manipulation of their properties. The *Ontology Navigator* can be used for creating new objects. This can be done via the context menu on the empty canvas or on one of the many already existing nodes in the tree. In most cases, the shortcut *CTRL* + *N* can also be used to create a new entity, depending on the currently selected node (the context). Thus, users can create new ontologies in a project, new subclasses, or new subproperties, etc.

For manipulating the properties of an entity, users typically will use the *Entity Properties View*. Since OWL ontologies consist of axioms, users also essentially interact with the NeOn Toolkit on the axiom level. The tabs of the *Entity Properties View* contain forms for different aspects of each entity where users can add new properties or alter or delete existing ones. In later sections, we will give some more details for the most commonly used forms.

13.2.1.2 Loading and Saving

In order to load or save ontologies (outside of the workspace), the NeOn Toolkit provides import and export features, which are available from the context menu of ontology nodes in the *Ontology Navigator* tree and from the File menu. The NeOn Toolkit supports several serialization formats for OWL ontologies:

- OWL/RDF – the official OWL 2 W3C recommendation in RDF/XML
- OWLX – an OWL 2-XML presentation according to the OWL 2 recommendation
- OWL 2 – the functional syntax of the OWL 2 recommendation
- OMN – the Manchester syntax for OWL 2 (Horridge and Patel-Schneider 2009)
- TTL – a Turtle serialization format for the RDF graph of the OWL 2 ontology (Beckett and Berners-Lee 2008)

13.2.1.3 Entity Label Modes

The labels that are displayed for entities in the different fields of the user interface depend on the selected *Entity Label Mode*. The toolbar icon labeled “ns”[3] shown in Fig. 13.2 lets the user choose between four different modes:

- *Complete URI*: displays the complete URI of an entity (e.g., “http://www.fao.org/aims/geopolitical.owl#group”) including its local name.
- *Local name*: displays only the local name of an entity (e.g., “group”). This makes the ontology a lot more readable, but since the local names are not necessarily unique, there is potential for conflicts.

[3] “ns” here stands for “namespace.”

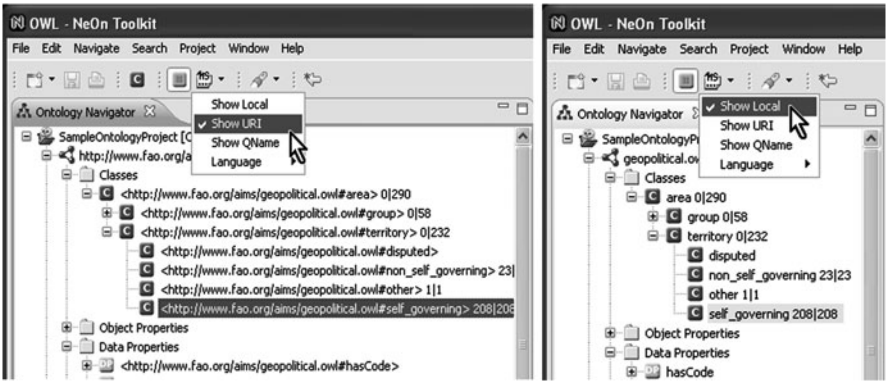

OWL - NeOn Toolkit
File Edit Navigate Search Project Window Help
Ontology Navigator
Show Local
✓ Show URI
Show QName
Language
SampleOntologyProject
http://www.fao.org/a
Classes
<http://www.fao.org/aims/geopolitical.owl#area> 0|290
<http://www.fao.org/aims/geopolitical.owl#group> 0|58
<http://www.fao.org/aims/geopolitical.owl#territory> 0|232
<http://www.fao.org/aims/geopolitical.owl#disputed>
<http://www.fao.org/aims/geopolitical.owl#non_self_governing> 23|
<http://www.fao.org/aims/geopolitical.owl#other> 1|1
<http://www.fao.org/aims/geopolitical.owl#self_governing> 208|208
Object Properties
Data Properties
<http://www.fao.org/aims/geopolitical.owl#hasCode>
OWL - NeOn Toolkit
File Edit Navigate Search Project Window Help
Ontology Navigator
✓ Show Local
Show URI
Show QName
Language
SampleOntologyP
geopolitical.ow
Classes
area 0|290
group 0|58
territory 0|232
disputed
non_self_governing 23|23
other 1|1
self_governing 208|208
Object Properties
Data Properties
hasCode

Fig. 13.2 Selecting how entities should occur in the user interface

- *QName*: displays the namespace prefix and local name of an entity (e.g., "geo: group"). This is the recommended setting and is especially useful when multiple ontologies are used or different namespaces are in use. In case a default namespace is defined for an ontology, no prefix is shown for this namespace.
- *Language*: If the ontology contains labels (*rdfs:label*) for entities, this option can display the human readable labels in a specified language, thus providing multilanguage support. Note that all languages are available, which are specified in the preferences.

13.2.1.4 Manchester Syntax

Most fields in the *Entity Property View* can take single entity names, URIs, QNames, or local names. Others can hold complex class expressions, for example, the superclasses of a class can be either a simple class (referenced by its ID) or an anonymous class specified as a complex class expression. The NeOn Toolkit supports the Manchester syntax for formulating these expressions.

The Manchester syntax is a user-friendly compact syntax for the ontology language OWL (Horridge and Patel-Schneider 2009), especially suited for writing OWL class expressions. Although the syntax borrows ideas from the OWL abstract syntax, it is much less verbose, meaning that it is quicker to write and easier to read. While following the compactness of the DL syntax, special mathematical symbols such as the universal or the existential quantifiers have been replaced by keywords such as "only" and "some." The Manchester syntax for OWL 2 is not strictly a part of the OWL 2 recommendation by the W3C, but it was developed by members of the W3C OWL 2 working group, and the syntax is published as a W3C Note. This Note contains the complete language specification and also examples for using it.

Table 13.1 Example expressions in Manchester syntax

OWL construct	Manchester syntax keyword	Example
owl:someValuesFrom	**Some**	*hasRelative* ***some*** *Person*
owl:allValuesFrom	**Only**	*hasAuthor* ***only*** *Writer*
owl:hasValue	**Value**	*writtenBy* ***value*** *Goethe*
owl:minCardinality	**Min**	*hasPlayer* ***min*** *3*
owl:cardinality	**Exactly**	*hasPlayer* ***exactly*** *3*
owl:maxCardinality	**Max**	*hasPlayer* ***max*** *3*

Table 13.2 Complex class constructions

OWL construct	Manchester syntax keyword	Example
owl:intersectionOf	**And**	*Writer* ***and*** *Male*
owl:unionOf	**Or**	*Male* ***or*** *Female*
owl:complementOf	**Not**	***not*** *Child*

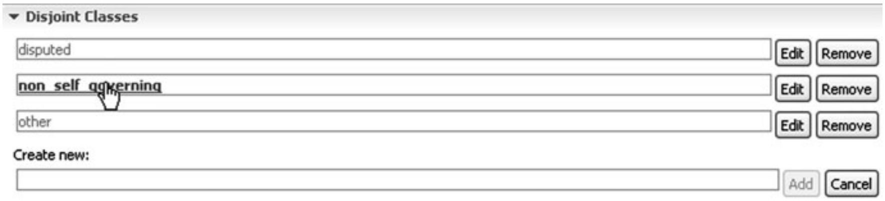
Table 13.1 gives examples for the look and feel of the Manchester syntax. Restrictions in OWL are special anonymous classes, constructed using certain OWL primitives. Based on existing classes and class expressions, the Manchester syntax allows to create more complex classes using Boolean class constructors (see Table 13.2).

Of course, users can also create even more complex classes by nesting multiple class expressions within each other. For example, the following formula in Manchester syntax describes the set of people who have at least one child that has some children that are only men, that is, grandparents that only have grandsons:

Person ***and*** *hasChild*
some (*Person* ***and*** (*hasChild* ***only*** *Man*) ***and*** (*hasChild* ***some*** *Person*))

13.2.1.5 Navigation

The fields in the *Entity Properties View* often contain references to other entities, for example, the domain of a property or the superclass of a class. These references can also be nested within more complex expressions. Oftentimes, it is desirable and useful to jump directly to the referenced entities to further inspect the model. The NeOn Toolkit provides a handy navigation utility which supports this task. By pressing the Control (Ctrl) key and hovering over the name of an entity, the text becomes highlighted in blue and gets underlined (see Fig. 13.3). It becomes a clickable hyperlink. By using the left mouse button, the NeOn Toolkit will jump to the selected entity by updating the *Ontology Navigator* and *Entity Properties View* to show it.



Disjoint Classes

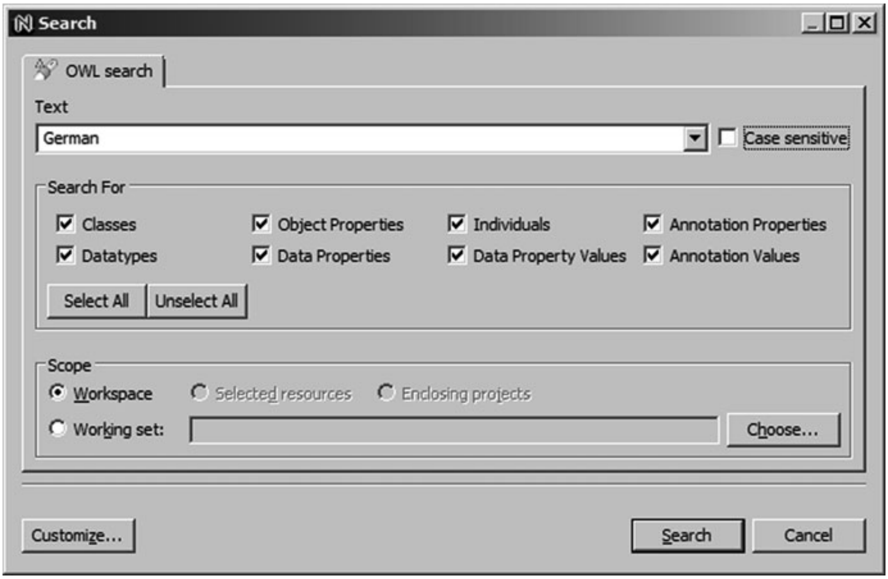
disputed

non_self_governing

other

Create new:

Fig. 13.3 Entity labels as hyperlinks



Text: German

□ Case sensitive

Search For

☑ Classes ☑ Object Properties ☑ Individuals ☑ Annotation Properties

☑ Datatypes ☑ Data Properties ☑ Data Property Values ☑ Annotation Values

Scope

● Workspace ○ Selected resources ○ Enclosing projects

○ Working set:

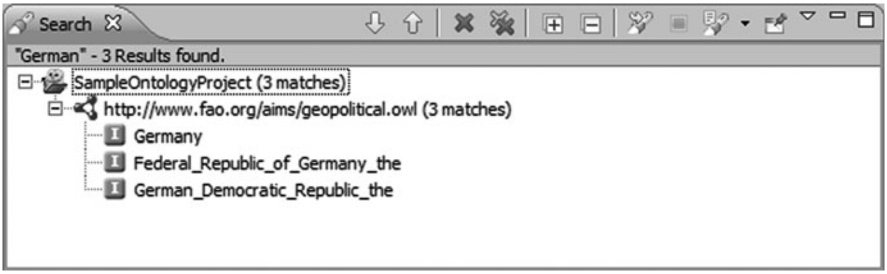
Fig. 13.4 Search dialog

13.2.1.6 Search and Finding References

Another useful feature of the NeOn Toolkit is its *Search* facility. Ontologies can be quite large in size, and to this purpose, the NeOn Toolkit provides a helpful search dialog (see Fig. 13.4). Besides a search term, the dialog allows users to specify the type of entity and also the scope for the search (e.g., a single ontology, a project, or the complete workspace). In order to have fast reaction times, the search is internally performed on asserted knowledge only.

After hitting the *Search* button, the results will be displayed in the *Search View*. It presents a list of all entities matching the keyword, organized according to the ontology (and project) they were found in. By double-clicking a line in the *Search View*, it will jump to the respective entity (see Fig. 13.5).

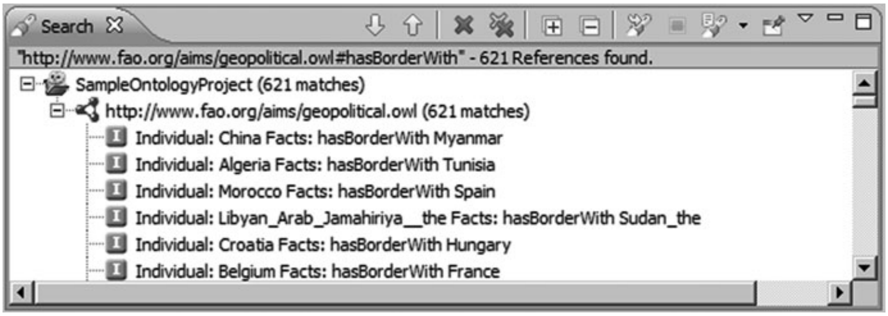
All nodes in the *Ontology Navigator* provide a context menu action to *Find References*. When selecting this item, the NeOn Toolkit will find all axioms that



"German" - 3 Results found.

- SampleOntologyProject (3 matches)
  - http://www.fao.org/aims/geopolitical.owl (3 matches)
    - Germany
    - Federal_Republic_of_Germany_the
    - German_Democratic_Republic_the

Fig. 13.5 Search results organized by project and ontology



"http://www.fao.org/aims/geopolitical.owl#hasBorderWith" - 621 References found.

- SampleOntologyProject (621 matches)
  - http://www.fao.org/aims/geopolitical.owl (621 matches)
    - Individual: China Facts: hasBorderWith Myanmar
    - Individual: Algeria Facts: hasBorderWith Tunisia
    - Individual: Morocco Facts: hasBorderWith Spain
    - Individual: Libyan_Arab_Jamahiriya__the Facts: hasBorderWith Sudan_the
    - Individual: Croatia Facts: hasBorderWith Hungary
    - Individual: Belgium Facts: hasBorderWith France

Fig. 13.6 Search results for a “find references” search

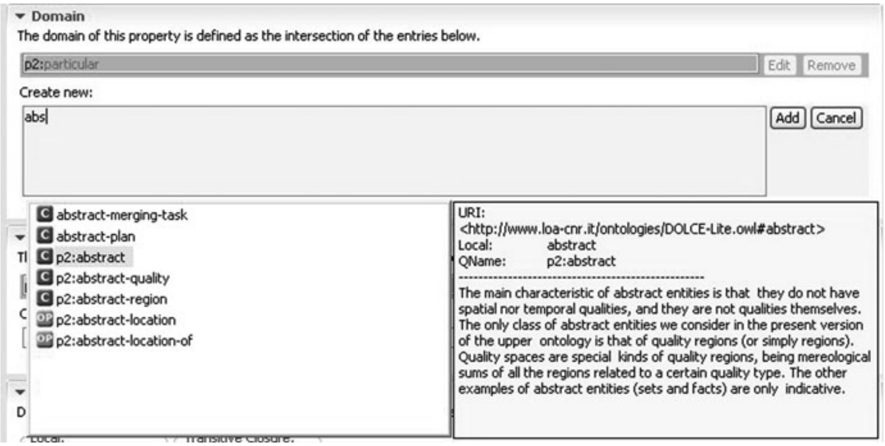
contain this entity. The results, again, are displayed in the *Search View* and are clickable, exactly as in the case of the normal search above (see Fig. 13.6).

13.2.1.7 Autocompletion

The NeOn Toolkit also features a useful autocompletion function, which is available in all text boxes of the *Entity Properties View* while in edit mode. Autocompletion is triggered by clicking *CTRL* + *Space* and starts automatically after a second of idle time. Since expressions in Manchester syntax can be complex, the NeOn Toolkit also tries to limit its proposals to applicable entity names, for example, only class names or only property names. In Fig. 13.7, the user typed “abs” and the Toolkit proposes a list of classes that contain this substring. On the right, it also shows a tooltip with the complete URI and also the *rdfs:comment* for the selected entities, in this case “p2:abstract.”

13.2.1.8 The Help Facility

The built-in Help system of the NeOn Toolkit is quite elaborate, and a full user documentation is available from within the Toolkit. It is accessible via the *Help*



▾ Domain

The domain of this property is defined as the intersection of the entries below.

p2:particular

Create new:

abs

- abstract-merging-task
- abstract-plan
- p2:abstract
- p2:abstract-quality
- p2:abstract-region
- p2:abstract-location
- p2:abstract-location-of

URI:
<http://www.loa-cnr.it/ontologies/DOLCE-Lite.owl#abstract>
Local: abstract
QName: p2:abstract

The main characteristic of abstract entities is that they do not have spatial nor temporal qualities, and they are not qualities themselves. The only class of abstract entities we consider in the present version of the upper ontology is that of quality regions (or simply regions). Quality spaces are special kinds of quality regions, being mereological sums of all the regions related to a certain quality type. The other examples of abstract entities (sets and facts) are only indicative.

Fig. 13.7 Drop-down list provided by autocompletion

menu entry *Help Contents*. The help pages are organized in so-called books. The basic Toolkit provides a book about basic modeling of OWL with the NeOn Toolkit. If other plugins are installed, they typically also provide their own documentation in the form of own books.

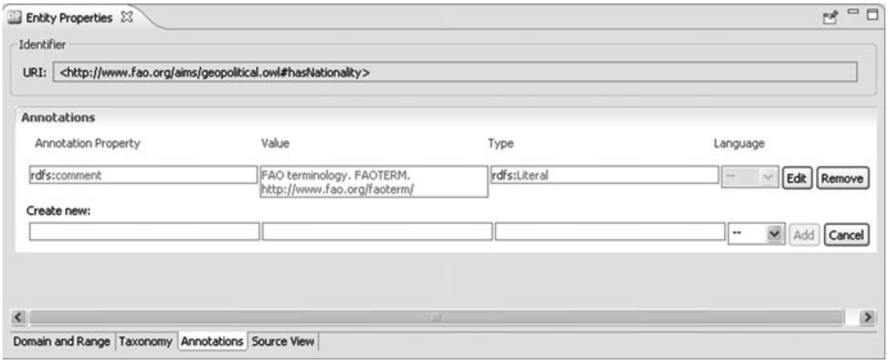
Additionally, many wizards or dialogs provide context-sensitive Help by clicking the question-mark icon in the lower left-hand corner. This will guide users to appropriate pages in the documentation.

13.2.2 OWL Entities in the NeOn Toolkit User Interface

By selecting a node in the *Ontology Navigator* or another view, users can set the focus of the NeOn Toolkit on the associated entity. As the focus changes, the content of the *Entity Properties View* updates to show the details of the newly selected entity. Depending on the *type* of entity (ontology, class, property, individual, etc.), the *Entity Properties View* looks different (e.g., the icon in the top-left corner of the view changes to reflect the type), and it offers different tabs at the bottom of the view to enable switching between the various aspects of the entity. Some tabs are shared between entity types, and we will present these first.

13.2.2.1 Annotations

The OWL 2 W3C recommendation offers the opportunity to add annotations to all kinds of entities. The NeOn Toolkit supports this feature with a single GUI element, the *Annotations Tab*, for all entities (see Fig. 13.8).



URI: <http://www.fao.org/aims/geopolitical.owl#hasNationality>

Annotation Property	Value	Type	Language
rdfs:comment	FAO terminology. FAOTERM. http://www.fao.org/faoterm/	rdfs:Literal	--

Fig. 13.8 The *Annotations Tab* is available for all entities

Annotations can either use the built-in annotation properties of OWL 2 or RDF (S), or they can use user-defined annotation properties. Using the *Edit*/*Remove* buttons, existing annotations can be altered or removed. With the last line, entitled "*create new*," additional annotations can be added. Autocompletion is supported for the property name and for the data type.

13.2.2.2 Source View

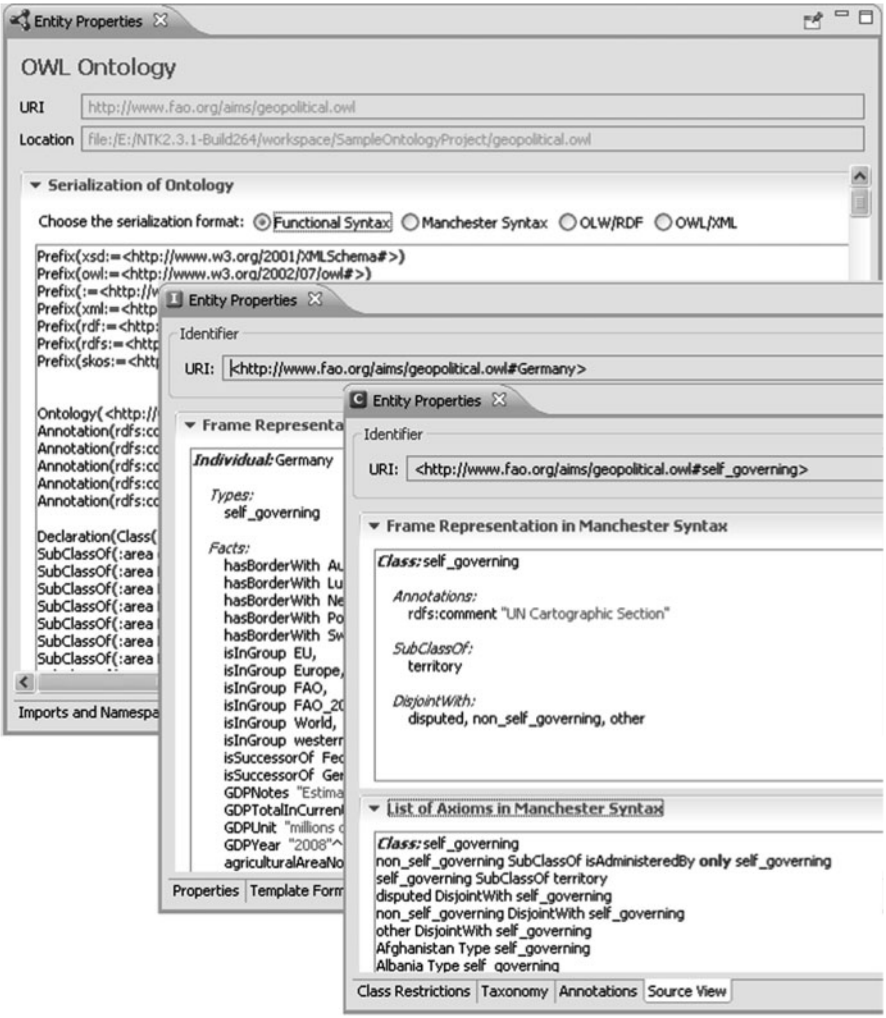
Another useful feature (for advanced users) is the ability to view an entity in some form of textual serialization. The NeOn Toolkit offers this functionality via the *Source View Tab* for each entity type. Ontologies can be displayed in several serialization formats, for example, RDF/XML, functional-style syntax, or Manchester syntax. For other entities, the NeOn Toolkit displays the Manchester syntax of the *frame* representing the entity and, additionally, a list of *all* axioms relevant for this entity. The tab uses syntax highlighting to make the display easier to read (see Fig. 13.9).

13.2.2.3 Project

The *Entity Properties View* for projects includes an *Aggregated Statistics* tab, in addition to a general tab that displays some metadata about the current project. In this tab, the NeOn Toolkit presents overall statistics about the ontologies in the project, for example, the total number of classes, properties, or axioms. Like the search, the statistics is based on asserted knowledge only.

13.2.2.4 OWL Ontology

The main tab for ontologies is the *Imports and Namespaces* tab, which lists all ontologies imported by this one (via *owl:imports*) and the defined namespaces.



OWL Ontology

URI: http://www.fao.org/aims/geopolitical.owl

Location: file:/E:/NTK2.3.1-Build264/workspace/SampleOntologyProject/geopolitical.owl

▾ Serialization of Ontology

Choose the serialization format: ● Functional Syntax ○ Manchester Syntax ○ OLW/RDF ○ OWL/XML

```
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(:=<http://w
Prefix(xml:=<http
Prefix(rdf:=<http:
Prefix(rdfs:=<http
Prefix(skos:=<htt


Ontology(<http://
Annotation(rdfs:cc
Annotation(rdfs:cc
Annotation(rdfs:cc
Annotation(rdfs:cc
Annotation(rdfs:cc

Declaration(Class(
SubClassOf(:area
SubClassOf(:area
SubClassOf(:area
SubClassOf(:area
SubClassOf(:area
SubClassOf(:area
SubClassOf(:area
```

Identifier

URI: <http://www.fao.org/aims/geopolitical.owl#Germany>

▾ Frame Representa

```
Individual: Germany

  Types:
    self_governing

  Facts:
    hasBorderWith  Au
    hasBorderWith  Lu
    hasBorderWith  Ne
    hasBorderWith  Po
    hasBorderWith  Sw
    isInGroup  EU,
    isInGroup  Europe,
    isInGroup  FAO,
    isInGroup  FAO_20
    isInGroup  World,
    isInGroup  westerr
    isSuccessorOf  Fec
    isSuccessorOf  Ger
    GDPNotes  "Estima
    GDPTotalInCurren
    GDPUnit  "millions c
    GDPYear  "2008"^
    agriculturalAreaNo
```

Identifier

URI: <http://www.fao.org/aims/geopolitical.owl#self_governing>

▾ Frame Representation in Manchester Syntax

```
Class: self_governing

  Annotations:
    rdfs:comment "UN Cartographic Section"

  SubClassOf:
    territory

  DisjointWith:
    disputed, non_self_governing, other
```

▾ List of Axioms in Manchester Syntax

```
Class: self_governing
non_self_governing SubClassOf isAdministeredBy only self_governing
self_governing SubClassOf territory
disputed DisjointWith self_governing
non_self_governing DisjointWith self_governing
other DisjointWith self_governing
Afghanistan Type self_governing
Albania Type self_governing
```

Fig. 13.9 *Source Views* for different entities

A screenshot can be seen in Fig. 13.1 (see Sect. 13.2.1.1). The set of imported ontologies and the namespace definitions can also be changed here. The closure of imported ontologies is visualized by the *Ontology Imports Graph* tab. Importing an ontology makes the entire set of classes, properties, and individuals provided by that ontology available to the current one. In the *OWL preferences* dialog, users decide whether they want to see imported axioms (and entities) in the user interface or not. Imported axioms will be highlighted with a light-blue background.

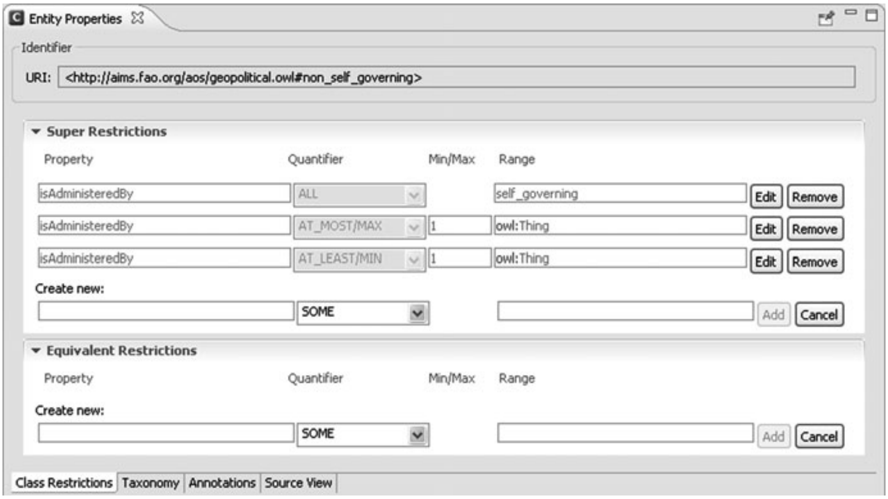
The NeOn Toolkit supports ontologies which are loaded from remote locations, for example, from the web. The *Location* field of the *Ontology Entity Properties View* shows whether the ontology is stored locally or loaded from the web.

13.2.2.5 OWL Class

Generally speaking, OWL classes can be considered as sets of individuals that share similar characteristics. These classes are organized in hierarchies, of which *owl: Thing* is the root class. OWL classes are described through so-called class descriptions. The class hierarchy is displayed in the *Ontology Navigator*. Clicking on a class shows its instances in the *Individual View* and all properties that have this class in its domain in a special *Domain View*. The number of individuals that belong to a class is also displayed in the navigator together with the class label, for example, in Fig. 13.1 (see Sect. 13.2.1.1), we see that the class "territory" has no direct instances but 232 inherited ones ("territory 0|232"). Thus, it is easier for users to see at a glance which classes are more or less populated, a key piece of information when making sense of an ontology or when keeping track of an ontology in development.

The *Entity Properties View* for classes provides two additional tabs, besides the *Annotations* and *Source View* tab. The *Taxonomy* tab lets the users view and modify the super- and subclasses of the current class, as well as equivalent and disjoint classes. A special feature of the NeOn Toolkit, not present in other ontology editors, is the special treatment of OWL restrictions. On a separate *Class Restrictions* tab (see Fig. 13.10), the user can see and specify the restrictions for the selected class and a given property. Since super restrictions are a common modeling pattern for formulating local properties with their ranges and cardinality, this presentation is adequate and easily understood, even by non-OWL experts. The NeOn Toolkit supports all OWL 2 restrictions:

- *owl:allValuesFrom* (ALL)
- *owl:someValuesFrom* (SOME)



URI: <http://aims.fao.org/aos/geopolitical.owl#non_self_governing>

Super Restrictions

Property	Quantifier	Min/Max	Range
isAdministeredBy	ALL		self_governing
isAdministeredBy	AT_MOST/MAX	1	owl:Thing
isAdministeredBy	AT_LEAST/MIN	1	owl:Thing

Create new: SOME

Equivalent Restrictions

Property	Quantifier	Min/Max	Range

Create new: SOME

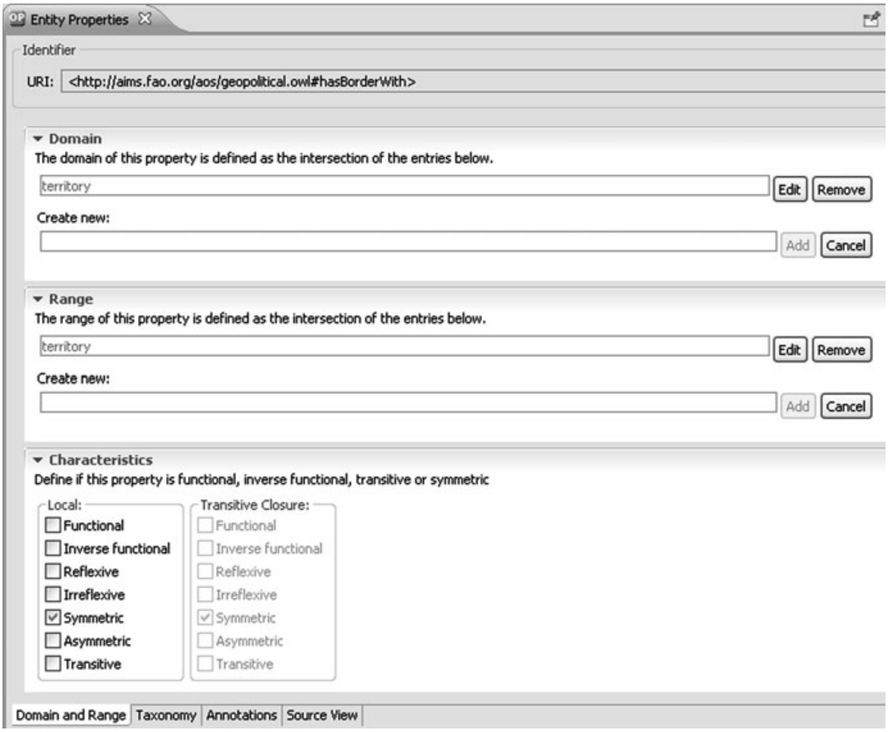
Fig. 13.10 Restrictions on OWL classes

- *owl:hasValue* (HAS_VALUE)
- *owl:hasSelf* (HAS_SELF)
- *owl:maxCardinality* (AT_MOST/MAX)
- *owl:minCardinality* (AT_LEAST/MIN)
- *owl:cardinality* (EXACTLY/CARD)

13.2.2.6 OWL Property (Object, Data, and Annotation Properties)

The *Entity Properties Views* for all three kinds of OWL properties are quite similar. In the following, we will only describe the *Entity Properties View* for object properties since its features are essentially a superset of the other properties.

The *Domain and Range* tab (see Fig. 13.11) contains information about the *rdfs:domain* and *rdfs:range* of the selected property. The values for the text fields can be arbitrary class descriptions formulated using the Manchester syntax. The domain and the range of each property are always maintained as a list, which can be manipulated with the edit and remove buttons. Thus, multiple domains and ranges are completely supported.



Entity Properties

Identifier

URI: <http://aims.fao.org/aos/geopolitical.owl#hasBorderWith>

Domain

The domain of this property is defined as the intersection of the entries below.

territory

Create new:

Range

The range of this property is defined as the intersection of the entries below.

territory

Create new:

Characteristics

Define if this property is functional, inverse functional, transitive or symmetric

Local:
□ Functional
□ Inverse functional
□ Reflexive
□ Irreflexive
☑ Symmetric
□ Asymmetric
□ Transitive

Transitive Closure:
□ Functional
□ Inverse functional
□ Reflexive
□ Irreflexive
☑ Symmetric
□ Asymmetric
□ Transitive

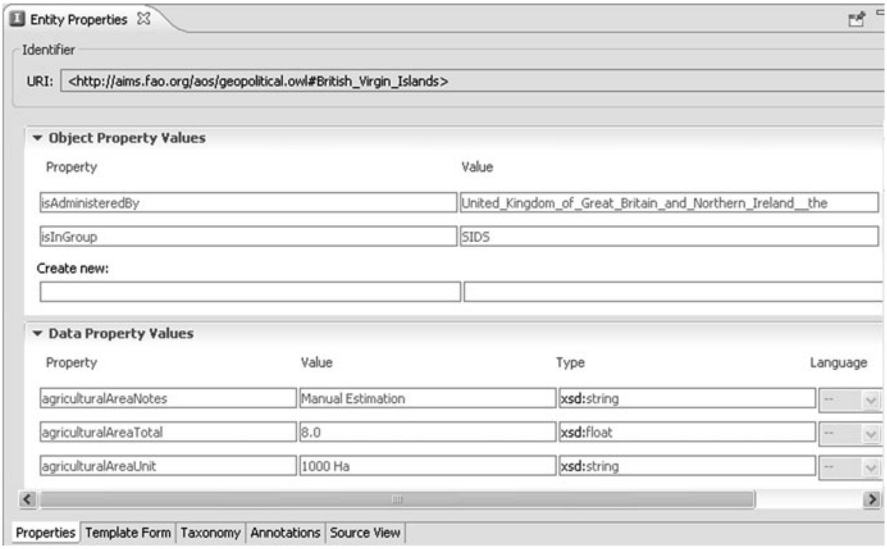
Fig. 13.11 *Domain and Range* can be defined for all properties

On the same tab, the *Characteristics* of the property are also displayed. Annotation properties have no characteristics in OWL, data properties can be defined as *functional,* and object properties can additionally be characterized as *inverse functional, reflexive, irreflexive, symmetric, asymmetric,* and/or *transitive*. Each characteristic is displayed twice. In the left-hand column, the assertions from the current ontology are shown, whereas the second column shows whether a characteristic is true in one of the directly or indirectly imported ontologies. This is based on the ontology import functionality described in Sect. 13.2.2.4.

OWL 2 and RDFS properties can be hierarchically organized. This information is collected in the *Taxonomy* tab. It displays *Super*- and *Sub-Properties*, as well as *Equivalent Properties*. Object properties also can have *Inverse Properties* or property chains (the composition of a sequence of properties) as subproperties. Thus, it is possible, for example, to specify that the property *hasUncle* is a subproperty of the concatenation of *hasParent* and *hasBrother*.

13.2.2.7 OWL Individual

The *Entity Properties View* for individuals contains the general tabs for *Annotations* and the *Source View* and two tabs that allow users to display and define properties for an individual. The *Properties* tab (see Fig. 13.12) has two sections, one for instantiating *Object Properties* and one for instantiating *Data Properties*. The *Template Form* tab provides forms for applicable properties, so that the users can see which properties are available for an individual (based on its classes) and can directly instantiate individual properties. The final tab for Individuals is the



Identifier

URI: <http://aims.fao.org/aos/geopolitical.owl#British_Virgin_Islands>

Object Property Values

Property	Value
isAdministeredBy	United_Kingdom_of_Great_Britain_and_Northern_Ireland__the
isInGroup	SIDS

Create new:

Data Property Values

Property	Value	Type	Language
agriculturalAreaNotes	Manual Estimation	xsd:string	--
agriculturalAreaTotal	8.0	xsd:float	--
agriculturalAreaUnit	1000 Ha	xsd:string	--

Fig. 13.12 The properties of an OWL individual

Taxonomy tab in which users can define equality and inequality between individuals (*owl:sameAs* and *owl:differentFrom*, respectively) and list also all classes the individual belongs to (*rdf:type*).

13.2.3 Extending the NeOn Toolkit's Feature Set

In the previous section, we have described the features that are built in the NeOn Toolkit core. Of course, the core only contains the essential functionalities of the NeOn Toolkit, and many more features (Harth 2010) are available for download. Users can configure the NeOn Toolkit environment according to their needs.

One of the prominent functionality that is available via easy-to-install features is reasoning. Thus, it is possible to cope with different reasoning functionality and with different reasoner realizations. For example, the reasoner plugin provides important reasoning functionality based on the Pellet2 and the HermiT3 reasoner (see Chap. 17).

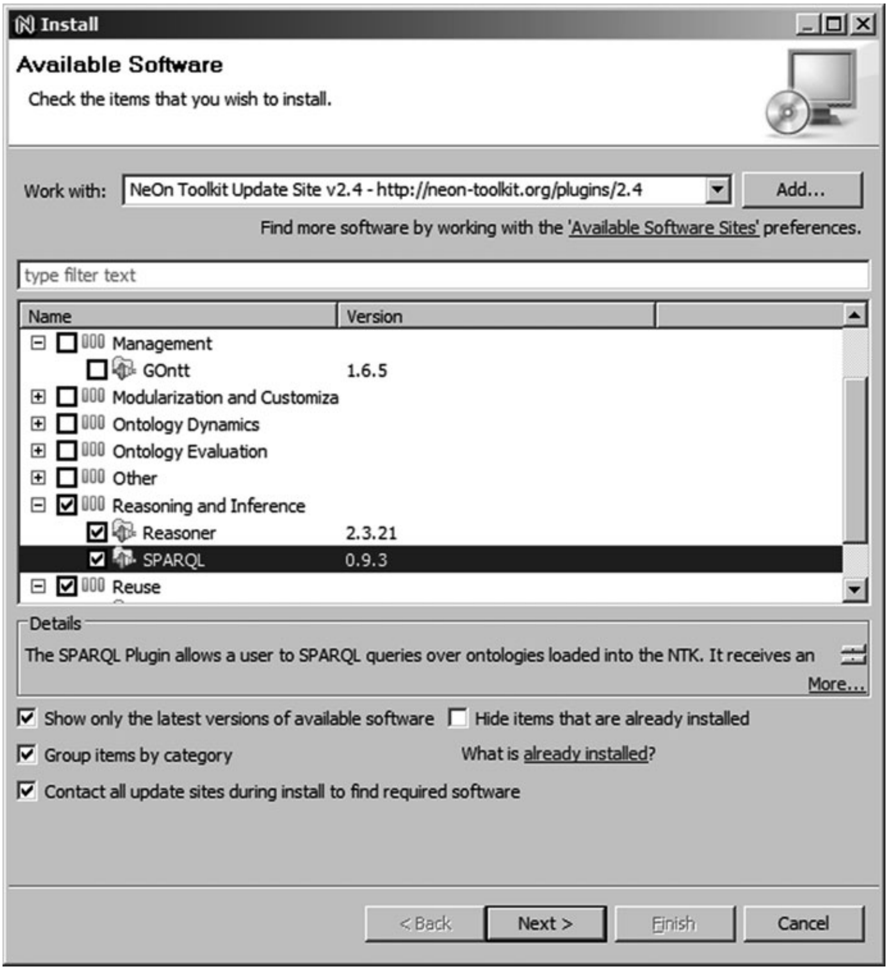
The mechanism to install new features while working with the NeOn Toolkit refers to a special location in the web, the so-called *NeOn update site*. This update site is maintained by the NeOn Foundation[4] and provides up-to-date plugins for a variety of ontology engineering activities. Users can access this update site from within the NeOn Toolkit (via *Install new software* in the *Help* menu). The resulting dialog (see Fig. 13.13) lets users select an update site (the NeOn update site is preconfigured already) and shows a list of features organized in a number of categories that can be selected, downloaded, and installed with a few clicks.

13.3 The NeOn Toolkit: For Developers

The NeOn Toolkit has an open and modular architecture, which it inherits from its underlying platform, Eclipse. Eclipse is a rich development environment, which is widely adopted in the programming world and also perfectly fits the modeling paradigm for ontologies. It provides developers a framework to easily create, publish, and integrate new features into the NeOn Toolkit. Eclipse is open-source, and the NeOn Toolkit also is published under the same open Eclipse Public License[5], which means that it can be used and extended for any purpose, commercial and noncommercial.

[4] The body responsible for the management and distribution of the NeOn Toolkit, http://www.neon-foundation.org/

[5] http://www.eclipse.org/legal/epl-v10.html



Install

Available Software

Check the items that you wish to install.

Work with: NeOn Toolkit Update Site v2.4 - http://neon-toolkit.org/plugins/2.4

Find more software by working with the 'Available Software Sites' preferences.

type filter text

Name	Version
⊟ □ Management	
□ GOntt	1.6.5
⊞ □ Modularization and Customiza	
⊞ □ Ontology Dynamics	
⊞ □ Ontology Evaluation	
⊞ □ Other	
⊟ ☑ Reasoning and Inference	
☑ Reasoner	2.3.21
☑ SPARQL	0.9.3
⊟ ☑ Reuse	

Details

The SPARQL Plugin allows a user to SPARQL queries over ontologies loaded into the NTK. It receives an

More...

☑ Show only the latest versions of available software □ Hide items that are already installed

☑ Group items by category

What is already installed?

☑ Contact all update sites during install to find required software

Fig. 13.13 Dialog for loading and installing additional plugins

The openness of the platform and the reliance on open standards was a major driver in the design and development of the NeOn Toolkit. Besides the open *Eclipse* platform[6], we use (as Eclipse also does) *OSGi* (2009) as our component framework and the *OWL API* (Horridge and Bechhofer 2009) as our data model for the ontologies managed with the NeOn Toolkit. These three important building blocks for NeOn will be described in what follows, after introducing the general architecture of the NeOn Toolkit (as an application development platform).

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

13.3.1 Architecture

The architecture of the NeOn Toolkit must cover the complete ontology functionality. This includes the coverage of tools for the whole ontology life cycle, and it must enable all ontology engineering activities. The NeOn Toolkit focuses on the development part of ontology functionality. However, for ontology-based applications, the distinction between *development time* and *runtime* is not as clear as in conventional applications, for example, schematic information like classes are often also changed and modified at runtime, which is impossible for conventional applications. Therefore, the architecture of the NeOn Toolkit also includes runtime components. The architecture must also allow the easy integration of additional ontology engineering functionalities in a highly modular fashion (Waterfeld et al. 2008a).

Thus, we defined for the NeOn Toolkit a generic architecture with a *layering approach*. The layering resembles increasing abstraction layers for ontology functionalities. The layers, however, also organize the data and control flow between the components of each layer. The components of higher layers invoke components of lower layers but not vice versa.

Based on these principles, the NeOn Toolkit architecture consists of three layers (see Fig. 13.14):

- *Infrastructure services*: These are the basic ontology services contained in all versions of the NeOn Toolkit. The OWL API implementation is the most important one.
- *Engineering components*: The main ontology functionality is contained in the engineering plugins provided by the NeOn Toolkit core and by additional plugins.
- *Front-end components*: They contain the user interfaces for the engineering plugins. They are similarly extendible like the engineering components.

This layering and more details are described in the NeOn Toolkit design documents (Waterfeld et al. 2007, 2008b).

13.3.2 APIs and Realization

13.3.2.1 Eclipse Platform

For the realization of the architecture, we use the Eclipse platform. Eclipse, which has a strong record as a software development environment, provides rich functionality for the development of plugins for the toolkit. Additionally, Eclipse provides, via its *extension point mechanism*, a simple way to extend its functionality for other types of development assets. For the NeOn Toolkit, this mechanism has been used extensively to realize the basic tools for ontology development.

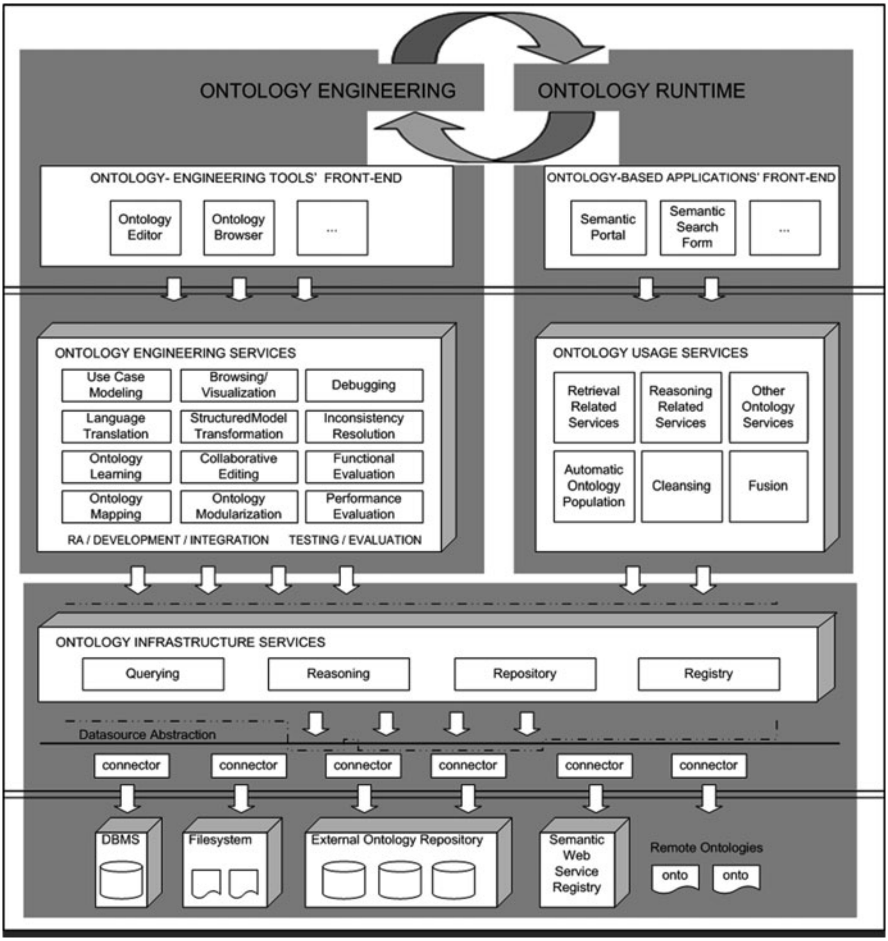


Fig. 13.14 The NeOn architecture

These can now be used to realize more specific ontology tools. The advantages of this extension point mechanism are twofold: First, it allows an easy realization of a basic ontology modeling tool because many of Eclipse's existing functionalities can be reused; second, the development and integration of additional, more specific ontology tools is almost seamless because the functionality can be easily *plugged in* as an extension of the core NeOn Toolkit.

13.3.2.2 OSGi

OSGi (2009) is a very flexible and dynamic component model. It allows a completely dynamic management of components in different versions.

Thus, components can be deployed, stopped, started, and uninstalled in the running system, and the resulting dependencies are correctly managed. Eclipse has been based on OSGi since 2004; however, not all capabilities of OSGi are yet used in the Eclipse context.

In NeOn, we have leveraged the implicit capabilities of NeOn plugins as OSGi bundles for publishing ontology engineering plugins as web services. This is possible by offering a web service container based on an OSGi server. Into such a web service container, all needed NeOn ontology engineering functionalities can be deployed. For a specific ontology engineering plugin, only a web service wrapper has to be generated and also deployed. Thus, any ontology functionality, originally only available in the NeOn Toolkit, is now available as a web service.

This is of course not possible for front-end plugins. However, due to the separation of engineering plugins and front-end plugins of the NeOn architecture, the NeOn Toolkit provides the means to realize this capability for any functionality defined in the engineering layer.

13.3.2.3 OWL API

Over the past few years, the W3C OWL working group has continuously extended and redefined the specification of the web ontology language, initially dubbed OWL 1.1, and in October 2009, released OWL 2 (Motik et al. 2009).

While originally the NeOn Toolkit had its own realization of an ontology API, following the release of the OWL 2 specification, it switched to the (*Manchester*) *OWL API*[7], both to support the new language features and also to ensure compatibility with other Semantic Web tools and future developments. The OWL API (Horridge and Bechhofer 2009) is available under the open-source LGPL license and has emerged as the de facto standard for implementing OWL-based applications. It has an active user community and promises a high degree of standard compliance. It is the reference implementation of the OWL 2 Recommendation by the OWL working group.

13.3.3 Create Your Own Plugin

To facilitate the deployment of new plugins, we make use of the Eclipse Update mechanism which allows for deploying and updating new features. Features are a concept of Eclipse to represent a unit of useful and deployable functionalities. The role of features is to allow providers to make collections of plugins that logically go together.

[7] http://owlapi.sourcrforge.net

Developers interested in adding functionalities to the NeOn Toolkit can find more information on the NeOn Toolkit wiki, in the *developer's corner*[8]. There they will find references to our source code management system for the NeOn Toolkit core, as well as for many other plugins. The wiki also links to the developer's mailing list, where questions can be posed to the NeOn community.

As a central entry point for information about available plugins, we maintain a *plugin wiki*[9]. The purpose of the plugin wiki is to enable both developers and users to create and find information about components of the NeOn Toolkit. The plugin descriptions can include metadata, such as the developer and his/her affiliation, availability, license, etc., along with a brief description of the functionality of the plugin.

For the NeOn Toolkit, we have created a dedicated *NeOn Update Site* referenced in the NeOn Toolkit core. After a quality assurance procedure, newly available plugins are uploaded to the update site and thus become immediately accessible to all users of the Toolkit. When developing a plugin for NeOn, a developer can even set up his/her own update site, following the instructions from Eclipse.org[10].

13.4 Conclusions

In this chapter, we have described the main features of the core NeOn Toolkit and illustrated how the core functionality can be extended by downloading additional plugins. The Toolkit is an *open* platform to which anybody can contribute, and a number of resources are available for users and developers interested in the NeOn Toolkit:

- http://neon-toolkit.org: From here, users can download the latest release of the toolkit. Plugins are documented here, and developers find important information to get started.
- http://neon-project.org: This site contains information about the EU-funded NeOn Integrated Project with a lot of documents describing results of our research on networked ontologies, most of which has also been translated in functional code in the form of plugins for the NeOn Toolkit.
- http://www.neon-project.org/nw/NeOn_Movies: This contains a collection of tutorial videos explaining the main functionalities of the NeOn Toolkit plugins.
- http://www.neon-toolkit.org/mailman/listinfo/: This web page provides information about two mailing lists. One is intended for developers of NeOn plugins to discuss implementation questions. The other is meant for users of the

[8] http://neon-toolkit.org/wiki/Developer_Corner

[9] http://neon-toolkit.org/wiki/Neon_Plugins

[10] http://wiki.eclipse.org/FAQ_How_do_I_create_an_update_site_%28site.xml%29%3F

technology where we provide tips and answer questions with respect to the Toolkit and its plugins.

- Some participants of the NeOn Project have recently founded the NeOn Technology Foundation Inc[11]. to make sure that the development and distribution of the NeOn Toolkit as a free and open-source tool for the ontology community continues. Under the auspices of the Foundation, we will continue to improve the toolkit and make sure that it stays a competitive competitor in the market.

References

Beckett D, Berners-Lee T (2008) Turtle – Terse RDF triple language. W3C team submission 14 Jan 2008. http://www.w3.org/TeamSubmission/turtle/

Gruber TR (1993) A translation approach to portable ontology specifications. Knowl Acquis 5(2):199–220. http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92–71.html

Harth A (coordinator) (2010) NeOn deliverable D6.10.3 updated NeOn Toolkit plugins. NeOn deliverable. http://www.neon-project.org/nw/images/a/a2/NeOn_2010_D6103.pdf

Hitzler P, Krötzsch M, Parsia B, Patel-Schneider P, Rudolph S (eds) (2009) OWL 2 web ontology language: primer W3C recommendation. http://www.w3.org/TR/owl2-primer/

Horridge M, Bechhofer S (2009) The OWL API: a Java API for working with OWL 2 ontologies. In: Proceedings of OWLED 2009 – OWL: experiences and directions. 6th international workshop co-located with ISWC 2009, Chantilly, VA, USA. http://www.webont.org/owled/2009/papers/owled2009_submission_29.pdf

Horridge M, Patel-Schneider PF (2009) OWL 2 web ontology language: Manchester syntax W3C working group note. http://www.w3.org/TR/owl2-manchester-syntax/

Motik P, Patel-Schneider F, Parsia B (eds) (2009) OWL 2 web ontology language: structural specification and functional-style syntax. W3C recommendation. http://www.w3.org/TR/owl2-syntax/

OSGi Alliance (2009) OSGi service platform V4.0, core specification

Waterfeld W, Weiten M, Haase P (2007) D6.2.1 specification of NeOn reference architecture and NeOn APIs. NeOn deliverable. http://droz.dia.fi.upm.es/neon/servlet/download?ontology=Documentation+Ontology&concept=Deliverable&instanceSet=neon&instance=D6.2.1%3A+Specification+of+NeOn+reference+architecture+%26+NeOn+API-s&attribute=On-line+PDF+Version&value=NeOn_2007_D6.2.1.pdf

Waterfeld W, Weiten M, Haase P (2008a) Ontology management infrastructures. In: Hepp M, De Leenheer P, de Moor A (eds) Ontology management: semantic web, semantic web services, and business applications. Springer, New York

Waterfeld W, Erdmann M, Schweitzer T, Haase P (2008b) D6.9.1 specification of NeOn architecture and API V2. NeOn deliverable. Available at http://www.neon-project.org/web-content/images/Publications/neon_2008_d6.9.1.pdf

[11] http://www.neon-foundation.org/

Chapter 14
Scheduling Ontology Engineering Projects Using gOntt

Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Oscar Muñoz-García

Abstract In order to manage properly ontology development projects in complex settings and to apply correctly the NeOn Methodology, it is crucial to have knowledge of the entire ontology development life cycle before starting the development projects. The ontology project plan and scheduling helps the ontology development team to have this knowledge and to monitor the project execution. To facilitate the planning and scheduling of ontology development projects, the NeOn Toolkit plugin called gOntt has been developed. gOntt is a tool that supports the scheduling of ontology network development projects and helps to execute them. In addition, prescriptive methodological guidelines for scheduling ontology development projects using gOntt are provided.

14.1 Introduction

One of the crucial aspects within engineering processes is the issue of planning and scheduling development projects. These two terms are often thought of as synonymous; however, they are not. While planning[1] is the act of drawing up a plan, that is, a series of steps to be carried out to achieve an objective, scheduling[2] is defined as the activity of placing planned events along a timeline. Scheduling clearly depends on planning, and both are crucial in any project.

Bearing in mind that ontologies are part of software products and that sometimes ontologies are considered a kind of software, experiences and practices in software

[1] http://www.wordnet-online.com/planning.shtml

[2] http://www.wordnet-online.com/scheduling.shtml

M.C. Suárez-Figueroa (✉) • A. Gómez-Pérez • O. Muñoz-García
Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid,
Campus de Montegancedo sn., 28660 Boadilla del Monte, Madrid, Spain
e-mail: mcsuarez@fi.upm.es; asun@fi.upm.es; omunozgarcia@gmail.com

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_14, © Springer-Verlag Berlin Heidelberg 2012

engineering can be adopted and adjusted in the ontology engineering community. For this reason and with the aim of achieving in ontology engineering a similar degree of maturity to that of the software engineering field, we take as basis software engineering works to provide ontology engineers with help in planning and scheduling ontology development projects.

In software engineering, every development project has a life cycle (Taylor 2008), which is produced by instantiating a particular life cycle model. Life cycle models can be seen as abstractions of the phases or stages through which a product passes along its life. Examples of life cycle models are waterfall (Royce 1970), incremental (Sommerville 2007), iterative (Pfleeger 2001), evolutionary prototyping (Davis et al. 1988), and rapid throwaway prototyping (Davis et al. 1988).

To properly manage software development projects, it is crucial to have knowledge of the entire software development life cycle (Stellman and Greene 2005). In this regard, software engineers always plan and schedule every development project before starting it. The project plan defines the tasks to be carried out and establishes the human resources to perform the project work. To estimate the effort required to perform each task, techniques such as (Stellman and Greene 2005) Wideband Delphi, PROBE, and COCOMO II can be used.

The project schedule is a calendar that links the tasks to be performed with the resources to support their performance. One of the most common forms of representing schedules is to use a Gantt chart (Gantt 1974); and the most popular tool for creating a project schedule is Microsoft Project, according to (Stellman and Greene 2005). However, according to our knowledge, any tool for managing project schedules provides guidelines on how to execute the project.

Ontologies are used for making knowledge explicit and allowing it to be shared. One of the keys when building ontologies as in the case of software products is to plan and schedule the ontology development. However, in ontology engineering, planning and scheduling are still in their early stages. Only METHONTOLOGY (Fernández-López et al. 1997; Blázquez et al. 1998) defines the scheduling activity, but it does not provide guidelines for helping ontology developers to plan and schedule their projects. Other methodologies, such as On-To-Knowledge (Staab et al. 2001) and DILIGENT (Pinto et al. 2004), do not include these activities in their developments. Regarding the calculation of cost estimation of projects, the only technique available is ONTOCOM (Simperl et al. 2009), a model that predicts the costs of ontology development projects.

To cover this lack of methods and tools for planning and scheduling, this chapter describes (a) the gOntt plugin, a tool that supports the scheduling of ontology network development projects and helps to execute them, and (b) prescriptive methodological guidelines for scheduling ontology development projects using gOntt.

14.2 Scheduling Ontology Development Projects

Scheduling, as defined in the NeOn Glossary of Processes and Activities mentioned in Chap. 2, refers to the activity of identifying the different processes and activities to be performed during ontology development, their arrangement, and the time and resources needed for their completion. Thus, this activity includes as an important task the establishment of the *ontology network life cycle*, that is, the specific ordered sequence of processes and activities that ontology developers carry out during the life of the ontology network.

The goal of scheduling is to organize the different processes and activities in time, that is, to state a concrete programming or scheduling that guides the ontology network development, including processes and activities, their order and time, as well as human resource restrictions.

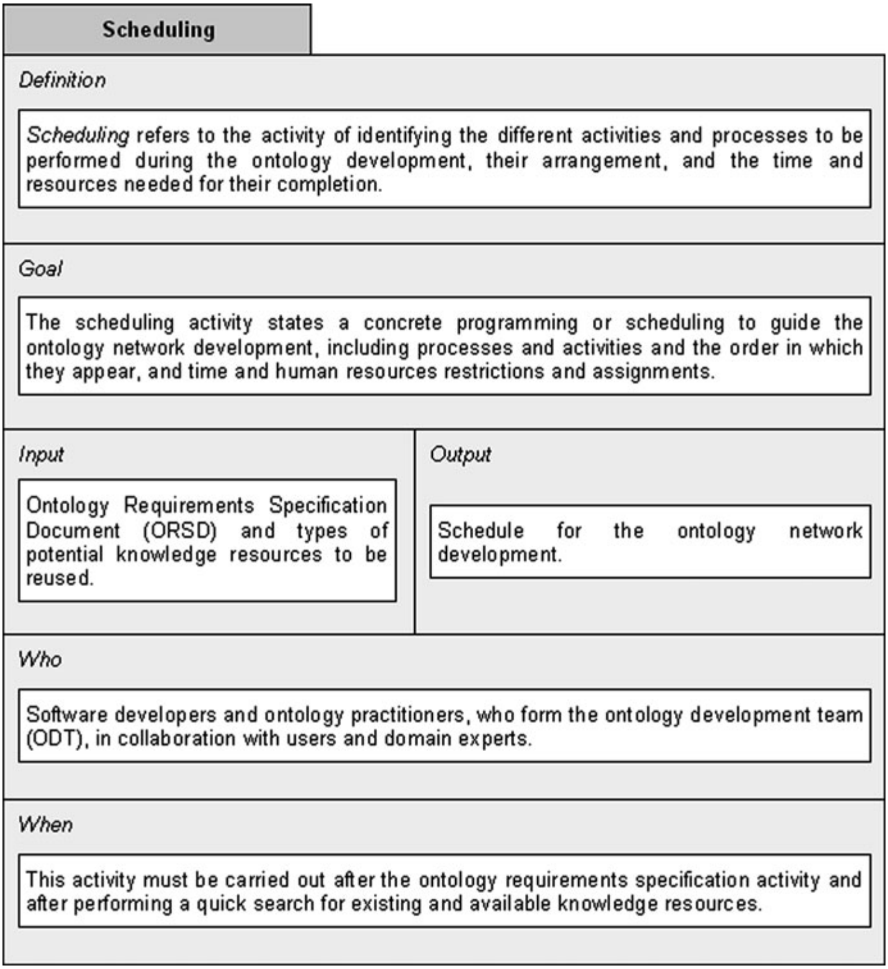
To establish the concrete schedule for the ontology network development, four important questions have to be answered:

1. How to organize an ontology development project into phases, or in other words, which ontology network life cycle model is the most appropriate for the ontology network development?
2. Which particular processes and activities should be carried out in the ontology network development?
3. Which order and dependencies exist among processes and activities?
4. What amount of resources (human and time) is needed and available for the development of the ontology network?

The first three questions are related to the establishment of the ontology network life cycle, and their responses would result in a general plan for the ontology network development. The fourth question is related to the inclusion of time and human resources restrictions for each process and activity included in the plan, and its response would result in the concrete schedule for the ontology network development. An estimate on how many people should be involved in the ontology network development can be obtained using the ONTOCOM model (Simperl et al. 2009). This is a cost estimation model, whose goal is to predict the costs (expressed in person per month) arising in typical ontology engineering processes.

As in the case of software engineering projects, an ontology engineer cannot start the ontology development project scheduling without having first identified the ontology requirements. In addition, the scheduling activity needs as input the types of potential knowledge resources (ontological resources, non-ontological resources, and/or ontology design patterns) to be reused during the development.

The *filling card* for the scheduling activity, presented in Fig. 14.1, includes the definition, goal, inputs and outputs, performer of the activity, and time of the activity.



Scheduling

Definition

Scheduling refers to the activity of identifying the different activities and processes to be performed during the ontology development, their arrangement, and the time and resources needed for their completion.

Goal

The scheduling activity states a concrete programming or scheduling to guide the ontology network development, including processes and activities and the order in which they appear, and time and human resources restrictions and assignments.

Input	*Output*
Ontology Requirements Specification Document (ORSD) and types of potential knowledge resources to be reused.	Schedule for the ontology network development.

Who

Software developers and ontology practitioners, who form the ontology development team (ODT), in collaboration with users and domain experts.

When

This activity must be carried out after the ontology requirements specification activity and after performing a quick search for existing and available knowledge resources.

Fig. 14.1 Scheduling filling card

14.3 gOntt: NeOn Toolkit Plugin for the Scheduling Activity

To support the scheduling of ontology development projects, the NeOn Toolkit provides a plugin called *gOntt*. This plugin is conceptually based on the following ingredients that are explained in Chap. 2: (a) the scenarios of the NeOn Methodology, (b) the set of ontology network life cycle models, and (c) the NeOn Glossary of Processes and Activities.

The gOntt plugin has the following main objectives:

1. To support ontology developers in the decision of which ontology network life cycle model is the most appropriate for building their ontologies
2. To help ontology developers in the decision of which concrete processes and activities should be carried out in the ontology network development and in which order
3. To instantiate the life cycle model selected and to create a particular life cycle for the ontology development with the processes and activities needed, including time restrictions on them
4. To inform ontology developers about how to carry out a particular process or activity through the NeOn methodological guidelines and a reference to the concrete NeOn plugins to be used – that is, to help ontology developers in the ontology project execution.

According to the aforementioned objectives, gOntt functionalities can be divided into two main groups: *functionalities for scheduling ontology development projects* and *functionalities for helping in the execution of ontology development projects*.

The functionalities for *scheduling an ontology network development* are:

- To create particular schedules from scratch, by allowing the ontology developer to include processes, activities, phases, and relationships, along with restrictions between them. Such processes and activities could either come from the NeOn Glossary of Processes and Activities or be new ones proposed by the developer.
- To create particular schedules in a guided way. gOntt creates preliminary plans for the ontology development with a simple two-step wizard. The ontology developer uses the wizard to answer a set of simple and intuitive questions that implicitly allow him to select the ontology life cycle model and the processes and activities to be carried out.
  - gOntt internally uses a set of heuristics based on methodological foundations and scheduling templates[3] (Suárez-Figueroa 2010) to automatically generate the initial plan.
  - gOntt provides the user with an initial plan in the form of a Gantt chart that the user can modify in the following fashion: (a) by including or deleting processes and activities, (b) by changing order and dependencies among processes and activities, and (c) by including resource assignments and restrictions to the planned processes and activities (this possibility is out of the scope of this chapter).
- To create, modify, and delete gOntt projects.
- To export and import gOntt projects in an interchange format based on XML (files with .got extension).

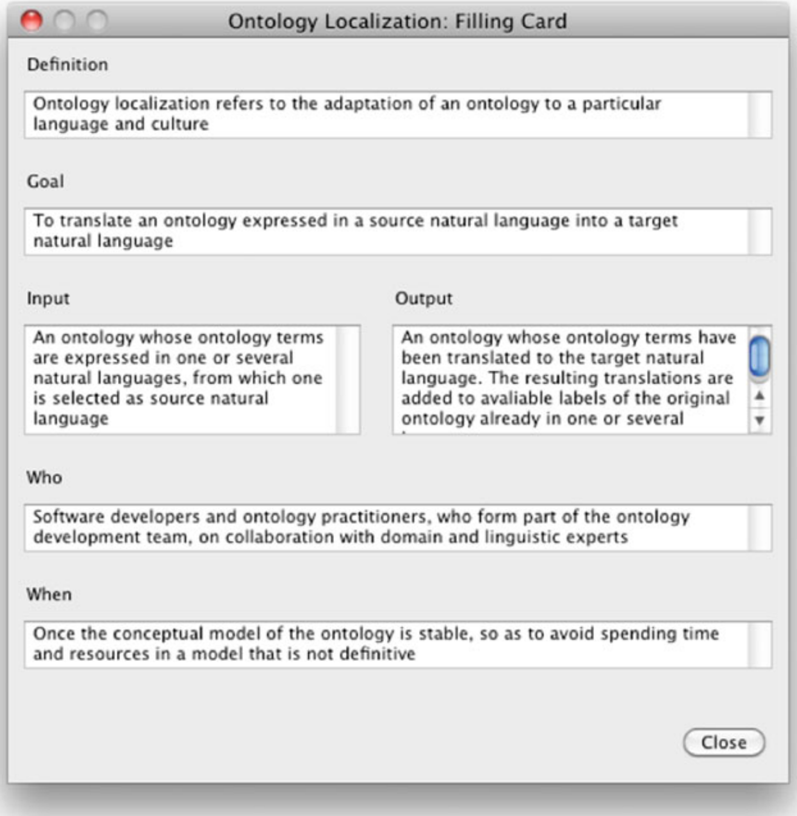
[3] Such scheduling templates show ontology project default plans based on the different and possible combinations among life cycle models, scenarios, and processes and activities.

- To provide graphical and textual visualizations of gOntt projects.
- To rename, reorder, and delete processes, activities, and phases from a gOntt project.
- To change the scope of a given process or activity (e.g., to change a given activity to a different phase).
- To create, modify, and delete connections between activities, between processes, and between activities and processes. If a connection exists between two elements, the latter cannot start until the former is completed. These connections can have two different meanings:
  - Logical dependencies: when it is required that one activity is carried out before another because of the nature of the activities (e.g., diagnosis before repair in ontology validation)
  - Temporal dependencies: when an activity should be performed after another because of project requirements (e.g., ontology reuse and non-ontological reuse can be carried out in parallel because they have no restrictions between them but, in some cases, there are not enough human resources to perform the activities in parallel, and so they should be set to perform in sequence)
- To include and modify the duration and the starting date of the processes, activities, and phases.
- To check gOntt projects with respect to logical and temporal constraints.
- To provide usual editing capabilities such as copy, paste, undo, and redo.

The functionalities for *helping in the execution of ontology development projects* are:

- To provide the developer with some methodological guidelines for the processes and activities identified in the NeOn Methodology, and thus
  - To display a *filling card*, which includes the process or activity definition, its goal, inputs and outputs, performer of the process or activity, and time of the performance. Figure 14.2 shows an example of the filling card for the ontology localization activity.
  - To display a *workflow* and some *methodological guidelines* explaining how the process or the activity should be carried out, including its inputs, outputs, and actors involved. Figure 14.3 shows an example of the methodological guidelines for the ontology localization activity. Workflows are implemented with Eclipse cheat sheets[4].

[4] Cheat sheets is a new emerging technology within Eclipse V3.0 that is meant to guide a developer through a series of complex tasks to achieve some overall goal. Some tasks can be performed automatically, such as launching the required tools for the user. Other tasks need to be completed manually by the user.



Ontology Localization: Filling Card

Definition

Ontology localization refers to the adaptation of an ontology to a particular language and culture

Goal

To translate an ontology expressed in a source natural language into a target natural language

Input

An ontology whose ontology terms are expressed in one or several natural languages, from which one is selected as source natural language

Output

An ontology whose ontology terms have been translated to the target natural language. The resulting translations are added to avaliable labels of the original ontology already in one or several

Who

Software developers and ontology practitioners, who form part of the ontology development team, on collaboration with domain and linguistic experts

When

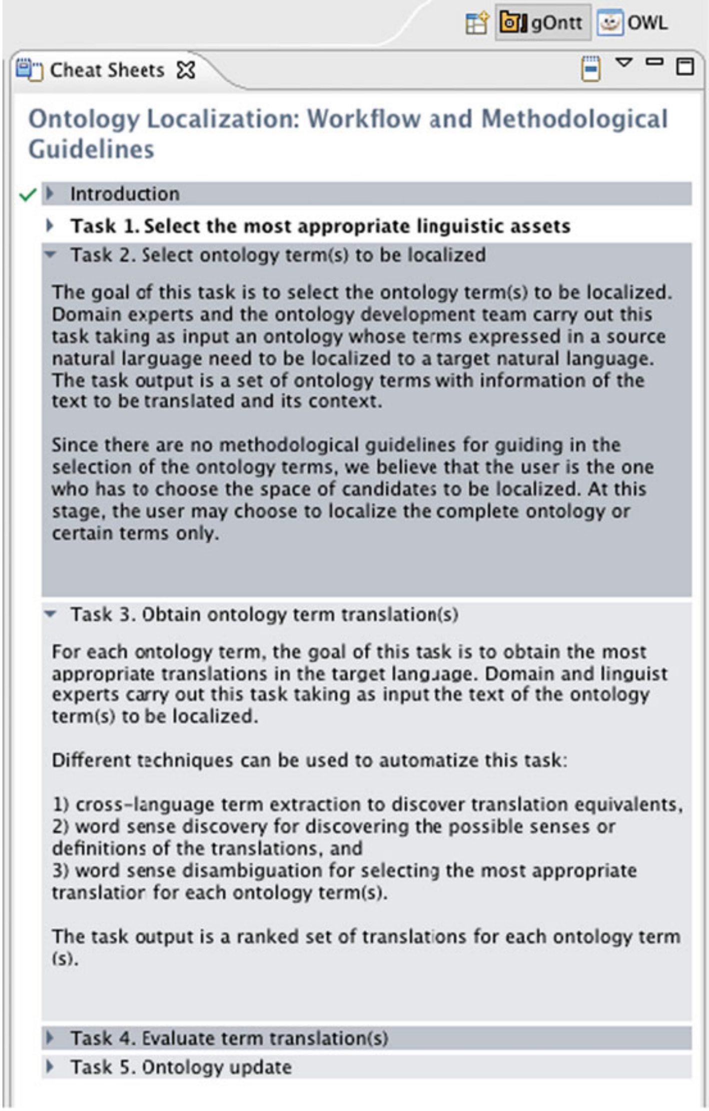
Once the conceptual model of the ontology is stable, so as to avoid spending time and resources in a model that is not definitive

Fig. 14.2 Example of the ontology localization filling card in gOntt

- To provide a direct access to the NeOn plugins associated to each process and activity planned. This means that gOntt triggers the different NeOn Toolkit plugins associated to each process or activity included in the plan. To make this possible, gOntt uses the so-called *extension points*[5].
- To launch the Kali-ma dashboard (described in Chap. 15) with the plugins that are related to a given activity, process, phase, or whole scheduling projects.

Figure 14.4 shows the general appearance of the gOntt plugin, whereas Fig. 14.5 shows a specific plan; as for the latter, it is worth mentioning that in the reuse phase, ontological and non-ontological resource reuses can be replicated for

[5] gOntt has its own extension point that the rest of NeOn Toolkit plugins should implement (see Suárez-Figueroa et al. 2010 for more detail).



Ontology Localization: Workflow and Methodological Guidelines

✓ Introduction

Task 1. Select the most appropriate linguistic assets

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:

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 translatior 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)

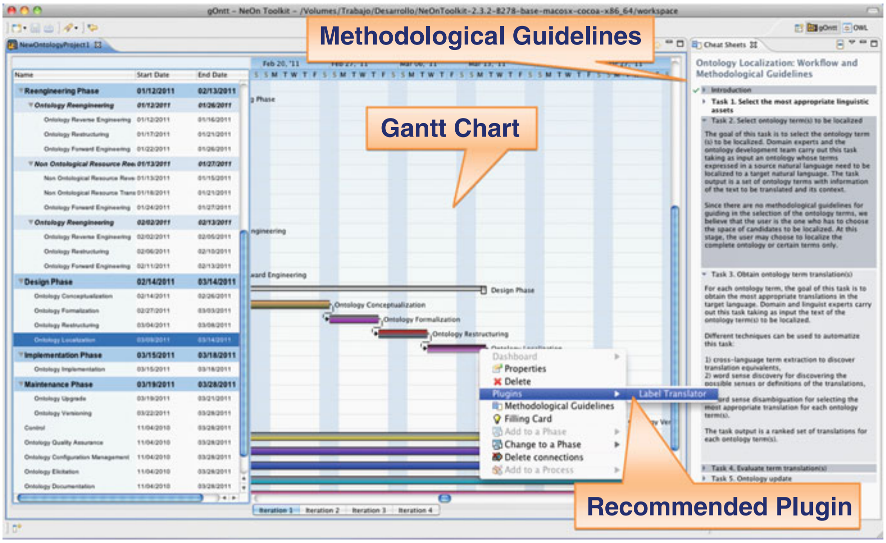
Task 5. Ontology update

Fig. 14.3 Example of the workflow and of some methodological guidelines for the ontology localization activity in gOntt

every single resource used in the ontology development; the same holds for the reengineering phase in the non-ontological resource reengineering process.

For each ontology network development project within the NeOn Toolkit, up to one gOntt scheduling can be created.

To create a new gOntt project, one of the following NeOn Toolkit new wizards must be launched (see Fig. 14.6): "Schedule a project from scratch"



Name	Start Date	End Date
Reengineering Phase	01/12/2011	02/13/2011
Ontology Reengineering	*01/12/2011*	*01/26/2011*
Ontology Reverse Engineering	01/12/2011	01/16/2011
Ontology Restructuring	01/17/2011	01/21/2011
Ontology Forward Engineering	01/22/2011	01/26/2011
Non Ontological Resource Ree	*01/13/2011*	*01/27/2011*
Non Ontological Resource Reve	01/13/2011	01/15/2011
Non Ontological Resource Trans	01/16/2011	01/21/2011
Ontology Forward Engineering	01/24/2011	01/27/2011
Ontology Reengineering	*02/02/2011*	*02/13/2011*
Ontology Reverse Engineering	02/02/2011	02/05/2011
Ontology Restructuring	02/06/2011	02/10/2011
Ontology Forward Engineering	02/11/2011	02/13/2011
Design Phase	02/14/2011	03/14/2011
Ontology Conceptualization	02/14/2011	02/26/2011
Ontology Formalization	02/27/2011	03/03/2011
Ontology Restructuring	03/04/2011	03/08/2011
Ontology Localization	03/09/2011	03/14/2011
Implementation Phase	03/15/2011	03/18/2011
Ontology Implementation	03/15/2011	03/18/2011
Maintenance Phase	03/19/2011	03/28/2011
Ontology Upgrade	03/19/2011	03/21/2011
Ontology Versioning	03/22/2011	03/28/2011
Control	11/04/2010	03/28/2011
Ontology Quality Assurance	11/04/2010	03/28/2011
Ontology Configuration Management	11/04/2010	03/28/2011
Ontology Elicitation	11/04/2010	03/28/2011
Ontology Documentation	11/04/2010	03/28/2011

Ontology Localization: Workflow and Methodological Guidelines

- Introduction
- Task 1. Select the most appropriate linguistic assets
- 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:

1) cross-language term extraction to discover translation equivalents,
2) word sense discovery for discovering the possible senses or definitions of the translations,
…ord 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)
- Task 5. Ontology update

Fig. 14.4 Screenshot of the gOntt plugin

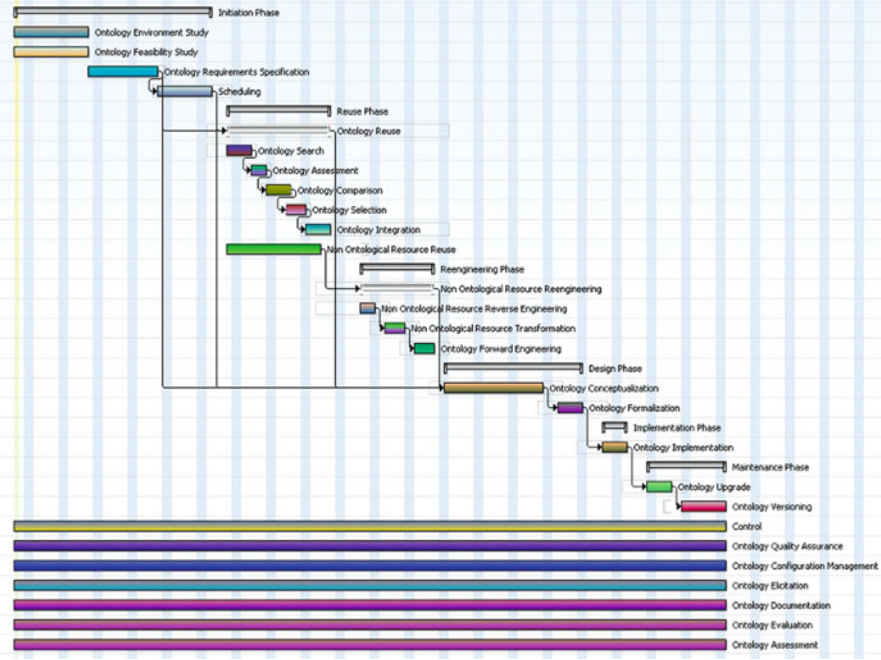


Fig. 14.5 A specific plan generated by gOntt

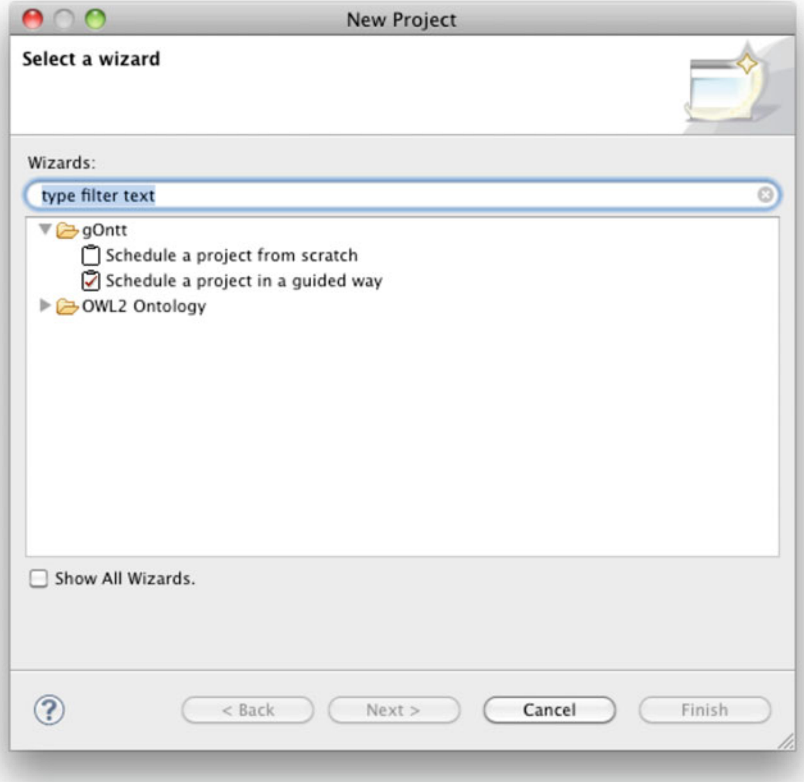


Fig. 14.6 gOntt wizards for scheduling new projects

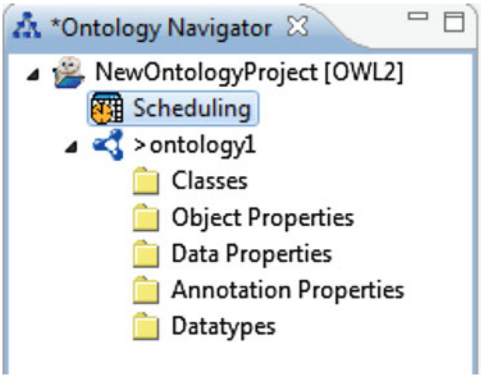


Fig. 14.7 Accessing a scheduling from the Ontology Navigator

or “Schedule a project in a guided way.” Then the wizard selected will ask about the ontology network development project that ontology developers want to schedule.

Once the scheduling has been created for a given project, ontology developers can access to the gOntt project by clicking on the “scheduling” item in the Ontology Navigator as Fig. 14.7 shows.

14.4 Guidelines for Scheduling Ontology Development Project Using gOntt

The *workflow* for carrying out the scheduling of ontology development projects using gOntt is presented in this section. Each of the tasks in the workflow proposed includes prescriptive methodological guidelines. The tasks for carrying out the scheduling activity are shown in Fig. 14.8 and are explained in detail below.

Task 1. Selecting the ontology network life cycle model. The goal of this task is to obtain the most appropriate ontology network life cycle model for the ontology network to be developed. Users, domain experts, and the ontology development team carry out this task, taking as input both the ontology requirement specification

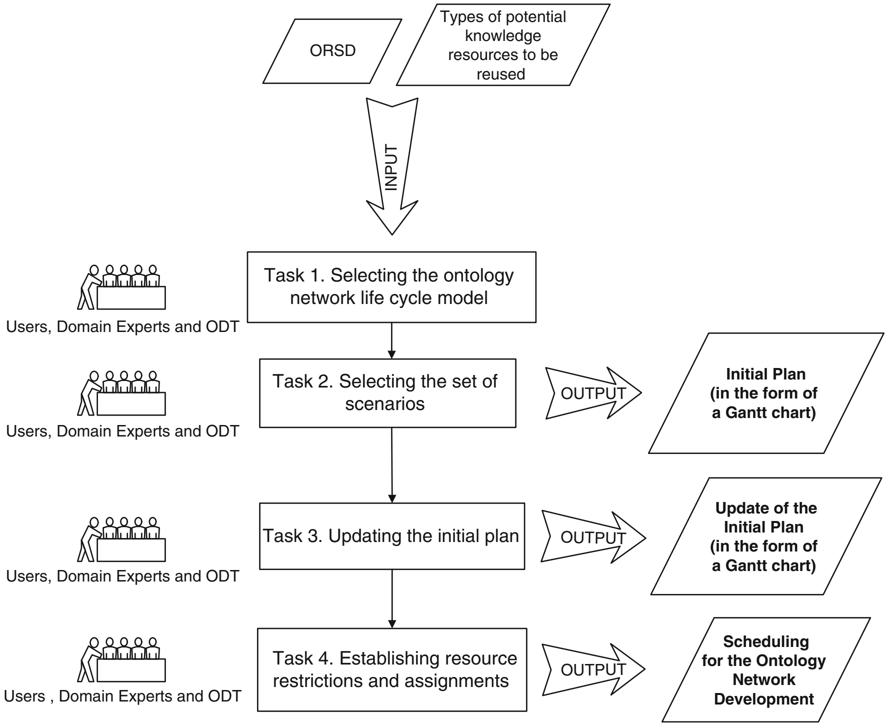
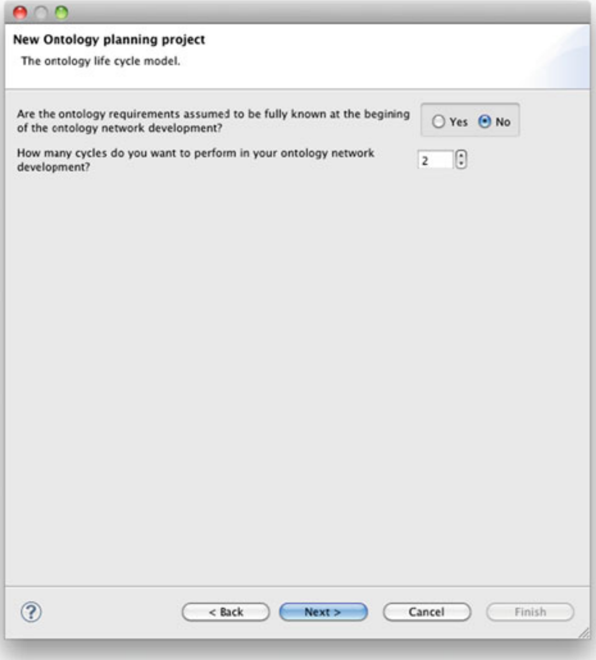

ORSD
Types of potential knowledge resources to be reused
INPUT
Task 1. Selecting the ontology network life cycle model
Users, Domain Experts and ODT
Task 2. Selecting the set of scenarios
Users, Domain Experts and ODT
OUTPUT
Initial Plan (in the form of a Gantt chart)
Task 3. Updating the initial plan
Users, Domain Experts and ODT
OUTPUT
Update of the Initial Plan (in the form of a Gantt chart)
Task 4. Establishing resource restrictions and assignments
Users , Domain Experts and ODT
OUTPUT
Scheduling for the Ontology Network Development

Fig. 14.8 Tasks for scheduling ontology development projects with gOntt



New Ontology planning project

The ontology life cycle model.

Are the ontology requirements assumed to be fully known at the begining of the ontology network development? ○ Yes ● No

How many cycles do you want to perform in your ontology network development? 2

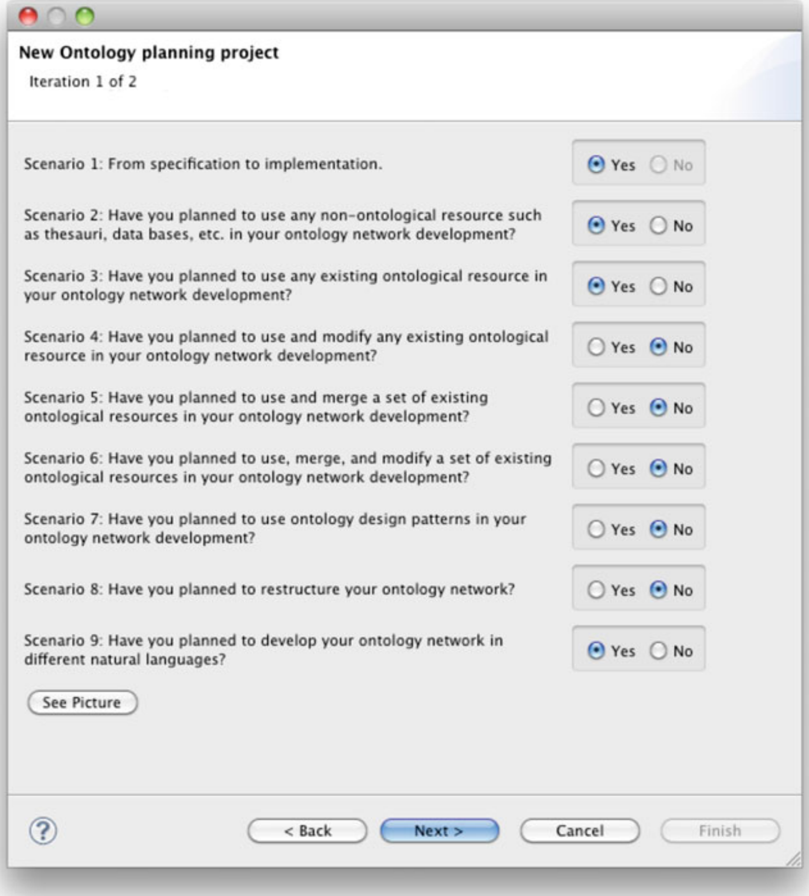
Fig. 14.9 Choosing the ontology network life cycle model

document (ORSD) (see Chap. 5) and the types of potential knowledge resources to be reused during the development. To help ontology developers in the decision of which is the most appropriate life cycle model among those presented in Chap. 2, gOntt presents a simple natural language question, displayed in Fig. 14.9. Based on the response given, either the waterfall model is selected, or the iterative-incremental model. In the latter case, ontology developers should also provide the expected number of iterations in the ontology development.

Task 2. Selecting the set of scenarios. The goal of this task is to select the set of scenarios to be followed during the ontology network development. Users, domain experts, and the ontology development team carry out this task, taking as input both the ontology requirement specification document (ORSD) and the set of potential knowledge resources to be used during the development.

To help ontology developers in this task, gOntt presents the set of natural language questions displayed in Fig. 14.10. If the model selected in Task 1 is the waterfall one, then questions should be answered once; on the other hand, if the model selected is the iterative-incremental one, then the set of questions should be answered once for each iteration expected.

With the responses to these questions, the set of heuristics based on methodological foundations and the set of scheduling templates (Suárez-Figueroa 2010),



New Ontology planning project

Iteration 1 of 2

Scenario 1: From specification to implementation. ● Yes ○ No

Scenario 2: Have you planned to use any non-ontological resource such as thesauri, data bases, etc. in your ontology network development? ● Yes ○ No

Scenario 3: Have you planned to use any existing ontological resource in your ontology network development? ● Yes ○ No

Scenario 4: Have you planned to use and modify any existing ontological resource in your ontology network development? ○ Yes ● No

Scenario 5: Have you planned to use and merge a set of existing ontological resources in your ontology network development? ○ Yes ● No

Scenario 6: Have you planned to use, merge, and modify a set of existing ontological resources in your ontology network development? ○ Yes ● No

Scenario 7: Have you planned to use ontology design patterns in your ontology network development? ○ Yes ● No

Scenario 8: Have you planned to restructure your ontology network? ○ Yes ● No

Scenario 9: Have you planned to develop your ontology network in different natural languages? ● Yes ○ No

Fig. 14.10 Selecting the scenarios

gOntt is able to obtain the initial ontology network life cycle, that is, an initial plan for the ontology network development. The task output is represented as a Gantt chart, which is *de facto standard* in software project management.

Task 3. Updating the initial plan. The goal of this task is to modify (if necessary) the initial plan presented by gOntt. Users, domain experts, and the ontology development team carry out this task, taking as input both the ontology requirement specification document (ORSD) and the set of potential knowledge resources to be used during the development.

Ontology developers can modify the initial plan in the following ways: (a) by including or deleting processes, activities, and model phases and (b) by changing order and dependencies among processes and activities.

Task 4. Establishing resource restrictions and assignments. The goal of this task is to include information about temporal scheduling and human resource assignments in the life cycle obtained in Task 3. Users, domain experts, and the ontology development team carry out this task, taking as input both the ontology requirement specification document (ORSD) and the set of potential knowledge resources to be used during the development.

14.5 Conclusions

In order to manage properly ontology development projects in complex settings and to apply the NeOn Methodology correctly, it is crucial to have knowledge of the entire ontology development life cycle before starting development. The ontology project plan defines the tasks to be executed, the time when the tasks will be executed, and the dependencies between tasks. The project plan is the only way, as can be shown in other disciplines, to commit people to the project and to show how the work will be performed. It also aids ontology engineers in monitoring project execution and assessing the impact of a particular delay in the planned tasks.

With these notions in mind, this chapter presented (a) the gOntt plugin, a tool that supports the scheduling of ontology networks developments as well as their execution, and (b) prescriptive methodological guidelines for scheduling ontology development projects using gOntt.

In relation to the work presented in this chapter, an integration of gOntt and the guidelines proposed with the ONTOCOM model (Simperl et al. 2009) is planned. Additionally, such works will be extended by providing details of the cost associated to carrying out a particular task in an ontology development project.

References

Blázquez M, Fernández-López M, García-Pinar JM, Gómez-Pérez A (1998) Building ontologies at the knowledge level using the ontology design environment. In: Gaines BR, Musen MA (eds) 11th international workshop on Knowledge Acquisition, Modeling and Management (KAW 1998), Banff, Canada, SHARE4:1–15

Davis AM, Bersoff EH, Comer ER (1988) A strategy for comparing alternative software development life cycle models. IEEE Trans Softw Eng 14–10:1453–1461

Fernández-López M, Gómez-Pérez A, Juristo N (1997) METHONTOLOGY: from ontological art towards ontological engineering. In: Spring symposium on ontological engineering of AAAI, Stanford University, Standord, CA, pp 33–40

Gantt HL (1974) Work, wages and profit, published by The Engineering Magazine. New York, 1910; republished as Work, wages and profits, Hive Publishing Company, Easton, PA, 1974, ISBN 0879600489

Pfleeger S (2001) Software engineering: theory and practice, 2nd edn. Prentice Hall, Upper Saddle River. ISBN 0-13-029049-1

Pinto HS, Tempich C, Staab S (2004) DILIGENT: towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies. In: López de

Mantaras R, Saitta L (eds) Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004). IOS Press, Valencia, Spain, pp 393–397, 22–27 Aug 2004. ISBN: 1-58603-452-9. ISSN: 0922-6389

Royce WW (1970) Managing the development of large software systems: concepts and techniques. In: Proceedings Western Electronic Show and Convention (WESCON), Los Angeles, 25–28 Aug 1970

Simperl E, Popov I, Bürger T (2009) ONTOCOM revisited: towards accurate cost predictions for ontology development projects. In: Proceedings of the European Semantic Web Conference 2009 (ESWC 2009), Heraklion, Greece, 20 May–04 June 2009

Sommerville I (2007) Software engineering, 8th edn. Addison-Wesley, Harlow/New York. ISBN 0-321-31379-8

Staab S, Schnurr HP, Studer R, Sure Y (2001) Knowledge processes and ontologies. IEEE Intell Syst 16(1):26–34

Stellman A, Greene J (2005) Applied software project management. ISBN: 0-596-00948-8. O'Reilly

Suárez-Figueroa MC (2010) NeOn Methodology for building ontology networks: specification, scheduling and reuse. PhD thesis, Universidad Politécnica de Madrid, España, June 2010. Available at http://oa.upm.es/3879/

Suárez-Figueroa MC, Gómez-Pérez A, Muñoz O (2010). NeOn deliverable. D5.3.3. gOntt plugin for scheduling ontology projects. NeOn project. Available at http://www.neon-project.org/nw/images/c/c1/NeOn_2010_D533.pdf

Taylor JC (2008) Project Scheduling and Cost Control: Planning, Monitoring and Controlling the Baseline. J Ross Publishing, ISBN: 9781932159110

Chapter 15
Customizing Your Interaction with Kali-ma

Alessandro Adamou and Valentina Presutti

Abstract This chapter presents the Kali-ma NeOn Toolkit plugin, which exploits the versatility of the C-ODO Light model to assist ontology engineers and project managers in locating, selecting, and accessing other plugins through a unified, shared interaction mode. Kali-ma offers reasoning methods for classifying and categorizing ontology design tools with a variety of criteria, including collaborative aspects of ontology engineering and activities that follow the NeOn Methodology. Furthermore, it provides means for storing selections of tools and associating them directly to development projects so that they can be shared and ported across systems involved in common engineering tasks. In order to boost Kali-ma support for third-party plugins, we are also offering an online service for the semiautomatic generation of C-ODO Light–based plugin descriptions.

A. Adamou (✉)
Semantic Technology Lab, Institute of Cognitive Sciences and Technologies (National Research Council – CNR), Via Nomentana 56, 00161 Rome, Italy

Department of Computer Science, Alma Mater Studiorum Università di Bologna, Mura Anteo Zamboni 7, 40126 Bologna, Italy
e-mail: alessandro.adamou@istc.cnr.it; adamou@cs.unibo.it

V. Presutti
Semantic Technology Lab, Institute of Cognitive Sciences and Technologies (National Research Council – CNR), Via Nomentana 56, 00161 Rome, Italy
e-mail: valentina.presutti@cnr.it

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_15, © Springer-Verlag Berlin Heidelberg 2012

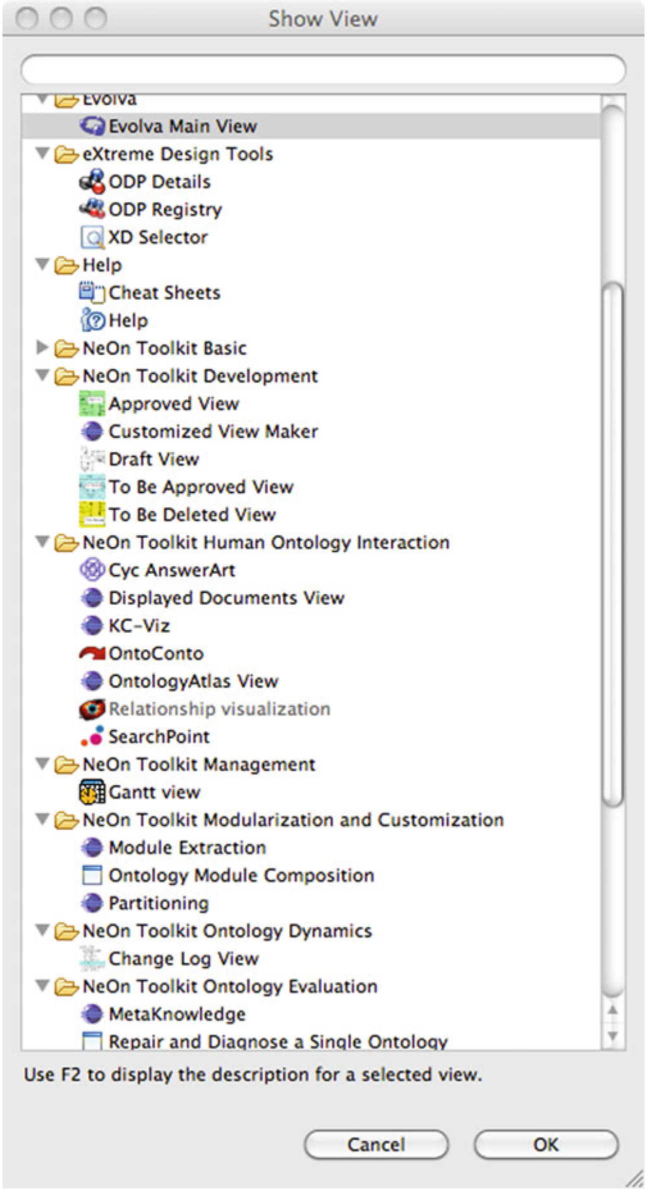
15.1 Introduction

Being an Eclipse RCP[1]-based ontology engineering platform, the NeOn Toolkit has an openly extensible feature set. Third parties may add custom functionalities in the form of software modules called *plugins*. Although the NeOn Toolkit provides its own set of specific extension points for manipulating ontology project hierarchies, most of those commonly provided by the Eclipse platform are supported. As with other Eclipse platforms, NeOn Toolkit plugins are maintained in dedicated repositories, called *update sites*. A pointer to one such update site, which was described in Chap. 13, is hardcoded in the core toolkit, but since the platform is open, anyone may set up their own update sites and use them as sources for additional plugins.

The RCP takes the burden of integrating plugins from the user interface perspective, e.g., by adding menus and toolbar buttons, populating the lists of views and perspectives, or adding new types of items that can be created via wizards. However, it is usually up to the developer to facilitate *conceptual* integration of her plugin by characterizing the goals of its features. The ways for doing so in the Eclipse platforms are little more than giving appropriate names and assigning categories to the UI contributions provided by their plugins. Performing this task can be tricky if the platform is supported by a large community, whose each member develops a plugin not knowing what others are doing. As a result, developers may arbitrarily add whatever categories, items, and labels they see fit for their plugins regardless of the rest. For example, two developers can create multiple categories for views, give them unique identifiers but label them both as *Visualization* independently on one another. As a result, end users will see two Visualization categories grouping different UI elements. Again, one developer could name a category after the plugin providing the corresponding UI elements, while another could name it after an arbitrarily named task supported by her plugin. In other words, when a contributor develops a plugin for the NeOn Toolkit, as well as for most plugin-based frameworks, she projects her own interpretation of the implicit metamodel of the user interface. Moreover, the uncontrolled proliferation of features (Damian and Chisan 2006) can clutter the user interface, e.g., if each plugin adds its own menu simply because no common agreement is reached as to which menus should be used for adding entries or submenus.

An instance of the scenarios described above is shown in Fig. 15.1. Here, a NeOn Toolkit user who has installed a large number of plugins from different sources is presented with this two-level tree list upon selecting the *Show View* menu entry. The names of views and the categories grouping them are widely varying in this example: some views, such as Evolva Main View, OntoConto, and SearchPoint, are named after the plugin that provides them; others, such as Partitioning, Relationship

[1] *Eclipse Rich Client Platform*, the software development toolkit originally written for the *Eclipse* integrated development environment (IDE).



Use F2 to display the description for a selected view.

Fig. 15.1 An example of a view selection menu in a very crowded NeOn Toolkit

Visualization, and Repair and Diagnose a Single Ontology, are named after the functionalities they provide; others, such as Gantt, refer to the structure of the view itself. Since there are no set rules about the naming of categories and interface components, there is no right or wrong with any of these rationales. However, because they come from different interpretations that each developer had of the user interface model, the overall picture may appear confused and cluttered. It is a goal of Kali-ma to try and bring some order into this confusion.

Kali-ma is a NeOn Toolkit plugin that aids developers and end users alike in creating a conceptually harmonized view on other known NeOn Toolkit plugins (and, more in general, tools that support the life cycle of ontologies). Kali-ma implements a user interface paradigm alternative to the Eclipse Workbench (and which can be switched with the latter in real time). This interface groups all UI contributions and access methods by the plugins issuing them and, with relatively little development effort, the plugins themselves by categories best representing the goals they are targeted at. It also adds a set of collaboration-oriented functionalities for end users, such as a metadata search feature, a whiteboard for executing dynamic plugin assemblies, and dedicated real-time chat support for ontology projects.

The remainder of this chapter provides an insight on the plugin as a whole, its functionalities, and the rationale behind them. Section 15.2 guides the reader through the plugin features and is structured so that the reader can concentrate on the section for end users (Sect. 15.2.1) or the one for developers (Sect. 15.2.2), depending on the reader's role. Developers are however advised to read both subsections in order to gain an understanding on the effects of their Kali-ma extensions on the interaction experience. Section 15.2.3 focuses on the underlying software architecture and how it combines standard components in Java with others in OWL (namely an extended version of the C-ODO Light ontology described in Chap. 4), thus being of interest for software engineers and ontology specialists alike.

15.2 Kali-ma Plugin Features

By the end of this section, the reader will have learned about the functionalities exposed by the Kali-ma plugin for facilitating interaction with and configuration of software components in the NeOn Toolkit. An insight is also provided, as well as documented, as to which steps the user needs to perform in order to activate and interact with these functionalities.

Although the Kali-ma plugin is oriented toward providing alternate modalities for end users to interact with the functionalities provided by the NeOn Toolkit, the rule body and several other aspects by which these modalities are provided are customizable. Some of such features are configurable at runtime by end users, while others are available by applying simple extensions to plugins by their respective developers. By this distinction, the remainder of this section is structured so as to allow a neat separation between functionalities that refer to end users for direct consumption and functionalities that refer to developers for their plugins to provide alternate interaction paths. In particular, the next section will also focus on what features can be configured by end users prior to launching the Kali-ma plugin on a running NeOn Toolkit platform.

15.2.1 Functionalities for End Users

When the Kali-ma plugin is activated, a desktop-integrated graphical user interface (GUI), called *Dashboard*, replaces the traditional Eclipse Workbench-based NeOn Toolkit interface. The constituents of this user interface, an example of which is shown in Fig. 15.2, are lightweight graphical elements, or *widgets*. A single widget represents either a built-in functionality provided by Kali-ma or a group of functionalities provided by some other NeOn Toolkit plugin.

Kali-ma provides a number of functionalities aimed at end users and aids them in the configuration of, and rapid access to, selected sets of tools apt for completing certain classes of tasks. These are as follows:

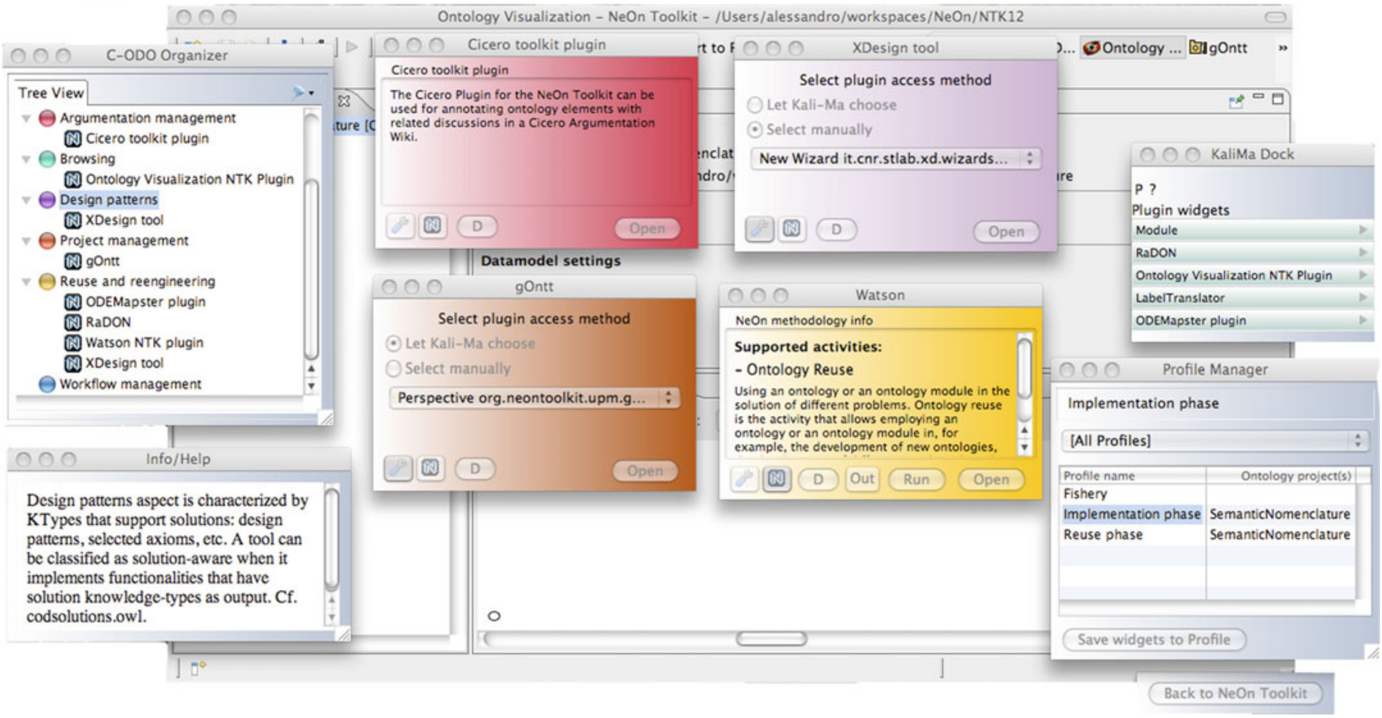
- *Tool organization and selection* based on preferred criteria.
- *Quick plugin access* that groups most functionalities of a plugin into a single widget.
- *Profile management* for bookmarking sets of plugins and associating them with ontology projects, thereby managing *profiles*.
- Project-based *real-time chat* that allows remote collaborating parties to share metadata of a common ontology project.
- *Advanced search* for ontology data and metadata.
- *Pipeline assembly*, for broadcasting the output of a plugin to other listening plugins in order to accomplish complex tasks.
- *Assistant*, for obtaining real-time guidance.

15.2.1.1 Preliminary Configuration

As with most NeOn Toolkit plugins, Kali-ma is configurable in several aspects concerning its way to handle interaction with the framework. While it does make sense to customize some of these aspects only once the Kali-ma dashboard has been activated, other features require prior configuration, as they affect the way dashboard elements are constructed. This section discusses the latter set of features and the steps to follow for configuring them.

Kali-ma comes with a "safe" default setup, in that all the plugin functionalities can be activated with no alteration of the default settings, granted an available internet connection. The only exception is the chat functionality, which requires the user to set the hostname of a Jabber/XMPP chat server where she has an account already registered.

All the settings of the Kali-ma plugin are grouped under a single *Kali-ma* entry in the *NeOn Toolkit Preferences* category. Remember that the *Preferences* panel



Design patterns aspect is characterized by KTypes that support solutions: design patterns, selected axioms, etc. A tool can be classified as solution-aware when it implements functionalities that have solution knowledge-types as output. Cf. codsolutions.owl.

The Cicero Plugin for the NeOn Toolkit can be used for annotating ontology elements with related discussions in a Cicero Argumentation Wiki.

Supported activities:

- Ontology Reuse

Using an ontology or an ontology module in the solution of different problems. Ontology reuse is the activity that allows employing an ontology or an ontology module in, for example, the development of new ontologies,

Profile name	Ontology project(s)
Fishery	
Implementation phase	SemanticNomenclature
Reuse phase	SemanticNomenclature

Fig. 15.2 The Kali-ma dashboard of widgets. Sorted by column, *top to bottom*, then *left to right*: the *codo organizer*; the *helper widget*; widgets representing the following plugins: Cicero, gOntt, XDesign Tools and Watson; the *dock* widget with placeholders for five more plugin widgets; the *profile manager*; the *switch* widget for returning to the NeOn Toolkit workbench

can be accessed in different ways, depending on the operating system used. For example, Windows users will find it in the *Window* top menu, while OS X users will find it in the *NeOn Toolkit* top menu.

Due to their intrinsic heterogeneity, the configuration parameters are in turn grouped into four categories:

1. *Appearance* is the category of customizable cosmetic aspects of the Kali-ma user interface.
   - *Open profiles docked* is an optional override for the docking options of each plugin widget in a user profile. When this option is checked, if the user opens a Kali-ma user profile, all of its plugin widgets will be minimized to the Kali-ma dock on startup, even if set otherwise in the profile itself. This option is preferable for users who wish to start with a dashboard as clear as possible.
   - *Widget background policy* determines what background color should be used for each plugin widget. Depending on the setting, the color can be either the one used for a category that classifies the plugin or one set by the user for that specific plugin.
2. *Network* deals with how Kali-ma exploits online resources. Currently, all the settings in this category are related to the built-in XMPP chat service.
   - *XMPP Host* and *Port* locate the resource where the XMPP messaging service is provided, e.g., for GTalk use Host `talk.google.com` and Port `5222`.
   - *XMPP Service name*, the identifier of the XMPP service on the host, if different from the host name, e.g., `jabber.org`.
   - *Multiuser chat service*, the identifier of the Multiuser Chat (MUC) service on the host, e.g., `conference.jabber.org`. Although not all XMPP-based services come with this functionality, this is required for the Kali-ma chat to work.
3. *Reasoning* enables the user to configure the parameters by which Kali-ma should locate and classify ontology design tools. These settings can have a significant impact on startup performance, but their default values are relatively safe on that respect. Note that changes to this configuration will only take effect the next time the Kali-ma dashboard is launched.
   - *Plugin address book location* is the physical URI of the ontology that indicates where the OWL descriptions of each plugin should be fetched from. Its default value is a plugin registry maintained by the Ontology Design Patterns portal[2] (Presutti et al. 2008).
   - *Criterion for tool classification* selects which property should be used as a criterion for classifying ontology design tools. Currently selectable criteria are *Design aspects, Processes and activities,* and *Design functionalities.*

[2] The Ontology Design Patterns portal, http://www.ontologydesignpatterns.org

- *Perform online update* denotes when Kali-ma should check for updates to the online plugin address book. Available options are "Each run," "Only on next run," and "Never." Note that if the address book has not been fetched yet (e.g., on the first run of Kali-ma ever), the update will be performed even if the "Never" option is set.
- *Cache plugin classification* indicates whether Kali-ma should materialize all inferences about plugins and store them into a local cache ontology. Because inferencing is a lengthy and highly CPU-intensive task, it is recommended to set this option unless major changes in the plugin registry occur. Note that this option only indicates whether the cache should be *built*, not whether it should be *used*: it will always be used if present. To force-rebuild the cache, the user can clear all the local data by clicking the *Clear now* button. This button is grayed out if there are no such local data.

4. *Toolkit integration* manages the way Kali-ma handles the standard NeOn Toolkit user interface along with its own. Users will configure these parameters according to their will to be provided with both interfaces altogether.

   - *Stick dashboard to main window*. If this option is set, the Kali-ma UI will appear on top of the standard NeOn Toolkit window, and its behavior will mimic the one of that window. Thus, when the NTK window is minimized, hidden, or maximized, so will be the Kali-ma widgets. Note that the Kali-ma dashboard is not modal; therefore, the NTK UI components in the background can still be interacted with.
   - *Main window behavior* allows the user to set how the main NTK window should appear or disappear when the Kali-ma dashboard is activated or deactivated. The user can opt for the main window to be hidden or minimized or neither. This option is only available when the "Stick dashboard to main window" option is unchecked.

   Example 15.1. This and all the examples in this chapter are based on a run-through scenario extracted from the case study described in Chap. 20. The Semantic Nomenclature of pharmaceutical products was carried out using the NeOn Methodology and related software support. Therefore, in order to use Kali-ma to carry out the activities specified in this methodology, an engineer will select *Processes and activities* from the *Reasoning → Criterion for tool classification* configuration panel.

15.2.1.2 Activating the Dashboard

Unlike most other NeOn Toolkit plugins, which support specific tasks in the engineering of networked ontologies and are therefore integrated with the platform, Kali-ma provides a GUI that runs in parallel with the standard one. For this reason, Kali-ma integration is limited to the preferences panel and the commands for activating its own user interface, called the *dashboard*. These commands are located:

- In the *Launch Dashboard* menu entry in the *Kali-ma* top menu
- In the NeOn Toolkit top bar as the *Launch Dashboard* button (an open perspective is required for displaying the button)

When one of these two actions is performed, the reasoning and plugin discovery tasks for preparing the dashboard are started as a background job. In particular, the following actions are performed:

1. The local tool descriptions and cache ontology are checked. If neither is present, or the online update parameter is set, plugin descriptions are fetched from the locations indicated in the online registry.
2. If variations between the local plugin ontology and the online registry are detected, the user is notified about these changes and prompted to choose whether to apply them or not. If changes are applied, any local cache is invalidated.
3. Plugins are classified by the designated criterion in one of the following ways:

   - If a valid local cache is present, it is queried directly.
   - If no valid cache is present but Kali-ma is configured to build one, it will first do so then query the cache it just built. This task is highly CPU intensive but will not have to be performed again as long as the cache remains valid.
   - If no valid cache is present and Kali-ma is *not* configured to build one, it will use a reasoner to classify plugins. This task is CPU intensive and will have to be run on every dashboard startup unless a cache is built.

4. The Kali-ma dashboard is activated and displayed in its default state. The NeOn Toolkit main window is hidden from view if set to do so.

Example 15.2. The project manager of the Semantic Nomenclature case study creates a new NeOn Toolkit ontology project called "SemanticNomenclature" and shares it with engineers using a version control tool such as CVS or Subversion. When the Dashboard is activated using the Launch Dashboard button, Kali-ma becomes aware of this project and can store profiles and configurations in its directory.

Recall that the dashboard is an aggregate of basic user interface components called *widgets*, whose look-and-feel exploits the capabilities offered by the GUI toolkit of the host operating system. Every widget identifies a functionality, or set of functionalities, in the NeOn Toolkit. Widgets can be grouped in two major categories: *native widgets* denote built-in interaction-oriented functionalities offered by the Kali-ma plugin itself and are always available regardless of what tools are installed on the platform; *plugin widgets* are representatives for plugins that are installed on the system, and they offer quick access to the functionalities available due to these plugins being installed. Widgets belonging to this latter category are available upon user request when the corresponding plugin is installed on the NeOn Toolkit platform, no matter what the canonical interaction paths to access them.

15.2.1.3 Organizing the Plugin Space

The heart of the Kali-ma approach for organizing the NeOn Toolkit as a functionality provider resides in the classification of its plugins by a unique, design-centered criterion that is nonetheless customizable. Therefore, its core functionality is to present end users with an overview of the plugins that are available in their running instance of the NTK and to help them select the one(s) whose coverage best suits the tasks that need to be performed.

The *C-ODO organizer* is the widget used for presenting this aggregate overview of plugins. This widget is named after C-ODO Light, the design ontology that is the base for all the classification criteria adopted by default in Kali-ma. Recall that an overview of the goal, rationale, and architecture of the C-ODO Light ontology network was given in Chap. 5.

The *C-ODO organizer* is the tool browser provided by Kali-ma. Users are free to choose from time to time, whether they wish to explore the plugin space as a tree or as a graph, by switching between the *Tree View* and the *Wheel View* tabs. The Tree View is organized as a simple *Category* → *Plugin* two-level tree; i.e., by expanding a category it is possible to view all and only the plugins that fall under that category. This also implies that a plugin that encompasses more than one category will appear as a child of multiple nodes in the taxonomy. The Wheel View, so called after the shape adopted by the category set, provides the same information in a graph. Although it takes up more space than the Tree View, it displays more useful information altogether. When a category is selected in the Wheel View, all and only the plugins under that category are displayed as in the Tree View. However, for each shown plugin, an edge appears for *every other* category it falls under.

The categories used for classifying plugins have a variable dependency on C-ODO Light, yet they are all based on this ontology for modeling the notion of an ontology design tool. The criterion used for identifying these categories can be selected from the *Reasoning* panel of the Kali-ma preferences (entry "Criterion for tool classification") prior to launching the plugin. The available criteria are as follows:

1. *Custom design functionalities*. These denote specific tasks and operations involved in the design of networked ontologies. They are arbitrarily defined by plugin developers, so the set of design functionalities can be highly fine grained, depending on the choices of developers. "Create project," "Cast vote," or "Delete annotation" are examples of such design functionalities. This criterion is enabled by selecting "*implements (Design Functionality)*" from the *Reasoning* preferences. End users should expect a sparse classification, with many

categories each with a limited number of plugins, yet with high redundancy across multiple categories, roughly one for each functionality implemented in that plugin.

2. *NeOn Methodology* refers to the fixed set of activities that are part of the NeOn Methodology canon as defined in Chap. 2. For this criterion, the categories are established a priori, and whether a plugin supports an activity in the methodology, this reflects the rationale used for selecting such plugins in gOntt (cf. Chap. 14). This criterion is enabled by selecting "*supports activity (Activity)*" from the *Reasoning* preferences.
3. *Ontology design aspects* is a limited, fixed set of generic design functionalities that aggregate the most common aspects of designing networked ontologies in a collaborative environment. The categories are set and very limited in order to provide dense classification of design tools. Also, it is the only case where the categories to which plugins belong are not explicitly defined but are instead obtained by inferencing over other features defined by the developers, namely the types of knowledge their plugins consume and produce. This criterion is enabled by selecting "*has aspect (Design Aspect)*" from the *Reasoning* preferences.

Both the Tree View and the Wheel View in the C-ODO organizer can be filtered by means of the funnel-shaped icon opposite the tabs. The filtering feature is due to the fact that the ABoxes describing ontology design tools, as well as their registries, are not bundled with the actual tools. In fact, they do not reside locally on the host platform in general but are instead exposed on the web. Moreover, they are not necessarily limited to NeOn Toolkit plugins but can span across several frameworks and architectural paradigms, such as plugins for other platforms, stand-alone applications, web applications, and web services.

Thus, three filters are available and can be cascaded: "Show only NeOn Toolkit plugins" will exclude all those design tools that, according to their ontological descriptions, do not qualify as plugins for the NeOn Toolkit. "Show only installed tools" will apply the previous filter and skim all the NeOn Toolkit plugins that are known to exist but are not detected as installed on the host platform. Finally, "Hide empty categories" will remove all the nodes representing categories to which no design tools are known to belong, regardless of the status of the other filters.

Example 15.3. The Semantic Nomenclature project manager has to select plugins for the implementation phase of the use case. The C-ODO organizer Tree View shows all the activities in the NeOn Methodology that come with software support. The *ODEMapster* plugin is selected (by double-clicking) from the "Non-Ontological Resource Reuse" activity, the OWLDoc plugin from the "Ontology Documentation" activity, the *Watson* plugin from the "Ontology Reuse" activity, and the *RaDON* plugin from the "Ontology Validation" activity. To create a schedule for all the activities to be performed in the phase, the *gOntt* plugin widget is also selected from the "Scheduling" activity. When each plugin is selected, its corresponding widget is displayed.

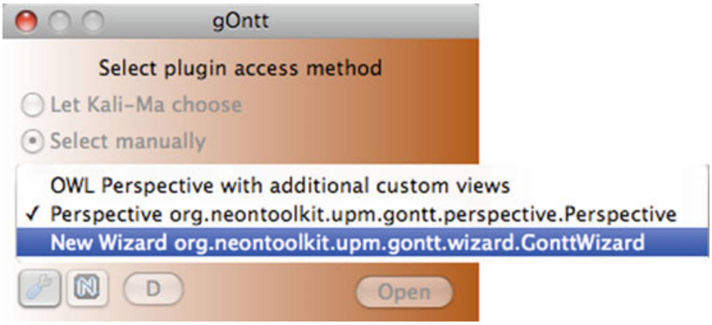
15.2.1.4 Interaction with Plugins

The standard mechanism by which a plugin is integrated with the Eclipse Rich Client Platform is by implementing *extension points*. An extension point allows a plugin to provide a contribution to the hosting platform, both on the functional level and on the user interface level[3]. The latter in particular includes a set of standard user interface objects that a plugin can implement to enrich the interactive experience with the platform. Some of them, such as wizards, views, or perspectives, can be stand-alone elements that can be displayed without any need for prior action upon other user interface or content items. For example, a wizard for exporting a given resource in a given format might depend on the user having previously selected the resource to export, but it might also allow the user to select that resource from a browser within the wizard instead. Conversely, other extension points contribute to the user interface by providing items that strictly depend on the interaction context. For example, context menu items will require the user to request a context menu on an item (typically by right-clicking on it). Therefore, running the action associated with a context menu item with no prior selection would make little sense and would in fact be unlikely to even work.

The current version of the Kali-ma plugin allows users to run NeOn Toolkit plugins through the following stand-alone access methods:

1. *Views* are single panels within the Eclipse workbench that serve as containers for arbitrary user interface controls. Multiple views can be aggregated in container objects, called Folders, which are essentially tabbed panes where each tab allows displaying one view at a time within the same folder. Views are usually associated to single-use cases, such as displaying the results of a SPARQL query, and can be manually moved across folders.
2. *Perspectives* are named composite panels that combine a group of folders and views in a predefined fashion. View combinations are usually associated to entire functionalities, which can be performed by interacting with the user interface elements in each view. Single views can only be shown within a perspective, and the NeOn Toolkit provides a default perspective for authoring OWL ontologies.
3. *New Wizards* are paged dialogs for guided creation operations. The list of available NewWizards in a system can be accessed from the "New" item in the "File" menu. Examples of this access method allow users to create ontology development projects, ontologies, and gOntt schedules. While we cannot rule out cases where new resources have to be created from existing ones (e.g., ontologies need to be created within an existing project), many New Wizards are associated to stand-alone use cases for creating new resources from scratch.

[3] http://wiki.eclipse.org/FAQ_What_are_extensions_and_extension_points%3F



gOntt

Select plugin access method

○ Let Kali-Ma choose

● Select manually

OWL Perspective with additional custom views
✓ Perspective org.neontoolkit.upm.gontt.perspective.Perspective
New Wizard org.neontoolkit.upm.gontt.wizard.GonttWizard
D
Open

Fig. 15.3 Access method selection for the gOntt plugin

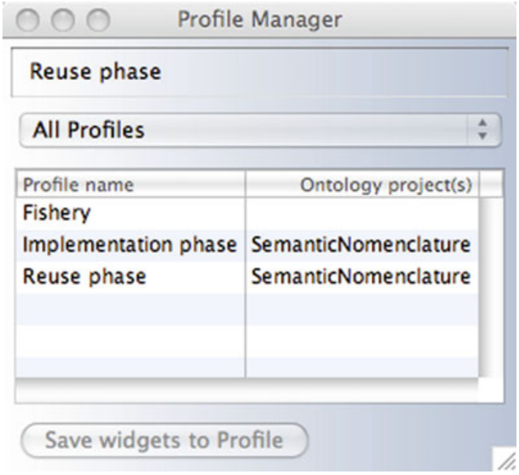
Figure 15.3 shows an example selection of access methods for the gOntt plugin (whose widget sports a white-to-rust gradient background, as this is the graphical feature assigned to the Project Management design aspect). The gOntt plugin contributes to the NeOn Toolkit by means of both a Perspective and a New Wizard for creating new schedules. A user can select either access method for launching the gOntt plugin once the “Open” button is clicked.

15.2.1.5 Profile Management

A selection of plugins to be displayed as widgets in the Kali-ma dashboard could be of much more use than simply assisting a single user during a single engineering session. If an open dashboard were just a volatile object that had to be manually rebuilt from scratch every time the NeOn Toolkit is restarted, not only would it be awkward to share in a collaborative context (which is assumed to be recurrent in NeOn-compliant ontology engineering), it would also discourage users and project managers from adopting Kali-ma to support medium- and long-term phases in an ontology engineering project.

In order to counter these preposterous potential shortcomings, Kali-ma offers a *profile management* functionality, which is concretely available as a native widget by its own right. The *Profile manager* widget, depicted in Fig. 15.4, allows users to store, open, and manage dashboard profiles.

A dashboard profile is essentially a named sorted set of plugins that can be serialized as an XML element and lives in the scope of both NeOn Toolkit workspaces and single ontology projects. Having performed a selection of plugins, all of which have a corresponding widget open in the Kali-ma dashboard, the user is able to retain this selection of plugins for sharing or future reuse. To do so, it is sufficient to type a name for the new profile in the top area of the widget and click the “Save widgets to Profile” button in the bottom area. This done, the current set of plugins is stored locally in the `kalima_profiles.xml` file in the workspace metadata directory for the Kali-ma plugin. Profiles can be listed, renamed, or deleted and one at a time can be set as active and displayed on screen by opening



Reuse phase

Profile name	Ontology project(s)
Fishery	
Implementation phase	SemanticNomenclature
Reuse phase	SemanticNomenclature

Fig. 15.4 Profile manager widget. Three profiles have been stored and are displayed in the profile table. Two of them (named *Implementation phase* and *Reuse phase*) are bound to the *Semantic Nomenclature* ontology project

the corresponding set of widgets. These operations are made available through context menu actions on the table occupying the middle portion of the widget.

Although dashboard profiles exist by their own right in a given NeOn Toolkit workspace, it is possible to bind them to one or more ontology development projects. This operation is also available as a context menu action, and its effects are visible on the second column of the table in the center of the widget, which displays the names of the ontology projects to which a profile is bound to. Binding a profile to one or more ontology projects results in saving a copy of that profile in another `kalima_profiles.xml` file, this time placed in the project directory. This action implies the ability to carry profiles along with a single project when it is exported to another system, as it is a common practice to share entire projects in Eclipse environments.

Example 15.4. The Semantic Nomenclature project manager wishes to share the tools for the Implementation phase selected earlier with all the ontology engineers who are set to perform each activity. A profile named “Implementation phase” is created and bound to the “SemanticNomenclature” ontology development project in the NeOn Toolkit. Because all participants are synchronized on this project, they will all get a copy of the new profile the next time they update their working copy of the project.

15.2.1.6 Dashboard Control and Docking

To counter the risks of ending up with a screen overcrowded by widgets, Kali-ma comes with an additional interface element called the *Dock*. As its name suggests, the Dock is conceptually inspired by a consolidated praxis in modern operating systems, which provide a user interface feature for quickly switching between applications. In our interpretation, the Kali-ma Dock provides a compact user interface for holding references to elements of the dashboard that are not of immediate interest, yet it still makes sense to hold in the current view of the system. For example, the user may want to remember having selected a certain plugin but does not need to access it in that particular instant. Every widget that supports docking comes with a toolbar button that, when clicked, instructs the dashboard controller to hide that widget and add a corresponding entry in the Kali-ma Dock. A dock entry is a very simple interface element that serves a placeholder for a docked widget. Each entry consists of a label with the plugin identifier and an arrow button for restoring the docked widget to its original position.

The Dock widget itself responds to the same screen overcrowding issue that holds for plugin widgets and other dashboard widgets; therefore, it is not visible on screen at all times. The Dock hides itself every time the last docked widget is restored (i.e., there are no more dock entries) and becomes visible again once a widget is docked (i.e., a dock entry is added). This is due to the fact that, at this stage, the Dock serves the sole purpose of holding references to widgets that are hidden from view. This behavior may vary as further functionalities are added to the Kali-ma Dock in the future.

15.2.1.7 Project-Based Real-Time Chat

Several phases of the articulated ontology life cycle management process are conceived with user collaboration in mind, and as such should they be carried out (Holsapple and Joshi 2002). Activities such as the collective argumentation of ontologies, or portions thereof, can be performed asynchronously, i.e., no different than by posting comments on message boards and the like. There may be cases, however, where multiple users collaborating on the same ontology project may require to coordinate their efforts in real-time, in order not to bottleneck one another. One such circumstance may involve two ontology engineers developing separate modules of an ontology network, whose entities need to be related via equivalence statements nonetheless. In such a situation, the user who needs to perform the alignment will need to know the name of the alignment target as soon as possible, and this can be significantly sped up by synchronous communication.

Kali-ma includes a lightweight real-time chat system to support synchronous communication in an environment where users can instantly share references to resources in a common ontology project. Through the *Chat widget*, a single user can join one or more dedicated virtual chat rooms, each named after an ontology project she has in common with other users. Additionally, for each project, it is possible to send the identifiers of any OWL entity loaded within that project with just a few keystrokes.

Example 15.5. The project manager and engineers that share the “SemanticNomenclature” project and have the same XMPP Chat configuration in the Kali-ma preferences will all be presented with an option to join the “SemanticNomenclature” chat room and discuss their engineering activities there.

As with other optional widgets, the Kali-ma chat interface can be activated by means of the Dock widget by simply clicking the balloon-shaped icon on its toolbar. In the default panel of this widget, it is sufficient for a user to type in her credentials (set by the chat server administrator), freely choose an alternate label, or *alias*, and log into the chat server. With this done, a combo box will display the list of available chat rooms, each named after an ontology project in her NeOn Toolkit workspace. Multiple chat rooms, one per project, can be joined at once, and a chat room will be seamlessly created on the fly if it has not yet been configured by another user. A user may send any free text message by simply typing it in a chat room window. However, if a reference to an OWL class, property, or individual needs to be broadcast to other users sharing the same project, it is sufficient to start typing in part of its name (not necessarily a prefix) and invoke the autocompletion key combination (usually `Ctrl` + `Space`) to select from a list of matching entities that exist *within that project*. Multiple OWL entity references can be broadcast in a single message by invoking autocompletion.

Any party is free to host a chat server compatible with Kali-ma. The plugin uses the open standard instant messaging protocol *XMPP* (Extensible Messaging and Presence Protocol)[4], which sports numerous compatible instant messaging clients as well as communication services (Google Talk[5] and Jabber[6] being two of them). Anyone can set up an off-the-shelf XMPP server on a host and create accounts for users, who can quickly configure Kali-ma on their clients (see Sect. 15.2.1.1) to instantly use it for relaying their messages.

15.2.1.8 Obtaining Help

Kali-ma provides its own real-time help system, aimed at displaying appropriate justification of each node appearing in the C-ODO Organizer taxonomy, and in doing so, to take advantage of any metadata present in the ontologies describing tools and classification criteria.

Real-time guidance is provided through the *Helper widget*. The Helper is essentially a lightweight web browser capable of rendering HTML. However, it also reacts to local events within the dashboard, such as a particular widget being focused or a node being selected in the C-ODO organizer. While help messages related to native functionalities are hardcoded, those deriving from metadata such as OWL annotations derive from elements of the ontological component of Kali-ma, which also include remote tool descriptions. For instance, when a node is selected

[4] XMPP, http://xmpp.org

[5] Google Talk, http://www.google.com/talk

[6] Jabber, originator of the initial XMPP design and implementation, http://www.jabber.org

that represents a design aspect, NeOn Methodology activity, functionality, or design tool, the *Helper widget* displays the `rdfs:comment` annotation for the corresponding OWL individual.

15.2.2 Functionalities for Plugin Developers

One goal of Kali-ma is to reorganize the plugin space under a single, shared criterion that can apply to the majority of plugins. To that end, it provides a set of functionalities to aid developers in describing the features of their plugins so that Kali-ma can elaborate on them and construct a single, harmonic view. These functionalities belong to the following categories:

- *Plugin description management* guides users throughout the creation of the ontology that describes how a plugin contributes to the life cycle management of ontologies.
- The *interoperability API* allows developers to launch and customize a Dashboard programmatically from the code of any plugin.

15.2.2.1 Plugin Description Management

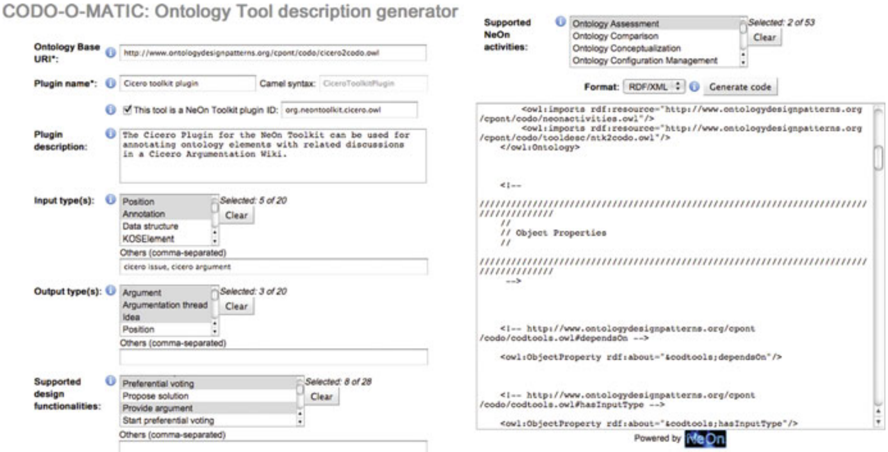
As will be presented in Sect. 15.2.3.2, the Kali-ma infrastructure includes a semantic layer involving components that are invariant in the domain of collaborative ontology engineering, as is the C-ODO Light network, and others that can be customized and adapted to new and refined taxonomies and criteria, such as the rules for categorizing the tool space. Standing amid these two levels are the real-world entities, i.e., the ABoxes where actual ontology design tools are instantiated and facts are provided for them. Kali-ma has no built-in or prior knowledge of which design tools exist, whether C-ODO Light–based ontologies describing them are provided and what physical URIs should be dereferenced for locating these descriptions. It does, however, provide a mechanism for locating such ontologies from a single, configurable source. Coupled with this mechanism, we are offering an online service for semiautomatic construction of C-ODO Light–based plugin descriptions. The next section details the key functional characteristics of both features mentioned above.

15.2.2.2 Plugin Description Generator

Knowledge of the ontology tool population is not delegated to a single online repository. It is the plugin provider's call to author pieces of structured knowledge concerning their own products; thus, it is reasonable to expect them to remain depositaries of this knowledge, while at the same time sharing it in an open environment such as Linked Data. Concurrently, it was felt convenient to have a system for aggregating references to these ontologies at disposal, rather than crawling the whole Semantic Web.

In an effort to meet both demands, an interactive tool for constructing these OWL tool manifests was devised as a service to be available anytime, anywhere. A working prototype of this service was released as C-ODO-o-matic (simply dubbed *Codomatic* throughout the remainder of the chapter)[7], its name paying homage to an inspiring online form for generating FOAF profiles. Codomatic is a simple, single-page Ajax application for constructing *C-ODO Light*–based OWL manifests of ontology design tools bearing the minimum set of axioms for allowing a DL reasoner to categorize the tool with respect to any of the three supported classification criteria explained in Sect. 15.2.1.3. The Codomatic service features willfully essential styling so as to keep it open to embedding within Wiki pages or web frames.

A sample of running Codomatic code generator for the Cicero argumentation plugin (Dellschaft et al. 2008), depicted in Fig. 15.5, shows what minimum user input is required and leveraged for the generation of the corresponding RDF code. The *Ontology base URI* field provides the default namespace to be used for any new entities asserted in the ontology to be generated and is advised to match the physical URI to be dereferenced for locating the ontology itself. The *Plugin name* field, along with its Camel syntax version, denote respectively the `rdfs:label` annotation and the actual URI local name for the OWL individual that identifies the tool itself, while the *Plugin description* field denote the English `rdfs:comment`


CODO-O-MATIC: Ontology Tool description generator

Ontology Base URI*: http://www.ontologydesignpatterns.org/cpont/codo/cicero2codo.owl

Plugin name*: Cicero toolkit plugin Camel syntax: CiceroToolkitPlugin

☑ This tool is a NeOn Toolkit plugin ID: org.neontoolkit.cicero.owl

Plugin description: The Cicero Plugin for the NeOn Toolkit can be used for annotating ontology elements with related discussions in a Cicero Argumentation Wiki.

Input type(s): Selected: 5 of 20

Others (comma-separated): cicero issue, cicero argument

Output type(s): Selected: 3 of 20

Others (comma-separated)

Supported design functionalities: Selected: 8 of 28

Others (comma-separated)

Supported NeOn activities: Selected: 2 of 53

Format: RDF/XML

```
        <owl:imports rdf:resource="http://www.ontologydesignpatterns.org
/cpont/codo/neonactivities.owl"/>
        <owl:imports rdf:resource="http://www.ontologydesignpatterns.org
/cpont/codo/tooldesc/ntk2codo.owl"/>
    </owl:Ontology>



    <!--
///////////////////////////////////////////////////////////////////////////////
///////////////
    //
    // Object Properties
    //
///////////////////////////////////////////////////////////////////////////////
///////////////
     -->




    <!-- http://www.ontologydesignpatterns.org/cpont
/codo/codtools.owl#dependsOn -->

    <owl:ObjectProperty rdf:about="&codtools;dependsOn"/>



    <!-- http://www.ontologydesignpatterns.org/cpont
/codo/codtools.owl#hasInputType -->

    <owl:ObjectProperty rdf:about="&codtools;hasInputType"/>
```

Fig. 15.5 The *Codomatic* tool description generator, after constructing the RDF code for an argumentation management plugin called Cicero

[7] At the time of writing, the service is hosted at http://wit.istc.cnr.it:8080/codomatic

annotation for that individual. It is possible to state the tool in question to be a NeOn Toolkit plugin, in which case its unique identifier must be supplied.

The list boxes that follow this field in the figure allow providers to include functional specifications of their tools: through these interface objects, it is possible to select an arbitrary number of knowledge types that the tool is known to consume as input or produce as output, as well as the design functionalities and NeOn processes and activities that it supports. For all fields but the NeOn processes and activities one, an additional text box is available, where the provider can arbitrarily instantiate new knowledge types and design functionalities, if the existing ones are felt to fall short of accuracy or completeness in describing the tool in question. However, while new design functionalities can immediately be exploited when classifying a set of tools with respect to them, new knowledge types cannot contribute to the rules for inferring supported design aspects, unless the providers include additional defined classes that are restricted on the `hasInputType` or `hasOutputType` properties for their new knowledge types.

The aforementioned statement supports the claim that by no means is Codomatic intended to serve as a replacement for a full-fledged OWL editor. The service is intended for the creation of minimal OWL manifests based on C-ODO Light, and yet it leaves room for extension and refinement. Providers can use the NeOn Toolkit OWL editor to add annotations for newly declared knowledge types and functionalities and relate them to existing ones where need be, as well as define additional rules for inferring supported design aspects from knowledge type statements.

The "Generate code" button triggers an asynchronous remote procedure call to a servlet that encapsulates submitted data and uses the same OWL API as Kali-ma's to output the corresponding ontology, whose source code is posted to the text area below the button. This code includes all the necessary ontology imports and is intended to be copied verbatim to an RDF document, which should then be uploaded to a location of the provider's choice. Codomatic does not pose restrictions to tool providers as to what physical locations should be used for their newly generated ontologies, nor does it store submitted base URIs or any other information used for generating the OWL code. References to physical locations can be submitted through the corresponding plugin pages on the NeOn Toolkit Wiki, as documented in its plugin development and submission guide[8]. Being a Semantic Media Wiki, it is then possible to export these references in RDF format for Kali-ma to consume.

[8] http://neon-toolkit.org/wiki/Plugin_HowTo

15.2.2.3 Interoperability API

In addition to supporting collaboration and interaction between end users, Kali-ma as a plugin comes with additional developer features that allow other ontology plugins to interoperate either with each other or with Kali-ma. There are two distinct methods of allowing *programmatic interoperability*, which is achieved through simple direct intervention on the plugin code. These two methods cover separate interoperability aspects and can be implemented independently. They are:

1. Construction of *Pipeline assemblies* within widgets, for executing plugin functionalities without switching to the plugin user interface for that plugin
2. External *Dashboard control*, for manipulating the contents of the Dashboard

Interoperability between plugins is achieved by construction of *pipeline assemblies*, which are dynamic software structures where the output of one component can be concatenated to one or more other components in the assembly in order to execute complex computational tasks. For instance, a design pattern selection service exposed by the *eXtreme Design* plugin (described in Chap. 3) could reuse the output axioms of a search issued using the *Watson* plugin (described in Chap. 7) in order to perform query expansion for broader pattern selection. Because the process has no strict coupling at build time, such a scenario can be realized without either plugin knowing a priori which other plugin it should expect its input from, or which one should accept its output.

The other supported interoperability aspect is *Dashboard control*, i.e., the programmatic manipulation of the Kali-ma user interface. This allows other developers to construct custom dashboard configurations specific for the engineering activity supported by another tool. Among NeOn Toolkit plugins, the *gOntt* tool for project scheduling supports Kali-ma dashboard interoperability, as it is possible from within a gOntt schedule to launch a Kali-ma dashboard containing widgets for all registered NeOn Toolkit plugins that support a given process, activity, or phase in that schedule. This support is among the features showcased by the gOntt plugin description in Chap. 14.

To reach either level of interoperability, a plugin must implement a simple Java API exposed by Kali-ma itself. A developer who wishes a plugin functionality to be directly called via its dashboard widget will simply have to implement an Eclipse *extension point*, which is mapped to a simple Java interface, both provided by Kali-ma. The developer will simply have to wrap a call to a plugin functionality into a Java class that implements this interface and annotate the single public method with the types of the parameters expected to be consumed and produced by that functionality.

The Dashboard control API is also simple and straightforward. It is enough for a developer to invoke any static method of the `DashboardLauncher` class exposed by the Kali-ma API, and a dashboard will be launched, containing widgets for all the available plugins whose identifiers were passed as parameters. This implementation may occur in a separate plugin, without any intervention on the original plugin code.

15.2.3 Architectural Design

The software architecture of the Kali-ma plugin, used for performing semantic reorganization of the tool space, incorporates both procedural and logical components. That is, although the plugin is essentially a Java program (or, to be more precise, a set of OSGi bundles) like most other plugins, some functionalities are not entirely encoded as procedures in the plugin code but instead rely on formal semantics that describe their behavior. Although the entire knowledge needed for managing the tool space is maintained in its original OWL formalism, this is treated in a similar fashion as runtime software libraries. Ontologies that describe the domain model, plugin space, and classification criteria are dynamically aggregated and linked at runtime.

The sections that follow provide an insight on the software architecture of Kali-ma. After a quick overview on the next section, Sect. 15.2.3.2 describes the actual ontology network used by the tool. Section 15.2.3.3 describes the software modules that handle and reason upon the ontology network in order to classify NeOn Toolkit plugins. Finally, Sect. 15.2.3.4 provides a quick insight as to how the result is presented to the user.

15.2.3.1 Basic Software Architecture

The heterogeneous representation of the Kali-ma components, as well as the openness to possibly reusing the procedural components in engineering fields other than ontologies, imply a layered infrastructure of the tool. This infrastructure can be seen as split into three major components as depicted in Fig. 15.6: the *ontological component* is responsible for providing Kali-ma with the necessary knowledge about existing NTK plugins and the rules by which to classify them; the *reasoning component* manages the extraction of such knowledge from the ontological component, as well as the aggregation and classification of plugins; and lastly, the *presentation component* generates the widgets and handles communication between Kali-ma, NTK plugins, and the NTK core.

15.2.3.2 Ontological Component

The ontological component, encoded in its entirety in OWL, is at the lowest level of the stack. It is itself a layered subsystem, as the dependencies between its modules are acyclic. The component as a whole can be seen as a large networked ontology, although only the essential logical infrastructure is hardwired, whereas expert ontology engineers can define categorization rules without an exhaustive knowledge of the tool space, while leaving plugin contributors the liberty to author descriptions for their tools and host them wherever they see fit.

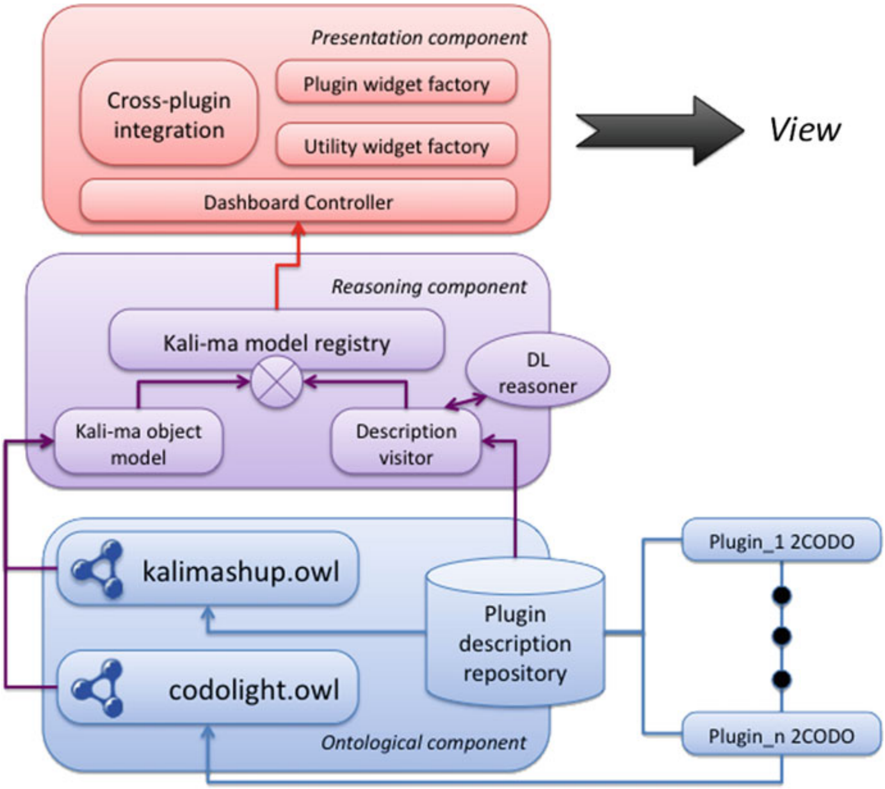


Fig. 15.6 The Kali-ma infrastructure and its main components

The layers of the Kali-ma ontological component include:

- A *foundational/domain layer*, which is essentially the *codolight* model for ontology design (described in Chap. 5). Its modularity and distinctive support for collaborative life cycle management make it easily extensible with specialized classes, additional primitives and rules, without any need for tainting the whole model. What's more, it is aligned to several widely used ontologies describing the Semantic Web for computational and social interoperability.
- Sets of *categorization rules*, where the classification criteria used for providing a taxonomy of tools in the Organizer widget are formalized. Recall that the default criteria are implementation of *design functionalities*, support for NeOn Methodology *processes and activities*, and coverage of collaborative *design aspects*. While applications of the first two criteria can be usually extracted without resorting to reasoning tasks, inference will be required in order to determine which design aspects are covered by a plugin.
- The *tool space population* model which includes C-ODO Light–based OWL descriptions of NeOn Toolkit plugins, each describing what types of task a certain plugin can help accomplish, what types of knowledge representation it

can handle, and so on. These are not hardwired in the built-in portion of the ontological component and can be located anywhere on the web. Recall from Sect. 15.2.2.2 that an online service is available for the automatic generation of these tool descriptions.

15.2.3.3 Reasoning Component

In avoidance of the unwise practice of allowing the presentation component to handle the knowledge base straight away, Kali-ma implements a dedicated subsystem for extracting relevant knowledge. The ontological component provides such knowledge that the reasoning component wraps into a Java object model, which can then be accessed from the Dashboard controller in the presentation subsystem. This lower-level component in the Kali-ma software architecture, and the intermediate layer in the whole infrastructure, provides a software counterpart to the ontological component.

The *reasoning component* comprises the following modules:

- The *Kali-ma object model* represents parts of the C-ODO Light network, along with attached ontologies with additional categorization rules, in the form of Java types. This model includes interfaces for design tool, knowledge type, NeOn activity and design aspect OWL classes, and for generic annotated entities, whose RDFS label and comment annotations are deemed significant in the context of Kali-ma (i.e., they are presented to end users).
- The *description visitor* is responsible for instantiating the object model mentioned above from the ABoxes supplied by C-ODO Light–based plugin descriptions and classification rule ontologies. This module includes monitorable operations for initializing OWL managers and DL reasoners (both supplied by external packages), loading them with fixed and user-defined ontologies and querying them. This system can be configured to manage a cache; thus, it does not necessarily query the DL reasoner on each Kali-ma run.
- A *model registry* is where the instantiated object model is stored and kept track of. It stores wrapped OWL individuals and relationships between them and allows changes to the model to be monitored through its own event system. The model registry is ephemeral and does not need to be serialized, as it can be completely rebuilt at runtime from the ontological component in reasonable time.

Through the components of this subsystem, Kali-ma becomes aware of what NeOn Toolkit plugins are known and/or installed in the running system, what are the relevant relations in ontology design, and which of them are supported by collected plugins. The Kali-ma application logic has no prior knowledge of such relationships.

15.2.3.4 Presentation Component

The top-level component of the Kali-ma infrastructure, called *presentation component*, implements both the user interface and its controller in the *Model-View-Controller* (MVC) paradigm (Reenskaug 1979). This element is responsible for leveraging the underlying C-ODO Light–based object model and presenting the outcome of reasoning tasks performed thereupon. Widget factories, dashboard management, and event handling support all belong to this component. Once generated, widgets are deployed on the target view (typically, the operating system desktop) and integrated among other operating system windows.

References

Damian D, Chisan J (2006) An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Trans Softw Eng 32(7):433–453

Dellschaft K, Engelbrecht H, Barreto JM, Rutenbeck S, Staab S (2008) Cicero: tracking design rationale in collaborative ontology engineering. In: Bechhofer S, Hauswirth M, Hoffmann J, Koubarakis M (eds) ESWC, Lecture notes in computer science, vol 5021. Springer, Berlin/Heidelberg/New York, pp 782–786

Holsapple CW, Joshi KD (2002) A collaborative approach to ontology design. Commun ACM 45:42–47

Presutti V, Gangemi A, David S, de Cea GA, Suárez-Figueroa MC, Montiel-Ponsoda E, Poveda M (2008) A library of ontology design patterns: reusable solutions for collaborative design of networked ontologies. Deliverable D2.5.1, NeOn project

Reenskaug T (1979) Models – views – controllers. Technical report, Technical note, Xerox Parc

Chapter 16
Visualizing and Navigating Ontologies with KC-Viz

Enrico Motta, Silvio Peroni, José Manuel Gómez-Pérez, Mathieu d'Aquin, and Ning Li

Abstract There is empirical evidence that current user interfaces for ontology engineering are still inadequate in their ability to reduce task complexity for users, especially non-expert ones. Here we present a novel tool for visualizing and navigating ontologies, *KC-Viz*, which exploits an innovative ontology summarization method to support a "middle-out ontology browsing" approach, where it becomes possible to navigate ontologies starting from the most information-rich nodes (i.e., *key concepts*). This approach is similar to map-based visualization and navigation in geographical information systems, where, e.g., major cities are displayed more prominently than others, depending on the current level of granularity. Building on its powerful and empirically validated ontology summarization algorithm, KC-Viz provides a rich set of navigation and visualization mechanisms, including flexible *zooming* into and *hiding* of specific parts of an ontology, visualization of the most *salient* nodes, *history* browsing, saving and loading of customized ontology views, as well as essential interface support, such as graphical zooming, font manipulation, tree layout customization, and other functionalities.

E. Motta (✉) • M. d'Aquin • N. Li
Knowledge Media Institute (KMi), The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK
e-mail: e.motta@open.ac.uk; m.daquin@open.ac.uk; N.Li@open.ac.uk

S. Peroni
Department of Computer Science, University of Bologna, Bologna, Italy
e-mail: speroni@cs.unibo.it

J.M. Gómez-Pérez
Intelligent Software Components (iSOCO), S.A. Avda. del Partenón, 16-18, 28042 Madrid, Spain
e-mail: jmgomez@isoco.com

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_16, © Springer-Verlag Berlin Heidelberg 2012

16.1 Introduction

A key component of the Semantic Web is provided by the large number of ontologies available online. Given such large-scale availability of ontologies, ontology reuse is becoming more common, and tools, such as the Watson plugin for the NeOn Toolkit (d'Aquin et al. 2008), are now available, which facilitate the task of locating and directly reusing ontologies or ontology fragments. In this reuse-centric context, it is highly desirable to have mechanisms that can efficiently help users in making sense of the content of an ontology, e.g., in the context of having to make a decision about whether an ontology retrieved online is suitable for a particular set of requirements. However, the empirical studies carried out in the NeOn project (Dzbor et al. 2006) have shown that the visualization and navigation facilities available in today's ontology engineering environments do not necessarily provide effective support for making sense of and effectively exploring ontologies, and often end up hindering rather than helping users. These studies show that this is a problem, especially for non-expert users.

To address this issue, we have developed a novel tool for visualizing and navigating ontologies, called *KC-Viz*, which has been realized as a plugin for the NeOn Toolkit. KC-Viz exploits automatically created ontology summaries, based on the idea of *key concepts* (Peroni et al. 2008), to facilitate the task of making sense of large ontologies. In addition, it also provides a rich set of navigation and visualization mechanisms, including flexible *zooming* into and *hiding* of specific parts of an ontology, visualization of the most *salient* nodes, *history* browsing, saving and loading of customized *ontology views*, as well as essential interface customization support, such as graphical zooming, font manipulation, tree layout customization, and other functionalities.

In this chapter, we present a description of the main functionalities provided by KC-Viz, and we show how it attempts to address some of the limitations of current tools for ontology engineering.

16.2 Limitations of Top-Down Approaches to Navigating Ontologies

16.2.1 Ontology Sensemaking

Throughout this chapter, we will use as an illustrative example a version (v2.4) of the SmartProducts ontology[1], which is being developed in the course of the EU-funded SmartProducts project[2]. The aim of this ontology is to support the

[1] This network of ontologies can be downloaded from http://projects.kmi.open.ac.uk/smartproducts/ontologies/SP_v2_4.zip

[2] http://www.smartproducts-project.eu

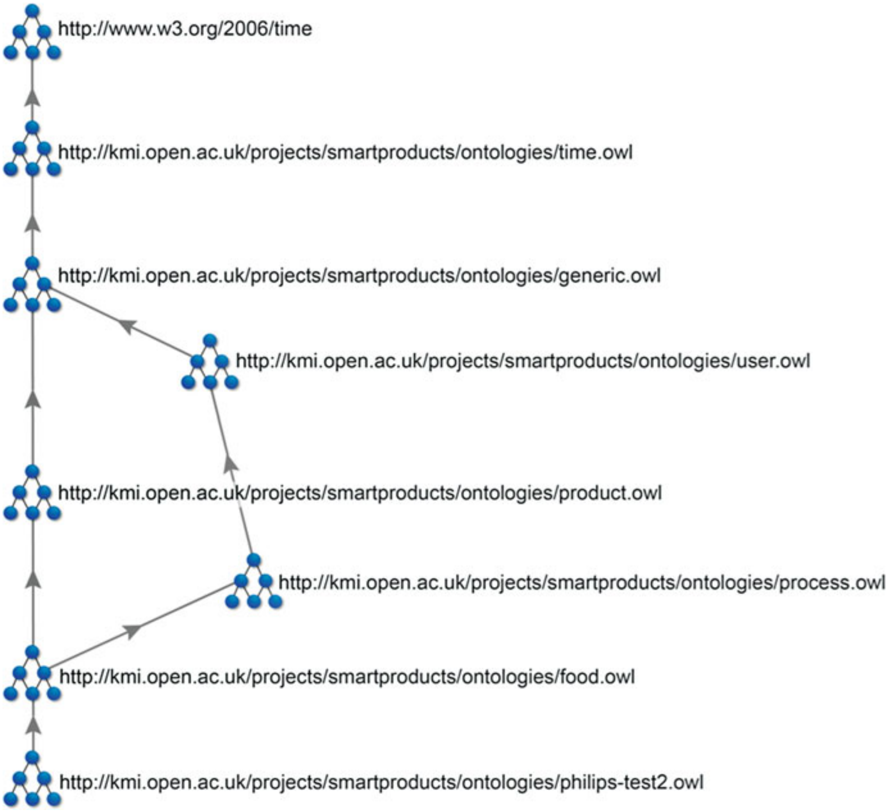

http://www.w3.org/2006/time
http://kmi.open.ac.uk/projects/smartproducts/ontologies/time.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/generic.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/user.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/product.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/process.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/food.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/philips-test2.owl

Fig. 16.1 Import relations in the SmartProducts network of ontologies

specification of smart devices, able to engage *proactively* in cooperative problem solving with other devices.

As shown in Fig. 16.1, the SmartProducts ontology is not a monolithic one but comprises a number of sub-ontologies, which define different notions, such as time, users, processes, products, etc. The project addresses three different test cases in the aerospace, car, and consumer appliances industries, and in particular, Fig. 16.1 shows the network of ontologies used to characterize the latter scenario, which we refer to as "Smart Kitchen." The structure of the network is highly reusable, with the top six nodes (i.e., ontologies) being shared across the three test cases, while the bottom two ontologies are specific to the Smart Kitchen application.

Like most other ontology engineering environments available today, the NeOn Toolkit provides an "*Ontology Navigator*" window, which supports ontology navigation using the classic top-down file system model, where clicking on a folder reveals its contents. In the case of an ontology engineering tool, the folder metaphor is used "to open up" a class, to reveal its sub-classes.

As discussed in (Katifori et al. 2007), this style of interface has several advantages, including its familiarity to users and the ability to support a systematic exploration of an ontology. For these reasons, it is more or less ubiquitous in ontology engineering toolkits and also tends to perform well in evaluations (Katifori et al. 2007). Nevertheless, it also exhibits some important limitations, including its inability to show role relations and “to support tasks related to the general ontology structure” (Katifori et al. 2007).

In this chapter, we will indeed focus on this category of tasks, which we will refer to informally as *ontology sensemaking* tasks. More specifically, we will use the term “sensemaking” to refer to the construction of a mental model of an ontology, which encompasses the ontology as a whole and is sufficient for a user to make a decision (for example) on whether an ontology is suitable for a particular application or whether it covers certain areas of interest to the required extent, with respect to user-specific criteria. In sum, the emphasis here will be less on supporting tasks which require understanding a particular detail of the ontology than on supporting tasks which require developing a “global” model of an ontology, at a certain level of abstraction. In addition, although KC-Viz also support the visualization of non-taxonomic (i.e., domain) relations, here we will focus the discussion almost exclusively on the navigation and visualization of taxonomies, on the basis that developing an understanding of the overall taxonomic structure of an ontology is an essential part of the sensemaking process.

16.2.2 Example: Using the Ontology Navigator for Sensemaking

Figure 16.2 shows a snapshot of the Ontology Navigator in the NeOn Toolkit, after we have clicked on the most specific ontology shown in Fig. 16.1 (see Sect. 16.2.1), which is called Philips-Test2. As shown in the figure, clicking on the folder Classes reveals the four topmost classes in the SmartProducts network of ontologies[3], making explicit the top-level structure of the ontology. However, while this initial visualization is useful to allow the user to understand the organization of the ontology at the highest level of abstraction, it is not yet comprehensive enough to allow the user to develop an overall model of the ontology in sufficient detail. In particular, without further exploration, it is not yet possible to achieve the following objectives, which are essential to the sensemaking process:

[3] This is because the relevant preference in the NeOn Toolkit is set to display all inherited classes, thus allowing us to browse the complete structure of the SmartProducts network of ontologies. Alternatively, we can choose to see only definitions local to the Philips-Test2 ontology, by deselecting the option “Show Imported Axioms.”

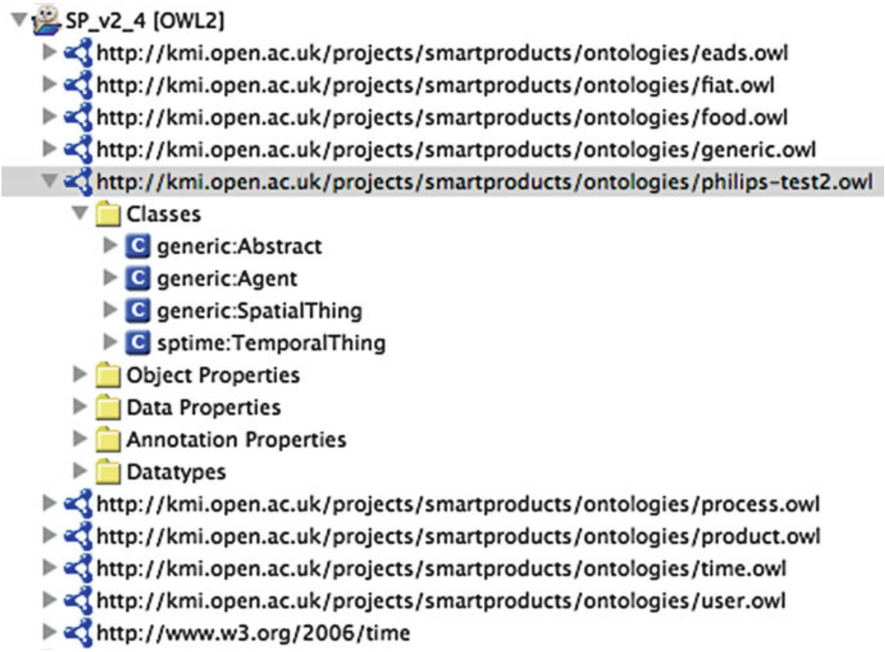

SP_v2_4 [OWL2]
http://kmi.open.ac.uk/projects/smartproducts/ontologies/eads.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/fiat.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/food.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/generic.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/philips-test2.owl
Classes
C generic:Abstract
C generic:Agent
C generic:SpatialThing
C sptime:TemporalThing
Object Properties
Data Properties
Annotation Properties
Datatypes
http://kmi.open.ac.uk/projects/smartproducts/ontologies/process.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/product.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/time.owl
http://kmi.open.ac.uk/projects/smartproducts/ontologies/user.owl
http://www.w3.org/2006/time

Fig. 16.2 Navigation through a file system metaphor

- Understanding the overall *size* and *shape* of the ontology. By “size” here we mean, given a node in the ontology[4], the total number of its direct and indirect sub-classes, while by “shape” we refer to an indication of the organization of the sub-classes. For instance, an ontology (or part of it) can have a *horizontal* (i.e., many sub-classes and few levels of depth), or a *vertical* (i.e., many inheritance levels and only a few sub-classes at each level) shape (Tartir et al. 2005). Understanding the shape of an ontology (or part of it) also means to understand whether it is *balanced*, indicating that all parts of the (sub-)ontology in question have been developed to a similar extent, or *unbalanced*, possibly indicating that some parts of the (sub-)ontology are less developed than others.
- Identifying the main components of the ontology and the typical *exemplars* of these components. For instance, from Fig. 16.2 (see Sect. 16.2.1), we understand that the SmartProducts ontology talks about spatial entities, a highly generic (and therefore not-so-informative) concept, but the display fails to tell us which kind of spatial entities the ontology primarily focuses on. Given that, for what we know, the sub-tree under class SpatialThing may contain dozens of sub-classes, it would be useful to have tools that could highlight to us the main spatial entities

[4] If the node is owl:Thing, then we are talking about the size of the whole ontology, otherwise the size of a particular subtree.

covered by the ontology (i.e., the *exemplars*), without the need for extensive exploration. In this case, this would require informing us that almost 50% of the sub-tree under SpatialThing concerns food-related notions. Informative exemplars can also help the user to predict the siblings of the class (i.e., the exemplar) in question, thus playing a summarization role not just with respect to its sub-tree, but also with respect to its siblings.

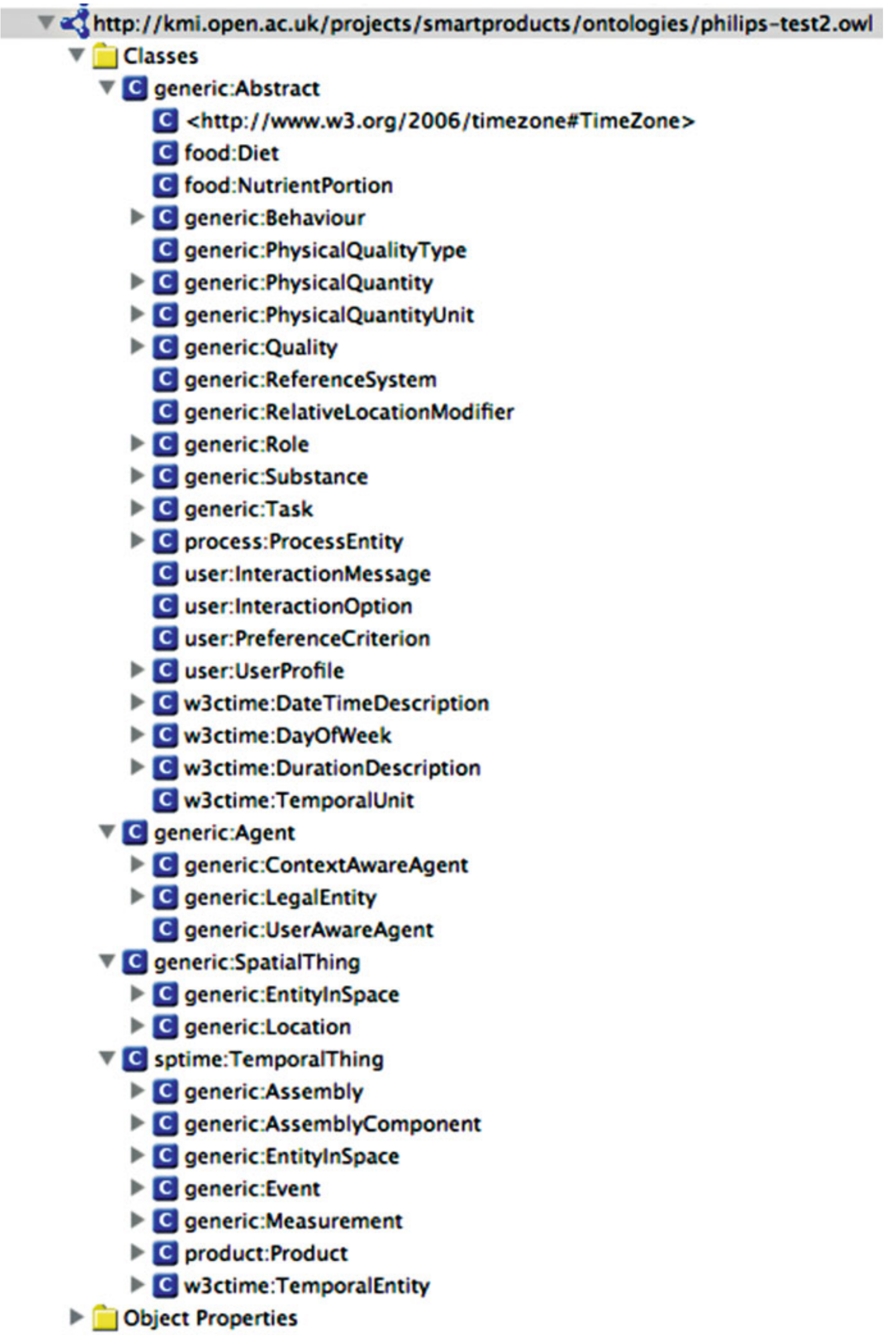
At this point, the reader may argue that what is needed is simply to explore the structure in more depth, by clicking on the top four classes, to open up the next level of detail. Figure 16.3 shows what happens when we do so and we click on all four level 1 classes. Thirty-eight classes are now displayed, making the picture rather complicated for the user. In addition, we are still none the wiser about which node we should further explore, which parts of the ontologies are developed more in detail, etc. And continuing to open up these nodes will simply bring more information on the screen, making it even more difficult for the user to develop a quick conceptual model of the ontology. Of course, the reader can also point out that part of the problem is the relative lack of structure underneath class Abstract, which contains 22 direct sub-classes. And indeed, a better organization of the sub-tree underneath class Abstract is obviously needed. However, it is also fair to say that the purpose of visualization and navigation tools is not simply to support navigation in relatively small, nicely organized ontologies. More importantly, they also need to help the user in making sense of and effectively explore large and possibly messy ontologies.

The brief and informal analysis shown here is consistent with the findings uncovered in more extensive empirical studies, such as (Dzbor et al. 2006), which highlight the problems users encounter when using rigid top-down navigation tools. These problems include:

- *Poor efficiency and effectiveness*. To open up the display shown in Fig. 16.3 has required six mouse clicks, and we still have a relatively poor understanding of the content of the ontology.
- *Lack of control when zooming on a particular node*. When clicking on a node, the user always opens up all the direct sub-classes. There is no way to control the number of sub-classes shown, or to open up more than one level with one mouse click.
- *No abstraction or saliency mechanisms*. The system has no way to automatically hide nodes which are deemed not important (i.e., *salient*) according to some criterion, and conversely, it is not able to bring to the attention of the user highly important nodes, again with respect to some user criterion.

It is also important to emphasize that such problems are less associated with the file system browsing metaphor than with the generic top-down navigation approach. For instance, ontology engineering toolkits such as TopBraid Composer[5]

[5] http://www.topquadrant.com/products/TB_Composer.html



```
▼ http://kmi.open.ac.uk/projects/smartproducts/ontologies/philips-test2.owl
  ▼ Classes
    ▼ generic:Abstract
        <http://www.w3.org/2006/timezone#TimeZone>
        food:Diet
        food:NutrientPortion
      ▶ generic:Behaviour
        generic:PhysicalQualityType
      ▶ generic:PhysicalQuantity
      ▶ generic:PhysicalQuantityUnit
      ▶ generic:Quality
        generic:ReferenceSystem
        generic:RelativeLocationModifier
      ▶ generic:Role
      ▶ generic:Substance
      ▶ generic:Task
      ▶ process:ProcessEntity
        user:InteractionMessage
        user:InteractionOption
        user:PreferenceCriterion
      ▶ user:UserProfile
      ▶ w3ctime:DateTimeDescription
      ▶ w3ctime:DayOfWeek
      ▶ w3ctime:DurationDescription
        w3ctime:TemporalUnit
    ▼ generic:Agent
      ▶ generic:ContextAwareAgent
      ▶ generic:LegalEntity
        generic:UserAwareAgent
    ▼ generic:SpatialThing
      ▶ generic:EntityInSpace
      ▶ generic:Location
    ▼ sptime:TemporalThing
      ▶ generic:Assembly
      ▶ generic:AssemblyComponent
      ▶ generic:EntityInSpace
      ▶ generic:Event
      ▶ generic:Measurement
      ▶ product:Product
      ▶ w3ctime:TemporalEntity
  ▶ Object Properties
```

Fig. 16.3 Exploring level 2 classes through the Ontology Navigator

provide graphic tools which also implement such top-down navigation and not surprisingly suffer from the same problems. Indeed, it can be argued that a graphical interface for top-down navigation typically performs worse than a file system model, primarily because the latter usually provides a much more compact representation – i.e., showing the 38 classes in a graphical tree representation will require a much larger display area, thus making it even more complex for a user to make sense of it (Plaisant et al. 2002).

KC-Viz is an ontology visualization and navigation system, which has been designed to address the issues highlighted here, by providing a rich set of navigation and visualization mechanisms, which include flexible *zooming* into and *hiding* of specific parts of an ontology, the ability to identify the most important concepts in an ontology, according to empirically validated criteria, as well as a plethora of other mechanisms to facilitate sensemaking and exploration of ontologies. As already mentioned, a key aspect of KC-Viz is its reliance on a key concepts extraction algorithm, which allows KC-Viz to produce the kind of ontology summaries that human experts are able to produce. Hence, in what follows we will first describe the key concept extraction algorithm used by KC-Viz, before providing an overview of its functionalities.

16.3 Key Concept Extraction

Informally, key concepts can be seen as the best descriptors of an ontology, i.e., information-rich concepts, which are most effective in summarizing what an ontology is about. In (Peroni et al. 2008), we considered a number of criteria to identify the key concepts in an ontology. In particular, we use the notion of natural category (Rosch 1978), to identify concepts that are information-rich in a psycholinguistic sense. This notion is approximated by means of two operational measures: name simplicity, which favors concepts that are labeled with simple names; and basic level, which measures how "central" a concept is in the taxonomy of an ontology. Two other criteria are drawn from the topology of an ontology: the notion of *density* highlights concepts which are information-rich in a formal knowledge representation sense, i.e., they have been richly characterized with properties and taxonomic relationships, while the notion of *coverage* is used to ensure that no important part of the ontology is neglected, by maximizing the coverage of the ontology with respect to its taxonomic relationships. Finally, the notion of *popularity*, drawn from lexical statistics, is introduced as a criterion to identify concepts that are likely to be most familiar to users. The *density* and *popularity* criteria are both decomposed in two sub-criteria: *global* and *local density*, and *global* and *local popularity*, respectively. While the global measures are normalized with respect to all the concepts in the ontology, the local ones consider the relative density or popularity of a concept with respect to its surrounding concepts. The aim here is to ensure that "locally significant" concepts get a high score, even though they may not rank too highly with respect to global measures. Each of these seven criteria produces a score for each concept in the ontology, and the final score assigned to a concept is a weighted sum of the scores resulting from individual criteria. As described in (Peroni et al. 2008), which provides a detailed account of our algorithm, KCE, and a formal definition of the criteria it employs (i.e., density, coverage, popularity, etc.), our approach has been shown to produce ontology summaries that correlate significantly with those produced by human experts.

16.4 Overview of KC-Viz

16.4.1 Initial Visualization of an Ontology with KC-Viz

Normally, a KC-Viz session[6] begins by generating an initial summary of an ontology, to get an initial "gestalt" impression of the ontology. This can be achieved in a number of different ways, most obviously by (1) selecting the ontology in question in the "Ontology Navigator" tab of the NeOn Toolkit, (2) opening up a menu of options by right clicking on the selected ontology, and then (3) choosing Visualize Ontology ➔ Visualize Key Concepts, through a sequence of menus. Figure 16.4 shows the result obtained after performing this operation on the ontology Philips-Test2[7], the most specific node in the SmartProducts network of ontologies[8]. As shown in the figure, we have now obtained an initial visualization of the network of ontologies, which includes concepts at different levels in the class hierarchy. This specific visualization includes 16 concepts because we have set the size of our ontology summary to 15, and the algorithm has automatically added the most generic concept, owl:Thing, to ensure that the visualization displays a connected graph. If we wish to display more or less succinct graphs, we can do so by changing the size of the ontology summary. The solid gray arrows in Fig. 16.4 indicate direct rdfs:subClassOf links, while the dotted green arrows indicate indirect rdfs:subClassOf links. As shown in the figure, by hovering the mouse over an indirect rdfs:subClassOf links, we can see the chain of rdfs:subClassOf relations, summarized by the indirect link.

Another important piece of information provided by KC-Viz is the size of the tree under a particular class, which is indicated by a pair of integers, indicating the

[6] All the examples in this paper have been generated using version 2.5 of the NeOn Toolkit and KC-Viz v1.3.0.

[7] It is important to point out that while Fig. 16.4 and later figures show exactly the concepts returned by KC-Viz, for the sake of readability we have, when appropriate, manually rearranged the layout, to try and minimize the compression caused by the physical size of this document. This is needed primarily because KC-Viz displays assume a landscape orientation, while this article is formatted according to a portrait orientation.

[8] Crucially, the option "Ontology summary considers also imported ontology" must be enabled in the KC-Viz preferences, otherwise only a summary of the concepts local to the Philips-Test2 ontology will be generated.

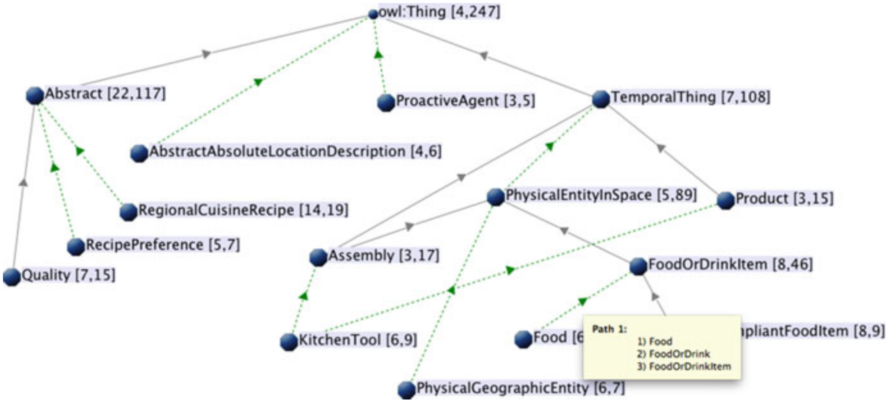


Fig. 16.4 Initial visualization of the SmartProducts ontologies

number of direct and indirect sub-classes. For instance, Fig. 16.4 tells us that class Abstract has 22 direct sub-classes and 117 indirect ones.

Although more exploration is obviously needed to get a thorough understanding of the contents of the SmartProducts ontology network, it can be argued that as a first step, the visualization shown in Fig. 16.4 already provides a rather effective starting point for the ontology sensemaking process. In particular, looking at the visualization, we can already reach a number of conclusions about the ontology, only one of which (the first one) could be concluded after opening up class owl: Thing in the ontology navigator – see Fig. 16.2, Sect. 16.2.1. For instance, we now understand that:

- The network of ontologies contains four top-level classes (i.e., classes directly linked to owl:Thing) – however, only two of them are displayed (Abstract and TemporalThing) in the initial summary.
- The ontology contains a lot of information about food – e.g., the tree under class FoodOrDrinkItem contains 46 classes.
- Key distinctions include time (TemporalThing) and space (PhysicalEntityInSpace). However, there are also temporal things, which are not physical entities in space. We can deduce this because the visualization tells us that TemporalThing has 108 sub-classes, while PhysicalEntityInSpace has 89.
- Class Abstract has a lot of sub-classes (117), but it may be relatively poorly structured, having 22 direct sub-classes.

More importantly, this initial visualization provides a much better structure for further exploration, than the rigid top-down navigation, which we illustrated in Sect. 16.2. In particular, on the basis of this initial snapshot, we can identify key “gaps” that we need to fill, in order to get a complete picture of the ontology. For instance, we may want to explore:

- The sub-tree under class Abstract to get a better understanding of this part of the ontology. As discussed in Sect. 16.2, because of the relatively poor structure of this part of the ontology, better control of the navigation process than that provided by the Ontology Navigator will be needed, in order to be able to explore this part of the ontology effectively.
- The sub-tree under FoodOrDrinkItem, as this is clearly a rich part of the ontology.
- The sub-tree under PhysicalEntityInSpace, which appears to encompass both location-related notions and device-related ones (Assembly).
- Which sub-classes of TemporalThing are not also sub-classes of PhysicalEntityInSpace.
- What kinds of agents are modeled by this ontology.
- Why products are not physical entities in space.
- Others.

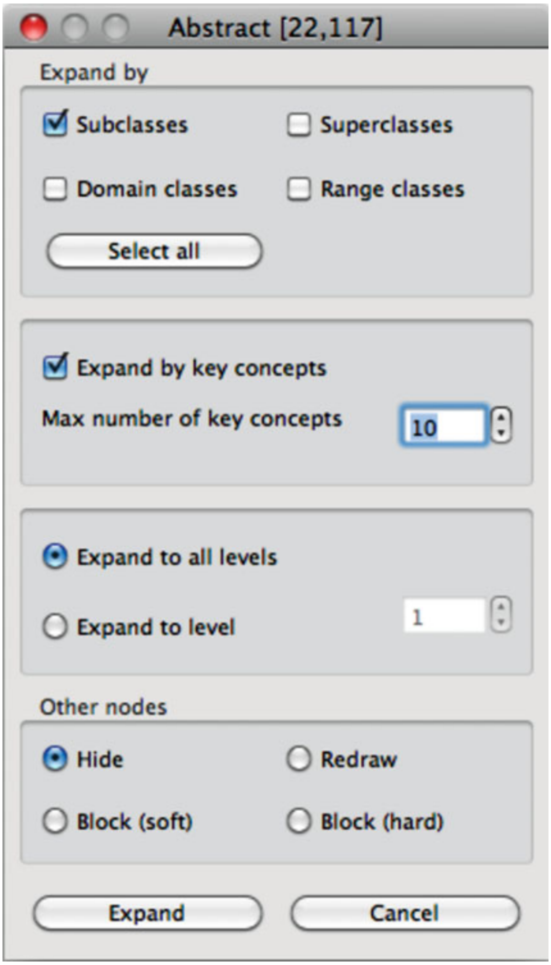
In sum, the claim here is that the key concept extraction algorithm used by KC-Viz, together with the degree of control that we get over it (size of summaries and whether or not to consider imported axioms), allows the effective generation of initial ontology snapshots, which helps the user in forming an initial idea of what an ontology is about. In what follows, we show how the flexible support for exploration provided by KC-Viz capitalizes on this initial summary to facilitate effective ontology navigation and sensemaking.

16.4.2 Exploring Ontologies with KC-Viz

Let us consider our first task: to get a better understanding of the sub-tree under class Abstract. We have already seen that a rigid top-down approach does not work very well here, in particular because class Abstract contains many direct sub-classes. So, let us try exploring with KC-Viz.

If we click right on a class displayed in KC-Viz, in this case, Abstract, we obtain a menu which includes options for inspecting, expanding, and hiding a class. If we select “Expand,” the menu shown in Fig. 16.5 pops up, which provides a rich set of options for exploring the sub-tree under class Abstract. In particular, the following four options for customizing the expansion algorithm are presented to the user:

- Whether to explore following taxonomic relations, other relations (through domain and range), or any combination of these.
- Whether or not to make use of the ontology summarization algorithm, which in this case will be applied only to the sub-tree of class Abstract.
- Whether or not to limit the range of the expansion – e.g., by expanding only to 1 or 2 levels.
- Whether to display the resulting visualization in a new window (“Hide”), or whether to add the resulting nodes to the current windows. In the latter case, some degree of control is given to the user with respect to the redrawing algorithm, by allowing her to decide whether or not to limit the freedom of the



Abstract [22,117]

Expand by

☑ Subclasses □ Superclasses

□ Domain classes □ Range classes

☑ Expand by key concepts

Max number of key concepts 10

● Expand to all levels

○ Expand to level 1

Other nodes

● Hide ○ Redraw

○ Block (soft) ○ Block (hard)

Fig. 16.5 Expanding sub-trees in KC-Viz

graph layout algorithm to rearrange existing nodes. This is particularly useful in those situations where expansion is meant to add only a few nodes, and the user does not want the layout to be unnecessarily modified – e.g., because she has already manually rearranged the nodes according to her own preferences.

As shown in Fig. 16.5, we have chosen to expand by key concepts, we have kept the limit of the expansion to 10 concepts, and we have also chosen to hide the other concepts, to be able to explore the sub-tree of class Abstract in a new window, without the “noise” from unrelated concepts.

Figure 16.6 shows the result of the expansion, which confirms the relatively poor degree of structure of the sub-tree under class Abstract, where only classes UserProfile and PhysicalQuantity appear to have been further characterized in

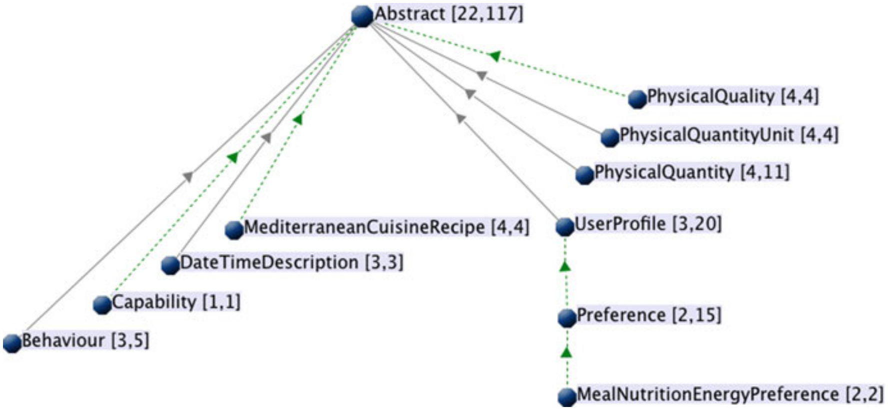


Fig. 16.6 Key concepts under class Abstract

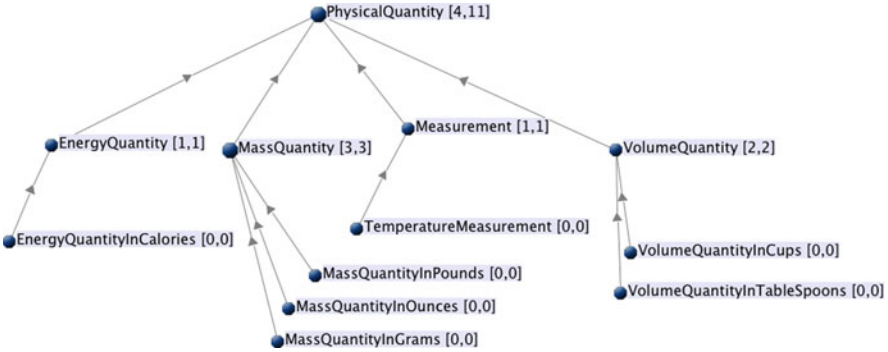

PhysicalQuantity [4,11]
EnergyQuantity [1,1]
MassQuantity [3,3]
Measurement [1,1]
VolumeQuantity [2,2]
EnergyQuantityInCalories [0,0]
TemperatureMeasurement [0,0]
VolumeQuantityInCups [0,0]
MassQuantityInPounds [0,0]
VolumeQuantityInTableSpoons [0,0]
MassQuantityInOunces [0,0]
MassQuantityInGrams [0,0]

Fig. 16.7 Expanding class PhysicalQuantity

some detail and may warrant further exploration. In particular, we can look in more detail at the latter, by expanding its sub-tree at all levels, without restricting it to key concepts, as shown in Fig. 16.7.

Analogously, we can also explore the other key constituents of the SmartProducts network of ontologies, by careful expansion of the sub-trees we wish to explore. For instance, again by choosing expansion by key concepts, we can find out more about the structure of the sub-tree under FoodOrDrinkItem, as shown in Fig. 16.8.

16.4.3 Other Functionalities Provided by KC-Viz

While the flexible expansion mechanism is the key facility provided by KC-Viz to support flexible exploration of ontology trees, a number of other functionalities are

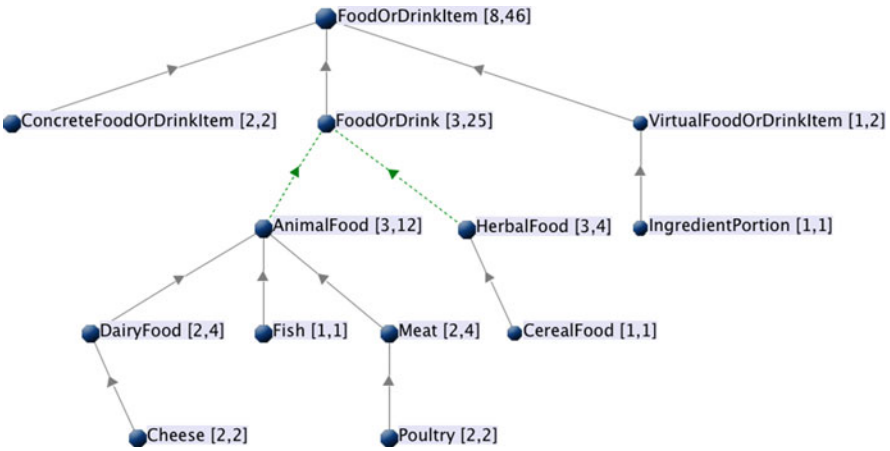
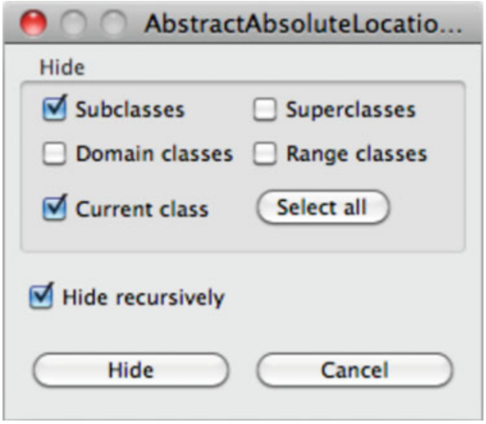


Fig. 16.8 Expanding class FoodOrDrinkItem



Hide

☑ Subclasses □ Superclasses

□ Domain classes □ Range classes

☑ Current class

☑ Hide recursively

Fig. 16.9 Options for removing classes from a display

also provided, to ensure a comprehensive visualization and navigation support. These include:

- A flexible mechanism for hiding nodes, as shown in Fig. 16.9.
- Integration with the Entity Properties and Ontology Navigator tabs in the NeOn Toolkit, to support detailed inspection of classes.
- A dashboard, shown in Fig. 16.10, which allows the user to move back and forth through the history of KC-Viz operations, to modify the formatting of the layout, and to save the current display to a file, among other things.
- A preferences menu, shown in Fig. 16.11, which allows the user to set defaults for the most common operations and also enables her to switch to a more efficient (but sub-optimal) algorithm when dealing with very large ontologies.

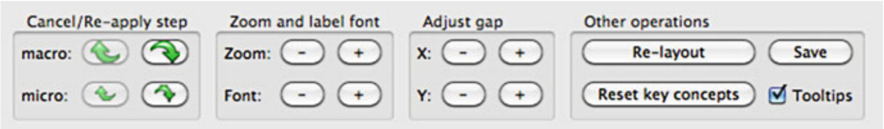
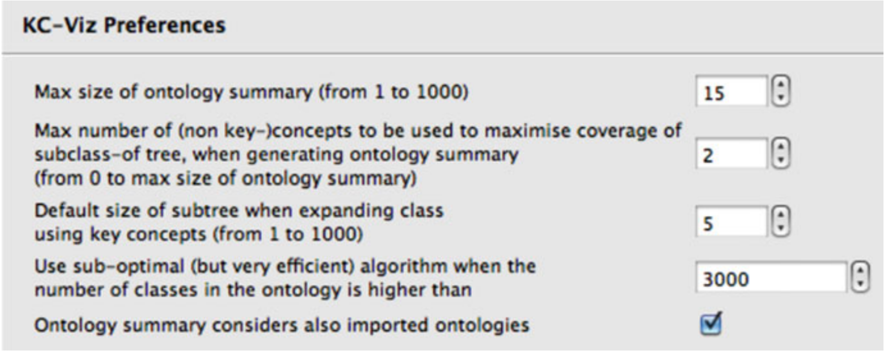


Fig. 16.10 The KC-Viz dashboard



KC-Viz Preferences

Max size of ontology summary (from 1 to 1000): 15

Max number of (non key-)concepts to be used to maximise coverage of subclass-of tree, when generating ontology summary (from 0 to max size of ontology summary): 2

Default size of subtree when expanding class using key concepts (from 1 to 1000): 5

Use sub-optimal (but very efficient) algorithm when the number of classes in the ontology is higher than: 3000

Ontology summary considers also imported ontologies ☑

Fig. 16.11 KC-Viz preferences

16.4.4 *Summing Up: How KC-Viz Addresses Key Challenges for Ontology Editors*

Echoing the findings reported in the paper by (Dzbor et al. 2006), in Sect. 16.2 we highlighted a number of issues which hamper the effectiveness of current tools for visualizing and navigating ontologies. Here we revisit these issues, discussing how KC-Viz attempts to address them:

- *Poor efficiency and effectiveness*. The exploration sequence shown in Figs. 16.4, 16.6, and 16.7 (see Sect. 16.4) only required three operations and arguably provided us with a rather good understanding (at a given level of abstraction) of a significant part of the ontology. In our view, this compares favorably with the sequence described in Sect. 16.2, where several expansion operations did not dramatically improve our understanding of the ontology. It is also relatively straightforward to see that by repeating the exploration process we applied to class Abstract to other three or four key classes shown in Fig. 16.4 (see Sect. 16.4.1), we should be able to converge quickly to a rather comprehensive overview of the SmartProducts network of ontologies.
- *Lack of control when zooming on a particular node*. KC-Viz addresses this limitation by providing a very flexible set of options for node expansion, as shown in Fig. 16.5 (see Sect. 16.4.2).

- *No abstraction or saliency mechanisms*. The main abstraction mechanism provided by KC-Viz is the key concept extraction algorithm, KCE, which automatically identifies the “most important” concepts in the ontology, thus making it possible to present snapshots of the ontology to the user, while hiding away the “less important” concepts. Crucially, KCE has been empirically validated, thus providing a sound basis to the approach used in KC-Viz. In addition, by displaying information about the size of the sub-graphs under each node and by also varying the size of the graphical node representing a class in KC-Viz, the tool also provides a simple but effective mechanism to highlight the most “salient” classes in an ontology.

It is also interesting to assess the functionalities provided by KC-Viz with respect to the seven visualization task types proposed in (Shneiderman 1996). As discussed below, KC-Viz supports all of them:

- *Overview*. This is one of the key functionalities provided by KC-Viz. In contrast with other approaches – e.g., CropCircles (Wang and Parsia 2006), which sacrifice the display of explicit labels for the sake of maximizing the number of nodes on display, KC-Viz follows an alternative (and to our knowledge, unique) approach: it exploits the ontology summarization algorithm to provide initial overviews of an ontology and then allows the user to explore any part of the model in details. In our view, the advantage here is that, at any given stage of the process, only relatively few nodes are displayed and all of them are readable, thus making it easy for the user to make sense of the model on display. In addition, the simple display of information about the size of a class’ sub-graph (shown as two integers describing the number of direct and indirect sub-classes) also provides summary information for the parts of the ontology which are not displayed, thus avoiding the need for abstract visualizations of clusters of nodes.
- *Zoom*. This functionality is supported by the Expand menu item, which provides a flexible set of options for exploring a sub-graph in detail. As a result, the user remains in control of both the size of the exploration space and the criteria used to generate it.
- *Filter*. This functionality is provided as a side effect of the ability of KC-Viz to focus on a particular part of the ontology and can also be invoked explicitly by the user by means of the Hide menu option.
- *Details-on-demand*. A tight integration with the Entity Properties view of the NeOn Toolkit makes it possible to click on any node displayed in KC-Viz and inspect it.
- *Relate*. KC-Viz supports the visualization of both taxonomic and domain/range relationships between classes. An example is given in Fig. 16.12, where all the “level 1” relationships between FoodOrDrinkItem and other classes in the ontologies are displayed. In particular, the dashed red arrows are used to indicate domain/range relations, with the labels being displayed when the mouse hovers over the arrow in question. In this case, we are showing that the ontology contains a relation hasNutrient, whose domain is FoodOrDrinkItem and whose range is NutrientPortion.

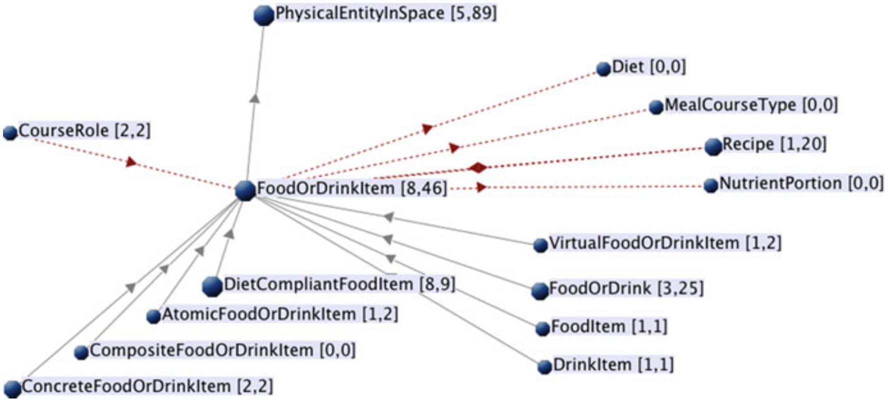

PhysicalEntityInSpace [5,89]
Diet [0,0]
MealCourseType [0,0]
CourseRole [2,2]
Recipe [1,20]
FoodOrDrinkItem [8,46]
NutrientPortion [0,0]
VirtualFoodOrDrinkItem [1,2]
DietCompliantFoodItem [8,9]
FoodOrDrink [3,25]
AtomicFoodOrDrinkItem [1,2]
FoodItem [1,1]
CompositeFoodOrDrinkItem [0,0]
DrinkItem [1,1]
ConcreteFoodOrDrinkItem [2,2]

Fig. 16.12 Displaying both taxonomic and domain/range relationships

- *History*. KC-Viz supports undo/redo actions at both macro and micro level, to allow users to go back to, replay, or undo earlier operations.
- *Extract*. The key mechanism for extracting parts of an ontology is through the Expand menu, allowing flexible extraction of nodes at various levels in the hierarchy, in accordance with the key concept extraction algorithm, and following taxonomic and/or domain or range relationships.

16.5 Related Work

Surveys on ontology visualization methods, like (Katifori et al. 2007), categorize the methods for visualizing ontologies in six non-exclusive main types, called *indented list, node-link and tree*, *space-filling*, *zoomable*, *context* + *focus* and *distortion*, and *3D information landscapes*.

The *indented list* category covers tree-centric views of the ontology, similar to the one provided by the Ontology Navigator in the NeOn Toolkit, which was shown in Figs. 16.2 and 16.3 (see Sect. 16.2). As already pointed out, because of its familiarity to users, this style of interface is pretty much ubiquitous in ontology engineering toolkits; however, it does not support sensemaking tasks very well, especially in the case of large or unstructured ontologies.

Methods like *IsaViz*[9], *OntoViz*[10], and *SpaceTree* (Plaisant et al. 2002) are typical members of the second category *(node-link and tree)*, as they represent an ontology through a graphical display of interconnected nodes. Hence, these systems are similar to KC-Viz, with the crucial difference that, while KC-Viz uses ontology

[9] http://www.w3.org/2001/11/IsaViz

[10] http://protegewiki.stanford.edu/index.php/OntoViz

summarization to abstract out large trees, these methods tend to use preview icons – e.g., a triangle in the case of SpaceTree, to abstract out large sub-graphs.

TreeMap (Shneiderman 1992) is representative of *space-filling* approaches, which focus on optimizing the use of screen space to maximize the amount of information displayed to the user. However, in order to achieve this goal, these approaches tend to move away from the visualizations familiar to users, such as indented lists and graphs, and therefore they tend to require much more effort from users. In addition, it can be argued that no matter how much optimization a system tries to achieve, eventually it will fill all the available screen space, once a large enough set of data is given as input. Hence, while space filling is a useful secondary goal, in our view, the key goal for a visualization system remains the ability to provide views at different levels of abstraction, and in this respect, it can be argued that KC-Viz is unique in its reliance on an empirically validated ontology summarization algorithm, as opposed to general-purpose data abstraction techniques.

CropCircles (Wang and Parsia 2006) is an example of a "*zoomable visualization*," i.e., an approach which presents "the nodes in the lower levels of the hierarchy nested inside their parents and with smaller size than that of their parents" (Katifori et al. 2007). Much like KC-Viz, these approaches can be effective in providing good overviews, abstracting from large numbers of nodes. However, as already mentioned, in contrast with KC-Viz, which uses ontology summarization to provide abstraction, they sacrifice the display of explicit labels for the sake of maximizing the number of nodes on display.

The group of techniques categorized as "*context + focus and distortion*" is based on "the notion of distorting the view of the presented graph in order to combine context and focus. The node on focus is usually the central one and the rest of the nodes are presented around it, reduced in size until they reach a point that they are no longer visible" (Katifori et al. 2007). These techniques offer a good trade-off – a part of the ontology is shown in detailed (often tree-like) view, while the rest is depicted around. A typical approach here is *HyperTree* (Souza et al. 2003), which skews the visualized model to emphasize the node currently explored by the user. However, the problem with these techniques is that they essentially attempt to show everything in the model, which often makes the "context" part of little consequence and illegible. In contrast with these approaches, KC-Viz leverages the advantages derived from using ontology summaries, allowing the user to focus on ontology entities bearing the highest information value and "contextualizing" them against the entities with systematically lower information values. Crucially, it also provides flexible and effective mechanisms to change the focus to other entities, as and when required.

Finally, Katifori et al. also discuss a class of systems called "*Information Landscapes*," which provide a 3D, landscape-oriented alternative to zoomable visualizations. Hence, the remarks we made above about the latter category of systems apply to information landscapes as well.

16.6 Conclusions and Future Work

In this chapter, we have presented KC-Viz, an innovative approach to visualizing and navigating ontologies, which exploits a powerful ontology summarization algorithm, KCE, to introduce effective abstraction mechanisms in the ontology exploration and sensemaking processes. Crucially, KC-Viz maximizes the value of the foundational functionality afforded by KCE, by providing a flexible set of options to zoom in or hide specific parts of an ontology, history browsing mechanisms, flexible graphical layout formatting, and integration with other components of the NeOn Toolkit.

Our next task will be to evaluate KC-Viz formally, by comparing the performance in sensemaking tasks of users equipped with KC-Viz versus other users, to try and determine whether there is objective evidence that KC-Viz improves both the efficiency and the effectiveness of a sensemaking task.

We also plan to improve the range of functionalities provided by KC-Viz, in particular by opening up the key concept extraction algorithm to the users, to allow them to decide which criteria to prioritize in the generation of ontology summaries. Also, better explanation facilities are needed, as in some cases it is not easy to understand why a particular concept is deemed “important” by KC-Viz, while another one is not.

In conclusion, it can be argued that, with a few exceptions, the ontology engineering community has historically overlooked the importance of HCI issues and has failed to provide user interfaces that can truly support users effectively, as highlighted by Dzbor et al. (2006). With KC-Viz, we are trying to make an important step in the direction of providing better user support for ontology exploration and sensemaking, and we hope that our forthcoming empirical evaluation studies will confirm our intuition that the approach implemented in KC-Viz does indeed provide better sensemaking support for users of ontology engineering environments.

Acknowledgments This work was partially supported by funding from the European Commission, in the context of the NeOn and SmartProducts projects. The paper has benefited greatly from many insightful comments from Pierluigi Miraglia, who kindly suggested both ways to improve the presentation of this work as well as interesting new directions for future research.

References

d’Aquin M, Sabou M, Motta E (2008) Reusing knowledge from the semantic web with the Watson Plugin. Demo at the 2008 international semantic web conference, Karlsruhe, Germany

Dzbor M, Motta E, Buil Aranda C, Gómez-Pérez JM, Goerlitz O, Lewen H (2006) Developing ontologies in OWL: an observational study. Workshop on OWL: experiences and directions, Athens, GA, USA, Nov 2006

Katifori A, Halatsis C, Lepouras G, Vassilakis C, Giannopoulou E (2007) Ontology visualization methods—a survey. ACM Comput Surv 39(4):Art.10

Peroni S, Motta E, d'Aquin M (2008) Identifying key concepts in an ontology through the integration of cognitive principles with statistical and topological measures. In: Third Asian Semantic Web Conference, Bangkok, Thailand

Plaisant C, Grosjean J, Bederson BB (2002) Spacetree: supporting exploration in large node link tree, design evolution and empirical evaluation. In: Proceedings of the international symposium on information visualization, 2002

Rosch E (1978) Principles of categorization. In: Cognition and categorization. Lawrence Erlbaum, Hillsdale

Shneiderman B (1992) Tree visualization with tree-maps: a 2d space-filling approach. ACM Trans Graph 11(1):92–99, 15

Shneiderman B (1996) The eyes have it: a task by data type taxonomy for information visualizations. In: Proceedings of the 1996 IEEE symposium on Visual Languages (VL 1996), Boulder, CO, USA. IEEE Computer Society, Washington, DC, USA

Souza K, Dos Santos A, Evangelista SRM (2003) Visualization of ontologies through hypertrees. In: Proceedings of the Latin American conference on Human-computer interaction, 2003, p 251–255

Tartir S, Arpinar IB, Moore M, Sheth AP, Aleman-Meza B (2005) OntoQA: Metric-based ontology quality analysis. In Proceedings of the IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources, co-located with the 5th IEEE International Conference on Data Mining (ICDM 2005). November 27, 2005, Huston, Texas, USA

Wang TD, Parsia B (2006) Cropcircles: topology sensitive visualization of owl class hierarchies. In: Proceedings of the 5th International Semantic Web Conference 2006, Athens, GA, USA

Chapter 17
Reasoning with Networked Ontologies

Guilin Qi and Andreas Harth

Abstract The chapter covers basic functionality pertaining to reasoning with ontologies. We first introduce general methods for detecting and resolving inconsistencies, and then present three plugins that provide reasoning and query functionality. The three plugins are: the reasoning plugin, which allows for standard reasoning tasks, such as materialising inferences and checking consistency in ontologies; the RaDON plugin, which provides functionality for diagnosing and resolving inconsistencies in networked ontologies; and the query plugin, which allows for users querying ontologies in the NeOn Toolkit via the RDF query language SPARQL.

17.1 Introduction

Reasoning plays an important role in knowledge engineering. With reasoning, users can check whether an ontology (or a network of ontologies) contains logical inconsistencies, which may indicate an error in modelling or an incompatibility between combined ontologies. Further, reasoning can provide means to deduce new facts from existing facts and axioms. Finally, users need means to pose questions to knowledge bases and retrieve answers to those queries. Ideally, query processing over ontologies takes the meaning – as formally specified via logical axioms – into account when deriving answers to a query. Ontology languages provide the mechanisms for encoding such meaning, and reasoners provide the mechanisms for processing such meaning.

G. Qi (✉)
School of Computer Science and Engineering, Southeast University, Nanjing, China
e-mail: gqi@seu.edu.cn

A. Harth
Institute AIFB, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
e-mail: harth@kit.edu

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_17, © Springer-Verlag Berlin Heidelberg 2012

Many ontology languages have been developed, ranging from simple languages such as the Resource Description Framework (RDF) and the RDF Vocabulary Description Language, that is, RDF Schema (RDFS), to expressive languages such as the Web Ontology Language OWL. Ontologies specified in these languages allow for deductive reasoning: drawing conclusions based on facts and axioms in a knowledge base. Often used reasoning tasks are checking the consistency of a knowledge base, materialising inferences that can be drawn from the codified knowledge in an ontology and answering queries over the knowledge base.

To support the life cycle of networked ontologies, we have developed three plugins which provide reasoning services in the NeOn Toolkit:

- The reasoning plugin, which provides standard reasoning tasks on OWL 2 ontologies. Using the reasoner plugin, a user can get all the facts that can be inferred from specified facts and axioms and check if an ontology is consistent.
- The RaDON plugin, which provides functionalities for diagnosing and resolving inconsistencies in networked ontologies. That is, in case the reasoner found an ontology to be inconsistent, it is difficult for a user to figure out the cause of the inconsistency, as the reasoner only provided a yes/no result. Within the NeOn Toolkit, the RaDON plugins enable a user to investigate the cause for an inconsistency. In addition, RaDON features algorithms which automatically resolve inconsistencies by removing those axioms which cause the inconsistency.
- The query plugin, which allows users to pose queries in the SPARQL query language (Prud'hommeaux and Seaborne 2008; Clark et al. 2008).

We cover preliminaries in Sect. 17.2, describe the approach for investigating and resolving inconsistencies in Sect. 17.3, explain the main functionalities the plugins provide in Sect. 17.4, summarise the usage of the plugins in Sect. 17.5 and finally conclude with Sect. 17.6.

17.2 Preliminaries

In the following, we introduce basic notions used throughout the rest of the chapter.

17.2.1 RDF and RDFS

RDF serves as foundational data model for the Semantic Web. RDF is a graph-structured data model for encoding semi-structured data. RDF data consists of RDF triples: statements of subject-predicate-object. The formal semantics of RDF is specified in Hayes (2004). We assume the reader is familiar with basic notions of web architecture, such as Internationalised Resource Identifiers[1] (IRIs), which are a generalisation of URIs[2].

Definition 17.1 (RDF Triple, RDF Graph, RDF Term). Let $\mathcal{I}$ be the set of IRIs, $\mathcal{B}$ the set of blank nodes and $\mathcal{L}$ the set of RDF literals. An element of the set $\mathcal{TR} = (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ is called RDF triple. A set of triples is called RDF graph. An element of $\mathcal{C} = \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ is called RDF term.

Ontology languages, such as RDFS and OWL, provide vocabulary for describing classes and properties. Constructs of RDFS, such as subclass and sub-property relations, as well as constructs of the more expressive ontology language OWL are covered in the next section.

17.2.2 Description Logics

Description logics (DL) (Baader et al. 2003) are a well-known family of knowledge representation formalisms for ontologies. They are fragments of first-order predicate logic. That is, they can be translated into first-order predicate logic according to Borgida (1994). They differ from their predecessors such as frames (Minsky 1974) in that they are equipped with a formal, logic-based semantics. In DL, elementary descriptions are *concept names* (unary predicates) and *role names* (binary predicates). Let N_C and N_R be pairwise disjoint and countably infinite sets of *concept names* and *role names,* respectively. We use the letters A and B for concept names, the letter R for role names and the letters C and D for concepts. $\top$ and $\bot$ denote the universal concept and the bottom concept, respectively. Complex descriptions are built from them recursively using concept and role constructors provided by the particular DL under consideration. For example, in DL $\mathcal{ALC}$ (Schmidt-Schau and Smolka 1991), the set of $\mathcal{ALC}$ concepts is the smallest set such that (1) every concept name is a concept and (2) if C and D are concepts, R is a role name, then the following expressions are also concepts: $\neg C$ (full negation), $C \sqcap D$ (concept conjunction), $C \sqcup D$ (concept disjunction), $\forall R{\cdot}C$ (value restriction on role names) and $\exists R.C$ (existential restriction on role names).

A *terminology axiom* is an expression of the form $C \sqsubseteq D$ where C and D are concept expressions, and a *role axiom* is an expression of the form $R \sqsubseteq S$, where R and S are role expressions. A TBox, denoted by $\mathcal{T}$, is a set of terminology axioms and role axioms which are viewed as intensional description of the domain of interest. An assertional axiom is an expression of the form $C(a)$ or $R(a, b)$, where C is a concept expression, R is a role expression and a and b are individual names. An ABox, denoted by $\mathcal{A}$, is a set of assertional axioms, which are viewed as extensional

[1] http://www.ietf.org/rfc/rfc3987.txt

[2] http://www.ietf.org/rfc/rfc3987.txt

information. Finally, A DL-based knowledge base (or ontology) is a pair $O = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ and $\mathcal{A}$ are a TBox and an ABox, respectively.

The semantics of a DL is defined by an interpretation $\mathcal{I} = \left(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\right)$ which consists of a non-empty domain set $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$, which maps from individuals, concepts and roles to elements of the domain, subsets of the domain and binary relations on the domain, respectively. Given an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ satisfies a terminology axiom $C \sqsubseteq D$ (respectively, a role axiom $R \sqsubseteq S$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (respectively,$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$). Furthermore, $\mathcal{I}$ satisfies a concept assertion $C(a)$ (respectively, a role assertion $R(a, b)$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (respectively, $\left(a^{\mathcal{I}}, b^{\mathcal{I}}\right) \in R^{\mathcal{I}}$). An interpretation $\mathcal{I}$ is called a *model* of an ontology O, iff it satisfies each axiom in O. The set of all models of an ontology O is denoted as $M(O)$. We say that two ontologies O_1 and O_2 are logically equivalent, denoted as $O_1 \equiv O_2$, if $M(O_1) = M(O_2)$. A concept name C in an ontology O is unsatisfiable if for each model $\mathcal{I}$ of O, $C^{\mathcal{I}} = \emptyset$. An ontology O is incoherent if there exists an unsatisfiable concept name in O. An ontology O is inconsistent iff it has no model.

17.3 Diagnosing and Repairing Inconsistent Networked Ontologies

17.3.1 Relationship Between Inconsistency and Incoherence

The relationship between incoherence and inconsistency is not simple. First, the fact that an ontology is inconsistent does not necessarily imply that it is incoherent, and vice versa. There exist different combinations of inconsistency and incoherence, as illustrated in Fig. 17.1 (most of the examples in this figure have been proposed in Flouris et al. (2006)) and discussed in the following.

Figure 17.1(1) is an example of a consistent but incoherent ontology. In this ontology, two concepts $C1$ and $C2$ are claimed to be disjoint but share a subconcept $C3$. Figure 17.1(2)–(4) show examples of inconsistent ontologies. Figure 17.1(2) is an example of an inconsistent but coherent ontology. In this ontology, two concepts $C1$ and $C2$ are claimed to be disjoint but share a subconcept which is a nominal $\{a\}$. Figure 17.1(3) is an example of an inconsistent ontology in which an instance a instantiates a concept $C1$ and its complement $\neg C1$. Figure 17.1(4) is an example of an inconsistent but coherent ontology, in which the

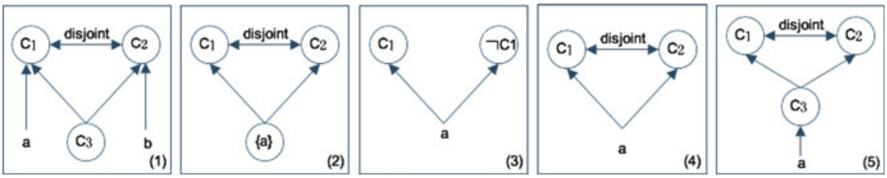


Fig. 17.1 Examples of variants of inconsistency. Pattern (1) to Pattern (5)

two disjoint concepts $C1$ and $C2$ share an instance a. Finally, Fig. 17.1(5) shows an example of an ontology that is both incoherent and inconsistent.

From the definitions of incoherence above, we know that incoherence can occur in the terminology level only. When dealing with inconsistency, we can differentiate terminology axioms and assertional axioms. We have the following categorisation of different kinds of reason for inconsistent ontologies.

- *Inconsistency due to terminology axioms*: In this case, we only consider inconsistency in TBoxes. Figure 17.1(2) is an example of such an inconsistency. Following our definitions, this kind of inconsistency will make the TBox incoherent:
- *Inconsistency due to assertional axioms:* This kind of inconsistency only occurs in ABoxes. It is not related to incoherence. A source of assertional inconsistency is that there are conflicting assertions about one individual, for example, an individual is asserted to belong to a class and its complement class, as in Fig. 17.1(2).
- *Inconsistency due to terminology and assertional axioms:* In this case, each conflicting set of axioms must contain both terminology axioms and assertional axioms. This kind of inconsistency is sometimes dependent on incoherence. Such an example is shown in Fig. 17.1(5). It is easy to see that C_3 is an unsatisfiable concept and that O is inconsistent. The reason for the inconsistency is that the individual a instantiates C_3 which is unsatisfiable. Therefore, if we repair the unsatisfiable concept C_3, then the inconsistency will disappear as well. On the other hand, the inconsistency in the example in Fig. 17.1(4) is not caused by an incoherence.

The first kind of inconsistency is only related to terminology axioms. In this case, the unit of change is a concept (either atomic or complex). Therefore, some revision approaches which take the individual names as the unit of change, such as the one proposed in Qi et al. (2006), cannot be applied to deal with this kind of inconsistency. In contrast, the other two kinds of inconsistency are related to assertional axioms. So the unit of change can be either a concept or an individual name.

From this discussion, we observe that the causes for incoherence and inconsistency are manifold, and their interdependencies are complex. Incoherence is always caused by conflicts in the terminology. It may or may not affect the consistency of the overall ontology. Inconsistency may arise due to conflicts in the ABox, in the TBox or a combination of both ABox and TBox.

17.3.2 Debugging Inconsistent/Incoherent Ontologies

Current DL reasoners, such as RACER, can detect logical incoherence and return unsatisfiable concepts in an OWL ontology. However, they do not support the diagnosis and incoherence resolution at all. To explain logical incoherence, it is important to debug *relevant* axioms which are responsible for the contradiction.

Definition 17.2. (Schlobach and Cornet 2003) Let A be a named concept which is unsatisfiable in a Tbox $\mathcal{T}$. A set $\mathcal{T}' \subseteq \mathcal{T}$ is a *minimal unsatisfiability-preserving sub-TBox (MUPS)* of $\mathcal{T}$ if A is unsatisfiable in $\mathcal{T}'$ and A is satisfiable in every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$. The set of all MUPS of $\mathcal{T}$ with respect to A is denoted as $MU_A(\mathcal{T})$.

A MUPS of $\mathcal{T}$ with respect to A is the minimal sub-TBox of $\mathcal{T}$ in which A is unsatisfiable. We will abbreviate the set of MUPS of $\mathcal{T}$ with respect to a concept name A by $mups(\mathcal{T}, A)$. Let us consider an example from Schlobach and Cornet (2003). Suppose $\mathcal{T}$ contains the following axioms:

$$\begin{array}{ll} ax_1 : A_1 \sqsubseteq \neg A \sqcap A_2 \sqcap A_3 & ax_2 : A_2 \sqsubseteq A \sqcap A_4 \\ ax_3 : A_3 \sqsubseteq A_4 \sqcap A_5 & ax_4 : A_4 \sqsubseteq \forall s.B \sqcap C \\ ax_5 : A_5 \sqsubseteq \exists s.\neg B & ax_6 : A_6 \sqsubseteq A_1 \sqcup \exists r.(A_3 \sqcap \neg C \sqcap A_4) \\ ax_7 : A_7 \sqsubseteq A_4 \sqcap \exists s.\neg B & \end{array}$$

where A, B and C are atomic concept names and A_i ($i = 1,\ldots,7$) are defined concept names, and r and s are atomic roles. In this example, the unsatisfiable concept names are $A1$, $A3$, $A6$, $A7$ and MUPS of $\mathcal{T}$ with respect to A_i (i = 1, 3, 6, 7) are:

$$\begin{array}{l} mups(\mathcal{T}, A_1) : \{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\} \\ mups(\mathcal{T}, A_3) : \{ax_3, ax_4, ax_5\} \\ mups(\mathcal{T}, A_6) : \{\{ax_1, ax_2, ax_4, ax_6\}, \{ax_1, ax_3, ax_4, ax_5, ax_6\}\} \\ mups(\mathcal{T}, A_7) : \{ax_4, ax_7\} \end{array}$$

MUPS are useful for relating sets of axioms to the unsatisfiability of specific concepts, but they can also be used to calculate a minimal incoherence-preserving sub-TBox, which relates sets of axioms to the incoherence of a TBox in general and is defined as follows.

Definition 17.3. (Schlobach and Cornet 2003) Let $\mathcal{T}$ be an incoherent TBox. A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a *minimal incoherence-preserving sub-TBox (MIPS)* of $\mathcal{T}$ if $\mathcal{T}'$ is incoherent and every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$ is coherent. The set of all MIPSs of $\mathcal{T}$ is denoted as $MI(\mathcal{T})$.

A MIPS of $\mathcal{T}$ is the minimal sub-TBox of $\mathcal{T}$ which is incoherent. The set of MIPS for a TBox $\mathcal{T}$ is abbreviated with $mips(\mathcal{T})$. For $\mathcal{T}$ in the above example, we get 3 MIPS:

$$mips(\mathcal{T}) = \{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$$

To debug an inconsistent ontology, the notion of a *minimal inconsistent sub-ontology* is proposed in Haase et al. (2005).

Definition 17.4. A minimal inconsistent sub-ontology (MIS) O' of O is an inconsistent sub-ontology of O that has not any proper subset O'' such that O'' is also inconsistent.

There are many algorithms for debugging incoherent DL-based ontologies, which can be classified into two approaches: a glass-box approach and a black-box approach.

A glass-box approach is based on the reasoning algorithm of a DL. The advantage of a glass-box approach is that it can find all MUPS of an incoherent ontology by a single run of a modified reasoner. Most of the glass-box algorithms are obtained as extension of tableau-based algorithms for checking satisfiability of a DL-based ontology. The first tableau-based algorithm for debugging of terminologies of an ontology is proposed in Schlobach and Cornet (2003). The algorithm is restricted to *unfoldable $\mathcal{ALC}$ TBoxes*, that is, the left-hand sides of the concept axioms (the defined concepts) are atomic and the right-hand sides (the definitions) contain no direct or indirect reference to the defined concept. It is realised by attaching label to axioms in order to keep track of which axioms are responsible for assertions generated during the expansion of the completion forests. This algorithm is then extended to more expressive DLs, like OWL DL, in Kalyanpur et al. (2005). As pointed out in Kalyanpur et al. (2007), one problem for the tableau-based algorithms is that some important blocking techniques cannot be used.

A black-box approach treats a DL reasoner as a 'black-box' or an 'oracle' and uses it to check satisfiability of an ontology. The approach is reasoner-independent, in the sense that the DL reasoner is solely used as an oracle to determine concept satisfiability with respect to an ontology. The disadvantage of this approach is that it needs to call the reasoner an exponential number of times in the worst case; thus, it cannot handle large ontologies. Several algorithms belong to this approach, such as those given in Kalyanpur et al. (2005) and Schlobach et al. (2007). The algorithm proposed in Kalyanpur et al. (2005) consists of two main steps. In the first step, it computes a *single* MUPS of the concept and then it utilises the Reiter's hitting set (HS) algorithm to retrieve the remaining ones. Two algorithms are given in Kalyanpur et al. (2007) to compute a single MUPS. One of them first expands a freshly generated ontology O' to a superset of a MUPS using a relevance-based selection function. Then O' is pruned to find the final MUPS, where a window-based pruning strategy is used to minimise the number of satisfiability-check calls to the reasoner. However, the relevant subset O' is expanded very quickly using the syntactic selection function and becomes very large after a small number of iteration. Therefore, they propose the other algorithm which extends a tableau algorithm to find a subset of the original ontology that is a superset of a MUPS, then apply the pruning strategy to find a MUPS. However, this algorithm still cannot handle large ontologies, such as SNOMED CT.

In order to handle large incoherent ontologies, which may naturally exist when ontologies from ontology network are integrated, we propose several optimisations. The first optimisation is based on the syntactic locality-based module defined for OWL DL ontologies in Grau et al. (2007). We generalise the results given in Baader and Suntisrivaraporn (2008) by showing that the syntactic locality-based module of an ontology with respect to an unsatisfiable concept covers all MUPSs of the ontology with respect to the concept in Suntisrivaraporn et al. (2008). As a consequence, it suffices to focus on axioms in the module when finding *all* MUPSs for an unsatisfiable concept. Empirical results demonstrate an improvement of several orders of magnitude in efficiency and scalability of computing all MUPSs in OWL DL ontologies. The second optimisation is based on a *relevance-based selection function* defined in Huang et al. (2005) . We propose a relevance-based algorithm for computing all MUPS of an unsatisfiable concept (Ji et al. 2009). The algorithm iteratively constructs a set of MUPSs of an unsatisfiable concept using a relevance-based selection function. Each MUPS returned by the algorithm is attached with a weight denoting its relevance degree with respect to the unsatisfiable concept.

Computing MISs of an inconsistent ontology can be similar to computing MUPSs of an unsatisfiable concept. That is, we can simply adapt the black-box algorithm for computing MUPS to compute MISs. However, such an algorithm cannot scale to large ABoxes. In Du and Qi (2010), a novel method for computing a set of sub-ontologies from an inconsistent OWL DL ontology is proposed so that the computation of all MISs can be separately performed in each resulting sub-ontology and the union of computational results yields exactly the set of all MISs. Experimental results show that this method significantly improves the scalability for computing all MISs of an inconsistent ontology.

17.3.3 A General Approach for Resolving Inconsistency and Incoherence in Ontology Evolution

In this subsection, we deal with the problem of resolving inconsistency and incoherence. More specifically, we consider this problem in the context of ontology evolution. In this case, the problem is similar to the belief revision in classical logic.

The problem of revision of ontology is described as follows. Suppose we have two ontologies $O = \langle \mathcal{T}, \mathcal{A} \rangle$ and $O' = \left\langle \mathcal{T}', \mathcal{A}' \right\rangle$, where O is the original ontology and O' is the newly received ontology which contains a set of axioms to be added to O. Even if both O and O' are individually consistent and coherent, putting them together may cause inconsistency or incoherence. Therefore, we need to delete or weaken some axioms in O to restore consistency and coherence. Note that when the original ontology is inconsistent or incoherent and the newly received ontology is empty, then the problem of revision of ontology is reduced to the problem of resolving inconsistency and incoherence in a single ontology. Therefore, the approach proposed in this chapter can be also used to deal with a single ontology.

Usually, the result of revision is a set of ontologies rather than a unique ontology (Qi et al. 2006). More formally, we have the following definition of ontology revision. We denote all the possible ontologies with $\mathcal{O}$.

We first introduce the notion of a disjunctive ontology (Meyer et al. 2005). A disjunctive ontology, denoted as **O**, is a set of ontologies. The semantics of the disjunctive ontology is defined as follows (Meyer et al. 2005):

Definition 17.5. A disjunctive ontology **O** is satisfied by an interpretation $\mathcal{I}$ (or $\mathcal{I}$ is a model of **O**) iff $\exists O \in \mathbf{O}$ such that $\mathcal{I} \models O$. **O** entails ϕ, denoted $\mathbf{O} \models \phi$, iff every model of **O** is a model of ϕ.

Definition 17.6. An ontology revision operator (or revision operator for short) in DLs is a function $\circ : \mathcal{O} \times \mathcal{O} \to \mathcal{P}(\mathcal{O})$ which satisfies the following conditions: (1) $O \circ O' \models \phi$ for all $\phi \in O'$, where $\mathcal{P}(\mathcal{O})$ denotes all the subsets of $\mathcal{O}$ and (2) for each $O_i \in O \circ O'$, O_i is consistent.

That is, an ontology revision operator is a function which maps a pair of ontologies to a disjunctive ontology which can consistently infer the newly received ontology. In practice, we may only need one ontology after revision. In this case, we can obtain such an ontology by ranking the ontologies obtained by the revision operator and then selecting the one with highest rank. Ranking of ontologies can either be given by the users or be computed by some measures, such as ranking of test cases and syntactic relevance (see Kalyanpur et al. (2006) for more details).

The current work on ontology revision suffers from some problems, to name a few, we have the following ones:

- There is much work on the analysis of applicability of AGM postulates for belief change to DLs (Flouris et al. 2005, 2006). However, few of them discuss the concrete construction of a revision approach.
- Current revision approaches often focus on dealing with logical inconsistency. Another problem which is as important as inconsistency handling is incoherence handling, where an ontology is incoherent if and only if there is an unsatisfiable named concept in its terminology. As analysed in Flouris et al. (2006), logical incoherence and logical inconsistency are not independent of each other. A revision approach which resolves both logical incoherence and inconsistency is missing.

We now propose our general approach which resolves incoherence and inconsistency in an integrated way. The approach consists of the process steps shown in Fig. 17.2. In this process, problems that are related only with either the TBox or the ABox are dealt with independently in two separate threads (c.f. left and right thread of Fig. 17.2, respectively). For the TBox, inconsistency resolution is done before incoherence resolution because incoherence is a consequence of inconsistency in the TBox. We first check if $\mathcal{T} \cup \mathcal{T}'$ is consistent. If it is not, then we resolve the inconsistency. This can be done by either deleting the erroneous terminology axioms or weakening them. In a next step, we resolve incoherence. There are several ways to resolve incoherence. The commonly used technique is to remove some (usually minimal numbers) of erroneous terminology axioms which are responsible for the incoherence. Alternatively, we can take the maximal coherent sub-ontologies of $\mathcal{T}$ with respect to $\mathcal{T}'$ as the result of revision (Meyer et al. 2006; Lam et al. 2006). For the ABox, we resolve inconsistencies that occur only due to

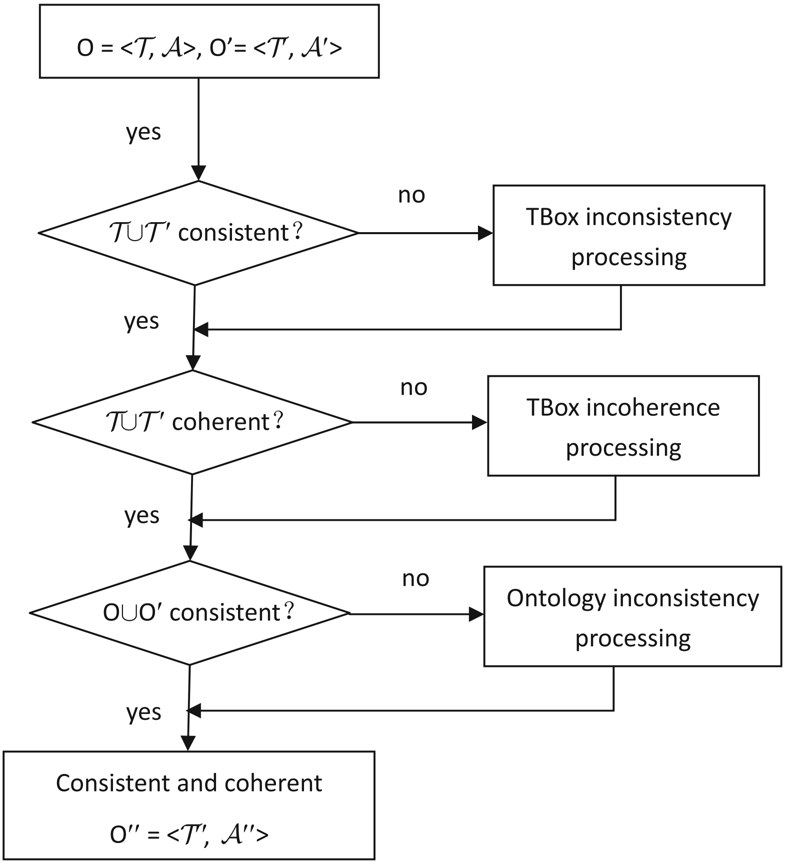


Fig. 17.2 Approach to resolving inconsistency and incoherence

assertional axioms. This can be done by either deleting the assertional axioms which are responsible for the inconsistency or weakening them. The weakening of assertional axioms may be different from that of terminology axioms. Finally, we deal with the inconsistency due to both terminology and assertional axioms. The resulting ontology is $O'' = \langle \mathcal{T}'', \mathcal{A}'' \rangle$, which is both consistent and coherent. In each of the revision steps, the result may be a disjunctive ontology, since there may exist several alternative ways to resolve the incoherence or inconsistency. However, in each step, a decision is made: Which single ontology should be selected as input for the subsequent step. This decision can be made either by the user or an automated procedure based on a ranking of the results as discussed above.

Our general approach does not yet specify *how* to deal with inconsistency or incoherence. Moreover, for different kinds of inconsistency, we can use different strategies to resolve them. For example, when resolving inconsistency due to terminology axioms, we can take the maximal consistent subsets of the original TBox with respect to the new TBox as the result of revision. Whilst when resolving inconsistency related to assertional axioms, we can apply the revision approach in Qi et al. (2006), which removes minimal number of individual resulting in the conflict. In the next section, we instantiate our approach by proposing a concrete approach to resolving incoherence and some concrete approaches to resolving inconsistency.

17.3.4 Repairing Ontology Mappings

There has been some work on handling inconsistency in distributed ontologies connected via mappings, where a mapping between two ontologies is a set of correspondences between entities in the ontologies. In a distributed system consisting of two ontologies and a mapping between them, correspondences in the mapping can have different interpretations. For example, in distributed description logics (DDL) (Borgida and Serafini 2003), a correspondence in a mapping is translated into two *bridge rules* that describe the 'flow of information' from one ontology to another one. In Meilicke et al. (2007), the authors deal with the problem of mapping revision in DDL by removing some bridge rules which are responsible for the inconsistency. The idea of their approach is similar to that of the approaches for debugging and repairing terminologies in a single ontology. Mappings can also be interpreted as sets of axioms in a description logic. A heuristic method for mapping revision is given in Meilicke and Stuckenschmidt (2007). However, this method can only deal with inconsistency caused by disjointness axioms which state that two concepts are disjoint. Later on, Meilicke et al. proposed another algorithm to resolve the inconsistent mappings in Meilicke et al. (2008). The idea of their algorithm is similar to the linear base revision operator given in Nebel (1994). However, both methods given in Meilicke and Stuckenschmidt (2007) and Meilicke et al. (2008) lack a rationality analysis with respect to logical properties.

In Qi et al. (2009), a conflict-based mapping revision operator is proposed based on the notion of a 'conflict set', which is a sub-set of the mapping that is in conflict with ontologies in a distributed system. A postulate from belief revision theory (Hansson 1993) is adapted, and it is shown that the mapping revision operator can be characterised by it. After that, an iterative algorithm for mapping revision is given by using a revision operator in description logics, and it is shown that this algorithm results in a conflict-based mapping revision operator. A revision operator is given, and it is shown that the iterative algorithm based on it produces the same results as the algorithm given in Meilicke et al. (2008). This specific iterative algorithm has a polynomial time complexity if the satisfiability check of an ontology can be done in polynomial time in the size of the ontology. However, this algorithm may still be inefficient for large ontologies and mappings because it requires a large number of satisfiability checks. Therefore, an algorithm to implement an alternative revision operator based on the *relevance-based selection function* given in Huang et al. (2005) is proposed and is further optimised by a module extraction technique given in Suntisrivaraporn et al. (2008). Neither of the above proposed revision operators removes minimal number of correspondences to resolve inconsistencies. To better fulfil the principle of minimal change, we consider the revision operator given in Qi et al. (2008) which utilises a heuristics based on a *scoring function* which returns the number of *minimal incoherence-preserving sub-ontologies (MIPS)* that an axiom belongs to. Instantiating the iterative algorithm with this existing revision operator results in a new conflict-based mapping revision operator.

17.4 Main Functionalities

In the following, we discuss the main functionality provided by the plugins which cover reasoning and query processing with networked ontologies.

17.4.1 Reasoning Functionality

The installed reasoner materialises inferences and combines them with the ontology, over which the query evaluation is carried out. Table 17.1 lists the axioms which are supported for materialisation via the OWL API (Bechhofer et al. 2003). Users may specify which of the axioms the reasoner should take into account in the materialisation procedure. The reasoner plugin also provides a simple consistency check which returns whether an ontology is consistent or not.

17.4.2 Diagnosing and Resolving Inconsistencies

Before representing the functionalities of RaDON, some terminologies involved are introduced first:

Table 17.1 Supported axioms for materialisation

Axiom
SubClass
DisjointClasses
PropertyAssertion
ClassAssertion
SubDataProperty
EquivalentClass
SubObjectProperty
DataPropertyCharacteristic
EquivalentDataProperties
EquivalentObjectProperty
InverseObjectProperties
ObjectPropertyCharacteristic

- Unsatisfiable concept: A named concept C in an ontology O is unsatisfiable if and only if, for each model of O, the interpretation of C is empty.
- Incoherent ontology: An ontology O is incoherent if there exists an unsatisfiable concept in O.
- Inconsistent ontology: An ontology O is inconsistent if and only if it has no model.

RaDON provides a set of techniques for dealing with inconsistency and incoherence in ontologies. In particular, RaDON supports novel strategies and consistency models for distributed and networked environments.

RaDON extends the capabilities of existing reasoners with the functionalities to deal with inconsistency and incoherence. Specifically, the functionalities provided by RaDON include:

- Debugging an incoherent or an inconsistent ontology to explain why a concept is unsatisfiable or why the ontology is inconsistent.
- Repairing an ontology automatically by computing all possible explanations with respect to all unsatisfiable concepts if the ontology is incoherent, or with respect to the inconsistent ontology if it is inconsistent.
- Repairing an ontology manually based on the debugging results. For the manual repair, the user can choose the axioms to be removed for restoring the coherence or consistency.
- Repairing a mapping between two ontologies.
- Coping with inconsistency based on a paraconsistency-based algorithm.

17.4.3 Query Functionality

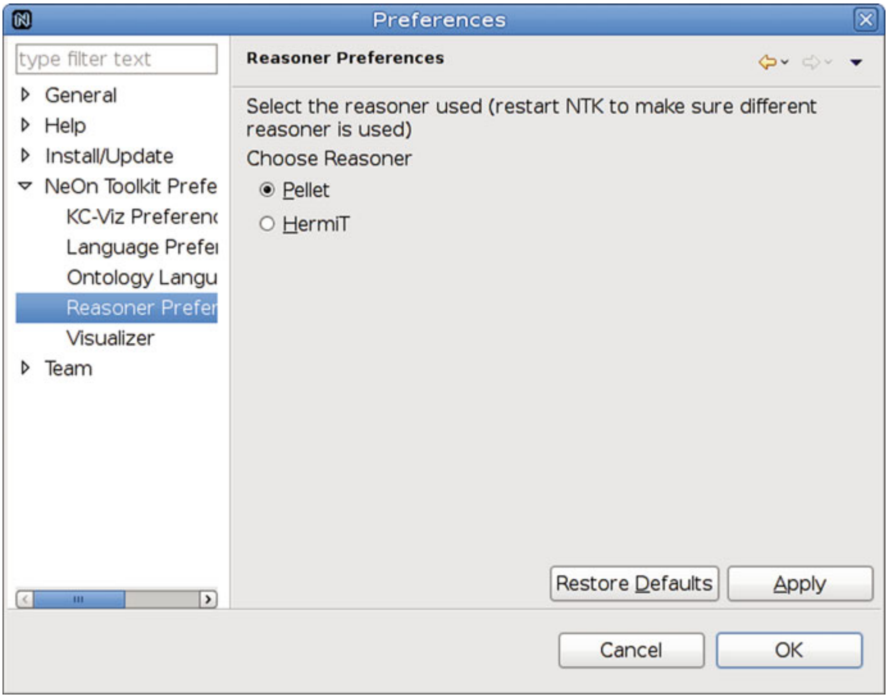
The query plugin allows for query answering over local ontologies residing in memory in the OWL API by using the ARQ query processor included in the Jena Semantic Web framework (Carroll et al. 2004). Results of the query evaluation can be further edited within the NeOn Toolkit.

17.5 Summary of Usage

In the following we show how to operate the plugins.

17.5.1 Reasoning Plugin

The reasoning plugin provides common access to reasoners for NeOn Toolkit components. The plugin can be configured in a Preferences view shown in Fig. 17.3.



Reasoner Preferences

Select the reasoner used (restart NTK to make sure different reasoner is used)

Choose Reasoner

● Pellet

○ HermiT

Fig. 17.3 Reasoner Preferences selection screenshot

Currently, two reasoners are supported, namely Pellet (Sirin et al. 2007) and HermIT (Shearer et al. 2008). The selected preferences apply to all NeOn Toolkit components.

17.5.2 RaDON Plugin

RaDON provides two plugins to deal with a single ontology or an ontology network. In the plugin of 'Repair a Single Ontology', the following specific functionalities are provided:

- Handle incoherence: This functionality corresponds to the button of 'Handle Incoherence' which can be activated if the ontology is incoherent. That is, there is at least one unsatisfiable concept in the ontology. All the minimal unsatisfiability-preserving subsets (MUPS) can be computed for each unsatisfiable concept.
- Handle inconsistency: This corresponds to the button of 'Handle Inconsistency' which is activated if the ontology is inconsistent. That is, there is no model for the ontology. All the minimal inconsistent subsets (MIS) can be calculated.

- Repair automatically: This corresponds to the button of ‘Repair Automatically’. If the button of ‘Repair Automatically’ is pressed, our algorithm will provide some axioms to be removed to keep the coherence or consistency of the ontology. Specifically, this can be done by computing the minimal incoherence-preserving subsets (MIPS) or MIS, respectively, and then choosing automatically some axioms to be removed.
- Repair manually: This corresponds to the button of ‘Repair Manually’. If this button is activated, a new dialogue will be shown with the information of MIPS or MIS which are computed based on the found MUPS or MIS, respectively. The user could choose the axioms to be removed by themselves.

In the plugin of ‘Repair and Diagnose Ontology Network’, the similar functionalities in the plugin above are given. The main difference is that this plugin is to repair and diagnose a mapping between two ontologies by assuming the two source ontologies are more reliable than the mapping itself.

17.5.3 SPARQL Query Plugin

The query plugin can be invoked using the ‘SPARQL Query’ context menu, which starts the SPARQL view.

Users first load the ontology into the SPARQL query processor. Optionally, users can load an ontology and materialise inferences at the same time. Having loaded an ontology, users can specify a SPARQL query against the loaded ontology. For example, the following SPARQL query lists all instances of type owl:Class.

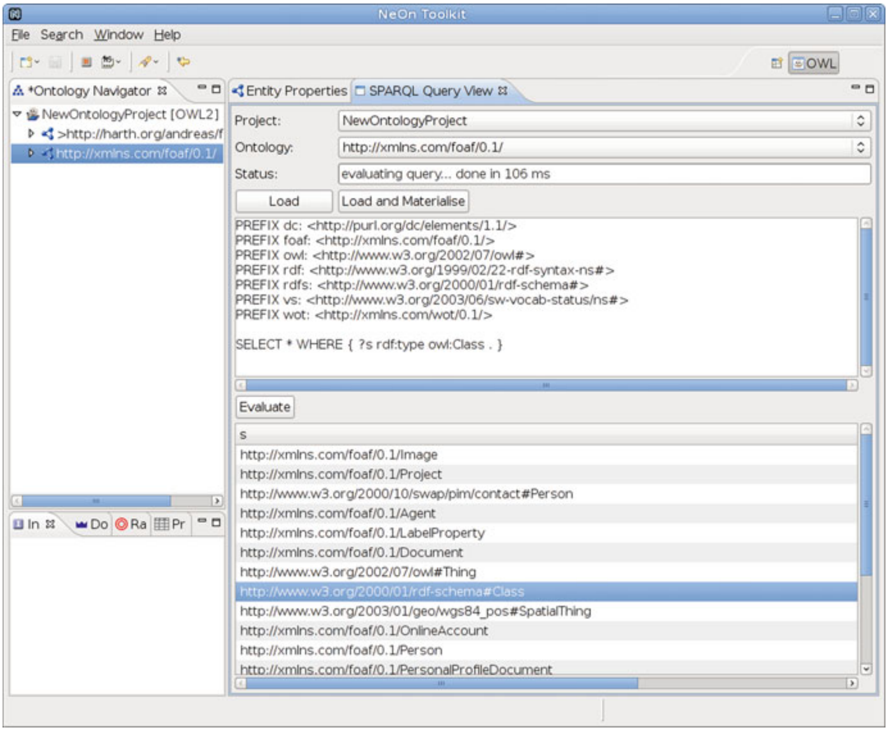
```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE– { ?s rdf:type owl:Class . }
```

When users click the ‘Evaluate’ button, the plugin generates a table with the variable bindings. Figure 17.4 shows the SPARQL plugin with results for a query.

In addition to SPARQL SELECT queries which return a table with variable bindings, the plugin also evaluates CONSTRUCT queries which return an RDF graph. The returned RDF graph can be used to create a new ontology in the NeOn Toolkit. Thus, the SPARQL query allows for the selection of parts of an ontology and further refinement inside the Toolkit.

17.6 Conclusion

In this chapter we have discussed the functionality and workings of NeOn Toolkit’s reasoning and query plugins. Reasoning tasks are important in knowledge engineering, for example, to check for logical consistency of modelled artefacts. The plugins provide users with advanced reasoning and query answering functionality over networked ontologies.



Project: NewOntologyProject

Ontology: http://xmlns.com/foaf/0.1/

Status: evaluating query... done in 106 ms

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX vs: <http://www.w3.org/2003/06/sw-vocab-status/ns#>
PREFIX wot: <http://xmlns.com/wot/0.1/>

SELECT * WHERE { ?s rdf:type owl:Class . }
```

s
http://xmlns.com/foaf/0.1/Image
http://xmlns.com/foaf/0.1/Project
http://www.w3.org/2000/10/swap/pim/contact#Person
http://xmlns.com/foaf/0.1/Agent
http://xmlns.com/foaf/0.1/LabelProperty
http://xmlns.com/foaf/0.1/Document
http://www.w3.org/2002/07/owl#Thing
http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing
http://xmlns.com/foaf/0.1/OnlineAccount
http://xmlns.com/foaf/0.1/Person
http://xmlns.com/foaf/0.1/PersonalProfileDocument

Fig. 17.4 Query plugin screenshot

References

Baader F, Suntisrivaraporn B (2008) Debugging SNOMED CT using axiom pinpointing in the description logic EL+. In: Proceedings of the 3rd international conference on Knowledge Representation in Medicine (KR-MED 2008), Phoenix, AZ, USA

Baader F, Calvanese D, McGuinness D, Nardi D, Patel-Schneider P (2003) The description logic handbook: theory, implementation and application. Cambridge University Press, Cambridge

Bechhofer S, Volz R, Lord P (2003) Cooking the semantic web with the owl api. In: The Semantic Web – ISWC 2003, Lecture notes in computer science, vol 2870. Springer, Berlin/Heidelberg, pp 659–675. doi:10.1007/978–3–540–39718–2 42

Borgida A (1994) On the relationship between description logic and predicate logic. In: Proceedings of the 3rd international conference on Information and Knowledge Management (CIKM 1994), Gaithersburg, MD. ACM, New York, pp 219–225

Borgida A, Serafini L (2003) Distributed description logics: assimilating information from peer sources. J Data Semant 1:153–184

Carroll JJ, Dickinson I, Dollin C, Reynolds D, Seaborne A, Wilkinson K (2004) Jena: implementing the semantic web recommendations. In: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. ACM, New York, NY, USA, WWW Alt. 2004, pp 74–83, DOI http://doi.acm.org/10.1145/1013367.1013381, URL http://doi.acm.org/10.1145/1013367.1013381

Clark KG, Feigenbaum L, Torres E (2008) SPARQL protocol for RDF. W3C Recommendation. http://www.w3.org/TR/rdf-sparql-protocol/

Du J, Qi G (2010) Decomposition-based optimization for debugging of inconsistent owl dl ontologies. In: Proceedings of the 4th international conference on Knowledge Science, Engineering and Management (KSEM 2010), Belfast, Northern Ireland, UK, pp 88–100

Flouris G, Plexousakis D, Antoniou G (2005) On applying the AGM theory to DLs and OWL. In: Proceedings of the 4th International Conference on Semantic Web (ISWC 2005), Galway, Ireland, pp 216–231

Flouris G, Huang Z, Pan JZ, Plexousakis D, Wache H (2006) Inconsistencies, negations and changes in ontologies. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006), Boston, MA, pp 1295–1300

Grau BC, Horrocks I, Kazakov Y, Sattler U (2007) Just the right amount: extracting modules from ontologies. In: Proceedings of the 16th international conference on World Wide Web (WWW 2007), Banff, AB, Canada, pp 717–726

Haase P, van Harmelen F, Huang Z, Stuckenschmidt H, Sure Y (2005) A framework for handling inconsistency in changing ontologies. In: Proceedings of the 4th International Semantic Web Conference (ISWC 2005), Galway, Ireland, pp 353–367

Hansson SO (1993) Reversing the levi identity. J Philos Log 22(6):637–669

Hayes P (2004) RDF semantics. W3C Recommendation. http://www.w3. org/TR/rdf-mt/

Huang Z, van Harmelen F, ten Teije A (2005) Reasoning with inconsistent ontologies. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, Scotland, UK, pp 454–459

Ji Q, Qi G, Haase P (2009) A relevance-directed algorithm for finding justifications of dl entailments. In: Proceedings of the 4th Asian Conference on Semantic Web (ASWC 2009), Shanghai, China, pp 306–320

Kalyanpur A, Parsia B, Sirin E, Hendler J (2005) Debugging unsatisfiable classes in OWL ontologies. J Web Semant 3(4):268–293

Kalyanpur A, Parsia B, Sirin E, Grau BC (2006) Repairing unsatisfiable concepts in owl ontologies. In: Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), Budva, Montenegro, pp 170–184

Kalyanpur A, Parsia B, Horridge M, Sirin E (2007) Finding all justifications of OWL DL entailments. In: Proceedings of the 6th International Semantic Web Conference (ISWC 2007), Busan, Korea, pp 267–280

Lam J, Pan JZ, Seeman D, Vasconcelos W (2006) A fine-grained approach to resolving unsatisfiable ontologies. In: Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence (WI 2006), Hong Kong, pp 428–434

Meilicke C, Stuckenschmidt H (2007) Applying logical constraints to ontology matching. In: Proceedings of the 30th annual German conference on Artificial Intelligence (KI 2007), Osnabrück, Germany, pp 99–113

Meilicke C, Stuckenschmidt H, Tamilin A (2007) Repairing ontology mappings. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007), Vancouver, BC, Canada, pp 1408–1413

Meilicke C, Völker J, Stuckenschmidt H (2008) Learning disjointness for debugging mappings between lightweight ontologies. In: Proceedings of the 16th international conference on Knowledge Engineering: Practice and Patterns (EKAW 2008), Acitrezza, Italy, pp 93–108

Meyer T, Lee K, Booth R (2005) Knowledge integration for description logics. In: Proceedings of 20th national conference on Artificial Intelligence (AAAI 2005). AAAI Press, Pittsburgh, PA, pp 645–650

Meyer T, Lee K, Booth R, Pan JZ (2006) Finding maximally satisfiable terminologies for the description logic ALC. In: Proceedings of 21th national conference on Artificial Intelligence (AAAI 2006), Boston, MA, pp 269–274

Minsky M (1974) A framework for representing knowledge. Massachusetts Institute of Technology, Cambridge

Nebel B (1994) Base revision operations and schemes: semantics, representation and complexity. In: Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI 1994), Amsterdam, the Netherlands, pp 341–345

Prud'hommeaux E, Seaborne A (2008) SPARQL query language for RDF. W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query/

Qi G, Liu W, Bell DA (2006) Knowledge base revision in description logics. In: Proceedings of the 10th European conference on logics in artificial intelligence (JELIA 2006). Springer, Liverpool, UK, pp 386–398

Qi G, Haase P, Huang Z, Ji Q, Pan JZ, Völker J (2008) A kernel revision operator for terminologies – algorithms and evaluation. In: Proceedings of the 7th International Semantic Web Conference (ISWC 2008), Karlsruhe, Germany, pp 419–434

Qi G, Ji Q, Haase P (2009) A conflict-based operator for mapping revision. In: Proceedings of the 8th International Semantic Web Conference (ISWC 2009), Chantilly, VA, USA, pp 521–536

Schlobach S, Cornet R (2003) Non-standard reasoning services for the debugging of description logic terminologies. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003), Acapulco, Mexico, pp 355–362

Schlobach S, Huang Z, Cornet R, van Harmelen F (2007) Debugging incoherent terminologies. J Autom Reason 39(3):317–349

Schmidt-Schau M, Smolka G (1991) Attributive concept descriptions with complements. Artif Intell 48(1):1–26

Shearer R, Motik B, Horrocks I (2008) HermiT: a highly-efficient OWL reasoner. In: Ruttenberg A, Sattler U, Dolbear C (eds) Proceedings of the 5th international workshop on OWL: Experiences and Directions (OWLED 2008 EU), Karlsruhe, Germany

Sirin E, Parsia B, Grau BC, Kalyanpur A, Katz Y (2007) Pellet: a practical owl-dl reasoner. Web Semant 5:51–53. doi:10.1016/j.websem.2007.03.004, URL http://portal.acm.org/citation.cfm?id= 1265608.1265744

Suntisrivaraporn B, Qi G, Ji Q, Haase P (2008) A modularization-based approach to finding all justifications for owl dl entailments. In: Proceedings of the 3rd Asian Semantic Web Conference (ASWC 2008), Bangkok, Thailand, pp 1–15

Part IV
Case Studies

Chapter 18
Knowledge Management at FAO: A Case Study on Network of Ontologies in Fisheries

Caterina Caracciolo, Juan Heguiabehere, Aldo Gangemi, Claudio Baldassarre, Johannes Keizer, and Marc Taconet

Abstract In this chapter, we illustrate the work conducted at the Food and Agriculture Organization of the United Nations (FAO) with the creation of a network of ontologies about fisheries, developed with NeOn technologies and methodologies. The network included the main thematic areas needed to talk about fish stocks (often referred to as aquatic resources) and included data sources of various types: reference data for time series, thesauri for document indexing, actual time series, and the reuse of an existing well-known ontology maintained by FAO (the geopolitical ontology). Such a network of ontologies was also used within a prototypical web-based application. After describing the methodologies used to create the network, and its contents and features, we draw some conclusions and highlight the lessons learned during the process.

C. Caracciolo (✉) • J. Keizer
Food and Agriculture Organization of the United Nations (FAO of the UN), OEK, v.le Terme di Caracalla 1,
00154 Rome, Italy
e-mail: caterina.caracciolo@fao.org; johannes.keizer@fao.org

J. Heguiabehere
Facultad de Ingeniería, Universidad de Buenos Aires, Paseo Colón 850, Buenos Aires, Argentina C1063ACV
e-mail: jheguiabehere@fi.uba.ar

A. Gangemi
Semantic Technologies Lab, Institute of Cognitive Sciences and Technologies (National Research Council – CNR), Via Nomentana 56, 00161 Rome, Italy
e-mail: aldo.gangemi@cnr.it

C. Baldassarre • M. Taconet
Food and Agriculture Organization of the UN (FAO), FIES, v.le Terme di Caracalla 1,
00154 Rome, Italy
e-mail: claudio.baldassarre@fao.org; marc.taconet@fao.org

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_18, © Springer-Verlag Berlin Heidelberg 2012

List of Acronyms

ASFA	Aquatic Sciences and Fisheries Abstracts
ASFIS	Aquatic Science and Fisheries Information System
CWP	Coordinating Working Party on Fishery Statistics
EEZ	Exclusive Economic Zones
FAO	Food and Agriculture Organization of the United Nations
FIES	Fisheries and Aquaculture Department, FAO
FSDAS	Fisheries Stock Depletion Assessment System
HS	Harmonized System
ISSCAAP	International Standard Statistical Classification for Aquatic Animals and Plants
ISSCFC	International Standard Statistical Classification of Fishery Commodities
ISO	International Standard Organization
LMEs	Large marine ecosystems
NAFO	Northwest Atlantic Fisheries Organization
NOAA	US National Oceanic and Atmospheric Administration
OAEI	Ontology Alignment Evaluation Initiative
ODP	Ontology Design Patterns Portal (www.ontologydesignpatterns.org)
OEK	Office of Knowledge Exchange, Research and Extension
SITC	Standard International Trade Classification
UNDP	United Nations Development Programme

18.1 Introduction

The Food and Agriculture Organization of the United Nations (FAO) has collected data about food and agriculture since its foundation in 1945. Since the beginning, metadata has been used to annotate, organize, and classify data, as in the case of thesauri used to index documents[1] and reference data for statistics, i.e., the concepts used as dimensions of a piece of statistical data[2] to store and retrieve statistics[3]. However, compared to most modern information systems, one may notice some limitations.

First of all, especially for what concerns reference data, much of the actual relationships between the objects that are referenced are kept away from it. For example, there is reference data for species and reference data for fishing areas, but

[1] http://www.fao.org/documents/en/docrep.jsp

[2] Reference data may be considered a specific type of metadata for statistics. Other types of metadata are about data provenance and methodology for data creation.

[3] For a list of statistical databases on fisheries maintained by FAO, see http://www.fao.org/fishery/statistics/en

if users want to know something about what species is found in a given fishing area, they should search for this information in other information systems, where data about species distribution is available. The result of this situation is that it is well possible to query the system for time series about “catch of bluefin tuna (*Thunnus thynnus*, Linnaeus, 1758) in the Indian Ocean,” although bluefin tuna is actually found in open waters of tropical and subtropical seas worldwide. Moreover, data is usually collected and stored according to one specific classification system. This implies that the only way to query a database is by using exactly the same classification used for storing the data. Conversion of data according to different classifications is then a time-consuming task that only domain experts may reliably perform. Obviously, the establishment of correspondences between classification systems is a delicate task that falls outside the scope of an information management project, but a modern information management approach should enable easy conversion between one system and the other, any time when the conversion is made available by experts.

For these reasons, the data owned by FAO is a good application area to prove and refine the technologies developed within the NeOn project. We concentrated on the domain of fisheries, as it is a good example of domain where the possibility of linking together different information systems is crucial. The Fisheries and Aquaculture Department (FIES) of FAO publishes reports about various aspects of fisheries, including aquatic resources, on a regular basis. Reports are usually based on data contributed by different institutions, often stored in different information systems and encoded in various formats. Therefore substantial data integration effort is usually required. In our use case on fisheries, we worked on the hypothesis that the conversion of data and metadata into a network of ontologies could lead to improved information sharing. The experimental application we designed is a Fisheries Stock Depletion Assessment System (FSDAS)[4].

In the rest of this chapter, we present our experience with the making of a network of ontologies and on the application based on that. In Sect. 18.2, we illustrate the domain and data used for the use case. In Sect. 18.3, we describe the methodology followed for the creation of the network. In Sect. 18.4, we present the resulting network of ontologies and highlight its features. In Sect. 18.5, we present the ontology-based system we developed. Finally, in Sect. 18.6, we draw our conclusion and hint at future work.

[4] For details about the requirements driving this work in the context of the use case, see Iglesias Sucasas et al. (2007).

18.2 Domains and Data

A fish stock is a subpopulation of particular species of fish with some definable attributes and living in definable marine areas. A simplified notion of fish stock is the one of marine resource: a fish stock considered from a management perspective. We adopted the operational notion of marine resource and concentrated on a few entities considered crucial to talk about it: first of all, marine species, but also water areas (needed to indicate where species are caught) and land areas (to keep track of legal dependency of water areas and vessels). We also considered other entities, such as fisheries commodities and fishing techniques.

As for the actual data sets, we considered a relevant subset of the data set available in FAO: a set of statistics on fisheries (mainly catch statistics) and documents about fisheries. The corresponding metadata (i.e., reference data for statistics and thesauri for documents) have been reengineered into a network of ontologies. We also reused and included in the network the FAO geopolitical ontology, used to aggregate information about countries and to operate the FAO Country Profiles portal[5]. In the following, we first provide some explanations about the entities we considered and then describe the data sets we used.

18.2.1 Data Sets Included in the Network of Ontologies

We have considered three distinct types of metadata sources:

1. Reference data used to identify the “dimensions” of a piece of statistical data (we focused on catch data)
2. Thesauri used to index documents about fisheries: AGROVOC[6] and ASFA thesaurus[7]
3. The geopolitical ontology[8]. an ontology about geopolitical information maintained by FAO

Reference data. A time series is a sequence of statistical observations ordered in time. FIES collects observations about catches (or captures), aquaculture production, fleets, trade of commodities, among others[9]. Each piece of statistical data is

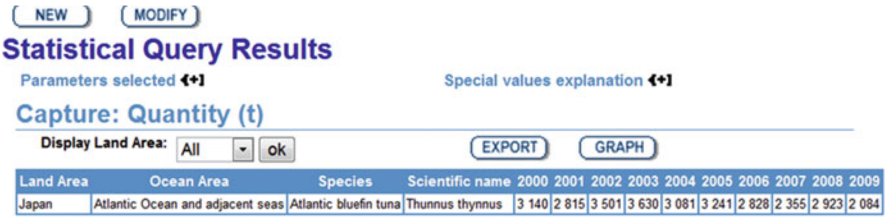
[5] http://www.fao.org/countryprofiles/

[6] http://aims.fao.org/website/AGROVOC-Thesaurus/sub

[7] http://www4.fao.org/asfa/asfa.htm

[8] http://www.fao.org/countryprofiles/geoinfo.asp?lang=en

[9] Detailed information regarding fisheries statistics can be found in the Handbook of Fishery Statistical Standards [HBFSS] by the Coordinating Working Party on Fishery Statistics (CWP). The Coordinating Working Party on Fishery Statistics (CWP) supported by its participating organizations has served since 1960 as the premier international and inter-organization forum for agreeing upon common definitions, classifications, and standards for the collection of fishery statistics.



Statistical Query Results

Capture: Quantity (t)

Display Land Area: All ok

Land Area	Ocean Area	Species	Scientific name	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
Japan	Atlantic Ocean and adjacent seas	Atlantic bluefin tuna	Thunnus thynnus	3 140	2 815	3 501	3 630	3 081	3 241	2 828	2 355	2 923	2 084

Fig. 18.1 Example of time series

referenced by the following dimensions: time (in years), space (land and/or water areas), and the variable representing the observed object (e.g., biological species, vessels, commodities).

Figure 18.1 shows an example of time series about fish catch. Columns 1–3 represent the dimensions, or reference data, of the statistical data (i.e., land area, water area, and species), while the last 10 columns report the yearly observations collected. Information systems in FIES organize reference data into tables (reference tables (RT)) that store the *codes* assigned to each reference object according to one or more *coding system* maintained by international organizations. They also store the association between codes and names in one or more languages (usually English, French, and Spanish). Correspondence between languages is one to one because it results from international agreements (e.g., on names of territories, on commodities classifications). The entire system that manages the RT is called reference tables' management system (RTMS)[10], whose core is an oracle database.

Reference data is also used in the fisheries fact sheets produced by FIES, where a large amount of information about fisheries, aquaculture, and related subjects (including fishing techniques, fishing areas, fisheries and aquaculture country profiles) is made available to the public in the form of semistructured text. All fisheries fact sheets in FAO are in XML format, structured according to a comprehensive XML schema that includes all elements used in all types of fact sheets. Fact sheets are organized by domains (e.g., cultured species, fishing equipment, fishery, gear type), each corresponding to an element under the root FIGISdoc, the root of any fact sheet (XML document). Domains are fully specified by means of nested elements. Each element includes a description meant for human use.

FAO thesauri. We considered two thesauri: AGROVOC and ASFA. *AGROVOC* is the FAO's corporate thesaurus, covering all domains of interest to FAO, including fisheries. AGROVOC is currently available in 19 languages, and 5 more are under development[11]. AGROVOC is available in various formats, and at the time of the NeOn use case, an OWL version was under study. Currently, it is encoded in

[10] Reference tables are browsable at: http://www.fao.org/fishery/rtms/en

[11] Issues related to the treatment of multilinguality in AGROVOC are presented in Caracciolo and Sini (2007).

SKOS, although several other formats are still available to users. *ASFA* is the thesaurus developed for the Aquatic Sciences and Fisheries Abstracts (ASFA) partnership, an international agreement between institutions active in the area of fisheries, to collect abstracts relevant to the domain. ASFA is available in three languages: English, French, and Spanish.

The *FAO geopolitical ontology* is a repository of information (e.g., names in various languages, codes, historical changes, geographical coordinates) about geopolitical entities such as countries and groups. It is largely used within FAO and by partners for the purpose of querying information systems using codes for internal storage of data about geopolitical entities (Kim et al. 2009). It is available as an OWL ontology and as RDF(S) data published in the Linked Open Data style.

18.2.2 Coverage of the Fisheries Network of Ontologies

Once identified the resources to use for our use case, we went on selecting the domain entities needed to ensure appropriate coverage of the network of ontologies and consequent use in the FSDAS. For example, aquatic species and water areas are necessary to define aquatic resources; land areas, together with aquatic species and water areas, are necessary to reference catch statistics. Other entities are useful to integrate our view on the fisheries domain, such as commodities (for production data), fishing gears, and fishing vessels. The user not acquainted with statistics and fisheries should keep in mind that names are important to enable users to search across different data sets, while codes are important for actual storage and retrieval of data.

In this section, we concentrate on the most important entities needed to understand and reference fisheries data: aquatic species, water areas, land areas, and aquatic resources.

Aquatic species. FAO maintains a list of aquatic species of interest to FAO (both for fisheries and aquaculture), and about which data is collected: the ASFIS (Aquatic Science and Fisheries Information System) list[12], which currently includes nearly 11,000 items[13]. Each species is provided with a taxonomic code, which reflects a biological point of view (simplified taxonomic classification), used for the purpose of data aggregation along taxonomic lines. Species are also given an ISSCAAP[14] code, whose purpose is aggregation according to groups formed according to a commercial point of view. Species are also given an "inter-agency 3-alpha code", which is a compact way to represent them and is used for data exchange across UN agencies. As for names, only one preferred name in English,

[12] http://www.fao.org/fishery/collection/asfis/en

[13] ASIFS also includes taxonomic entities above the species level such as families or orders, on the basis of the data reported to FAO by countries or other governing bodies.

[14] International Standard Statistical Classification for Aquatic Animals and Plants (ISSCAAP).

Spanish, and French is taken for each species. The English name is available for most of the records, and about one-third of them also have a French and Spanish name[15].

Water areas. Marine and inland waters are divided into a variety of zones and areas, depending on the purpose of the division (e.g., legal jurisdiction, statistical reporting of catch data, environmental assessment) and on the author of the division (e.g., national or international body). For our purpose, the most important organization of water areas is the FAO Division Areas, used by FAO and partners for data collection and statistical reporting. This is a system of 27 major areas, divided into subareas, each divided into divisions, and these finally into subdivisions, covering marine waters as well as inland waters[16]. We also considered large marine ecosystems (LMEs): regions of the world's oceans classified by the US National Oceanic and Atmospheric Administration (NOAA)[17] to identify areas of the oceans for conservation purposes. LMEs have great economic and nutritional importance, as the 64 classified LMEs produce 95% of the world's annual marine fishery biomass yields. We also considered exclusive economic zones (EEZ), i.e., sea zones over which a state has special rights with regards to the exploration and use of marine resources. The importance of EEZ cannot be overestimated, as can be witnessed by the disputes between states over marine waters. From the point of view of data collection and fisheries management, it is important to map the overlap between EEZ and FAO divisions used for statistical data reporting.

Land areas. Land territories are central to most statistical collections. Fisheries data is no exception since fish catches are either assigned to the country of the flag flown by the fishing vessel or to the country where the vessels lands. Either way, one may not forget territorial information. The names of territories (countries and groups) are established by international agreements. By agreement, at least two types of names of territory are given in each language: long names to be used in official documents and short names to be used in informal communications. A variety of codes are used for land areas. Most remarkably, ISO codes[18] are widely used and so are the codes established by the United Nations and its agencies, such as the United Nations Development Programme (UNDP), the UN Statistical Division, and by FAO. Each coding system tends to be specific to the purpose and application for which they are used.

Fish stock aka aquatic resources. From a biological point of view, a stock comprises all the individuals of fish in an area, which are part of the same reproductive process. A stock occupies a well-defined spatial range and is

[15] Member agencies of the CWP have agreed to use these standard species names in statistical publications and questionnaires. However, (a) it has not been possible to assign appropriate names in all three languages to all species items, and (b) these names may not correspond with nationally or regionally used common names.

[16] http://www.fao.org/fishery/area/search/en

[17] http://www.lme.noaa.gov/

[18] ISO codes are established by the International Standard Organization (ISO).

independent of other stocks of the same species. When dealing with fishing management, though, it is common to use the notion of aquatic resource, to vaguely defined "stocks", especially for management purposes. Just like a stock, a fishery resource is defined in space and its geographical demarcation and often has a political or jurisdictional connotation (e.g., Moroccan resources, exclusive economic zones (EEZ), or high seas resources).

18.3 Methodology for the Creation of the Network

The methodologies for creating ontology networks have been widely studied within NeOn (see Chap. 2). In our use case, when creating ontologies, we dealt with various different situations:

- Reusing and reengineering non-ontological resources (Scenario 2 of the NeOn Methodology)
- Building ontologies from scratch (corresponding to NeOn Methodology Scenario 1)
- Reusing ontological resources (Scenario 3)
- Reusing ontology design patterns (Scenario 7)
- Mapping the ontologies created in order to obtain a networked set of ontologies (Scenario 5)

We elicited the specifications for single ontologies, mappings among them, and for the entire network, by extensive conversation with domain experts, in some cases summarized by explicit competency questions. We also analyzed the systems and data sets currently in use, to infer requirements about the expressivity of the ontologies, their coverage and use for data collection, storage, and retrieval. In particular, the scope, purpose, and level of formality of the network were derived from those of the existing systems. In this sense, reaching consensus about the ontologies was not really a problem, while several phases of consistency check, verification, and validation of the requirements we extracted were performed together with domain experts.

The ontologies in the network have been built by (1) reengineering non-ontological resources either from relational databases or from informal knowledge organization systems: thesauri, classification schemes, etc.; (2) designing them from scratch; (3) reusing without any change. As for the reengineering of relational databases, we followed the approach proposed in Barrasa et al. (2004), which consists in creating ontologies based on the domain model and then creating mappings between the ontologies and the database. Within this approach, we used ontology design patterns (Presutti and Gangemi 2008; Gangemi and Presutti 2009) whenever possible (many of them were designed specifically for our purpose and then generalized) and the R2O mapping language through the ODEMapster processor[19].

We followed an iterative approach consisting in a phase of domain analysis, a phase of domain modeling, and a phase of data population of those models. A phase of validation made with the collaboration of FAO fisheries and information management experts followed, which triggered a new iteration of modeling and population. Each iteration was carried out in collaboration with domain experts, and sanity checks were performed thereafter.

As for the reengineering of knowledge organization systems, we have considered two thesauri: ASFA and AGROVOC. Both had been previously ported to RDF by using heterogeneous techniques: see Caracciolo et al. (2009) for a description and references to related literature. Therefore, we have used an existing SKOS version of ASFA and an experimental OWL version of AGROVOC, which was under development in an independent FAO project.

Note that the different nature of the resources we used as a starting point, thesauri and reference data (from relational databases), forced different modeling styles. In fact, the ontologies resulting from the reengineering of reference data have a precise semantics: if a class exists in the ontology, its *extensional* interpretation (the set of individuals that have that class as *rdf:type*) includes exemplifications of the domain concept expressed by the name of that class. For example, the *WaterArea* class refers to the collection of things that are water areas according to fishery experts; the Species class refers to the collection of things that are taxonomical species in the knowledge of fishery experts.

On the contrary, thesauri cannot be assumed to have an extensional semantics. For example, *asfa:Catchment_area* cannot be directly interpreted as a class of catchment areas but only as a thesaurus concept; in other words, it has a purely *intensional* semantics. SKOS[20] is a good formal language to encode this type of resources, thanks to its class *skos:Concept* and its properties that enable the representation of purely intensional relations between concepts, such as *broader*, *related*, *exact match*, etc.

In other cases, ontologies have been built from scratch in order to model in a more explicit way concepts that are implicitly used in the other resources that have been reengineered, or because the FSDAS system needed the implementation of some application requirements. Examples include the ontologies about aquatic resources and the ontology for catch records. In those cases, the importance of competency questions (Presutti and Gangemi 2008) was crucial because no previous formal modeling (as ontologies) of the domain existed, and the whole modeling activity was only implicit in domain experts' daily work. Those cases offered an occasion to develop new ontology design patterns and to refine existing ones.

[19] Initially, we used the stand-alone version of ODEMapster and then moved to the plugin version for the NeOn Toolkit.

[20] http://www.w3.org/TR/skos-reference

Finally, the inclusion in the network of the geopolitical ontology is a case of reuse of an ontological resource. We have only reused data about countries, while excluding other data that can be found in there, such as groups of countries.

Ontologies were networked in a variety of ways. First of all, improved versions (either for the model or for the data) of ontologies were connected to one another by means of the attribute *owl:priorVersion*. Second, some of the ontologies produced were also designed in a modular way, thus creating a partial order graph by means of *owl:import* statements. Third, mapping data was taken by the reference data, where it was made available in relational form. For example, this happened with ontologies about different commodity types. Fourth, a limited amount of mapping was manually provided by domain experts. Finally, mappings were learned through automatic methods (supervised by experts) and expressed as OWL ontologies that contain the linking axioms between the vocabulary and the data from the linked ontologies. Examples of these cases include mappings between ASFA and some RTMS-based ontologies (e.g., species, aquatic areas, etc.), the linking between countries in the geopolitical ontology and the catch data, and most of the mappings included in the network.

The mapping between ASFA and AGROVOC (we already highlighted the differences in their intended semantics) is an interesting case of automatic extraction of data and manual mapping. In that case, we had two options: either to enforce an extensional semantics in ontologies derived from thesauri or to leave the intensional semantics of the whole resource, and impose extensional semantics only on specific thesaurus fragments, when needed. As the former approach has proven to imply a time-consuming and partly arbitrary process, we opted for the second approach, both for ASFA and AGROVOC. The consequence of this choice is that, for any further usage of ASFA or AGROVOC concepts within the fishery ontology network, task-oriented decisions will be required to provide a domain semantics. For example, if the concept *asfa:Catchment_area* (an individual from the class *skos:Concept*) is aligned to the class *waterarea:Area* (a class from the FAO_fishing_area ontology), and the expected application aims at, for example, finding the water areas for catch records of tunas, *asfa:Catchment_area* should be also represented as an *owl:Class* by means of a refining rule so that any matching water area (e.g., Mediterranean_sea) extracted from a document indexed by means of ASFA can be represented as an instance of both *asfa:Catchment_area* (as a class) and *waterarea:Area*. OWL2[21] semantics greatly helps in performing such refinements because the interpretation of an ontology entity is made based on its usage context (axiom); therefore, if we declare the axiom:

asfa:Catchment_area owl:equivalentClass waterarea:Area

then *asfa:Catchment_area* is automatically interpreted as an *owl:Class*.

[21] http://www.w3.org/TR/owl2-syntax

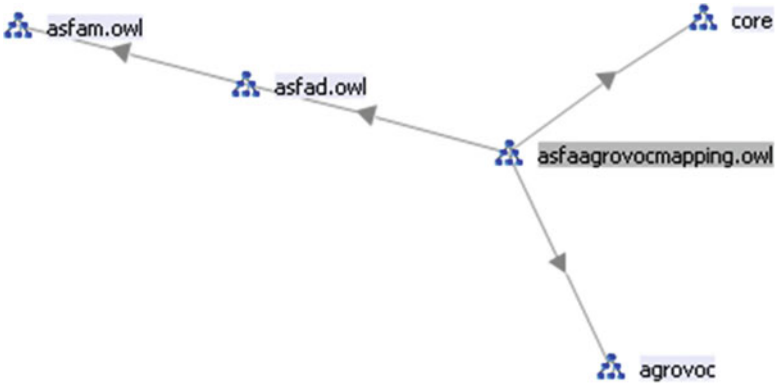


Fig. 18.2 The ontology network resulting from the mapping between ASFA and AGROVOC: asfaagrovocmapping.owl imports SKOS core, AGROVOC, and ASFA (asfad.owl is the actual thesaurus; asfam.owl is the TBox containing the ASFA data model)

The mapping between ASFA and AGROVOC was performed by using string matching techniques[22]. For the mapping vocabulary, the SKOS mapping terms have been used since the only alignment pattern needed is IndividualToIndividual, and the desired mapping semantics is entirely covered by the SKOS terms: exact, broad, narrow, close, related. The matches found in this way were then validated by domain experts, the result of which validation was used to produce the final SKOS mapping document.

Domain experts performing the evaluation of the suggested mappings were presented in a spreadsheet with three columns, each showing an ASFA concept, the candidate AGROVOC concepts (shown using their preferred English label) as a concept-list embedded in a menu box, and the available SKOS mapping relations to choose from.

The results of the ASFA-AGROVOC mapping are finally included in a new ontology that imports SKOS, ASFA, and AGROVOC (Fig. 18.2) and contains all mapping statements that have been validated by the experts, for example:

asfa:Mycotic_diseases skos:closeMatch agrovoc:Mycotoxicoses

[22] We applied matching techniques based on the family of edit-distance functions. We chose the Jaro-Winkler technique (Winkler 1990), which is not properly an edit-distance (as those based on the notion of distance first formulated by Levenshtein (1966)), though it uses a broadly similar metric which has proved of good results in the record-linkage literature. Some pre- and postprocessing of the data was needed in order to normalize the format of entries in AGROVOC and ASFA and present results to domain experts for validation.

18.4 FAO Network of Ontologies

In this section, we describe the FAO network of ontologies we produced and analyze in details its features. All ontologies are available from the FAO website[23] as OWL ontology schemas, populated with RDF instances. An HTML representation of both schema and instances was created by means of OWLDoc plugin for the NeOn Toolkit and made available from the website.

The thematic areas covered by the network are all those introduced in Sect. 18.2 (aquatic species, water areas, land areas, and aquatic resources), plus others (fishing gears and fishery commodities) that we do not describe in this chapter because less central to the notion of fish stock. Figure 18.3 provides a high-level representation of the network of fisheries ontologies. The use of the same icon signifies that the ontologies represented in the picture cover the same thematic areas (see, for example, the wave used to represent ontologies about divisions of water areas). The network includes all the type of data sources mentioned in Sect. 18.2.2 (i.e., reference data, thesauri, and the FAO geopolitical ontology), plus a sample data set of time series about "catch records" about aquatic species. Note that Fig. 18.3 does not explicitly show the ontologies used only for the internal functioning of the FSDAS nor those used to include mappings in the network (see below in this section). Solid arrows denote *owl:imports* statements, while dashed arrows are for mappings between classes and/or instances (see below in this section).

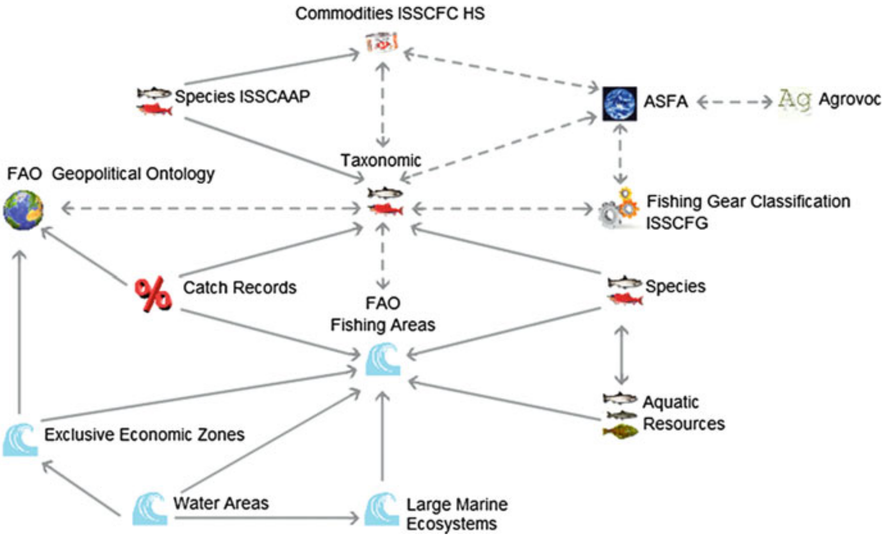


Fig. 18.3 Overview of the network of fisheries ontologies

[23] http://aims.fao.org/website/NeON/sub2

Below, we list the ontologies included in the network, grouped by thematic areas (see Fig. 18.3, ontologies symbolized by the same icon).

1. Aquatic species:
   - Organized according to scientific taxonomic classification. Includes biological entities relevant to fisheries classified taxonomically (Source: reengineering of reference data)
   - Species ISSCAAP. Includes the groups of aquatic species as defined in the ISSCAAP classification (Source: reengineering of reference data)
   - Species. Schema of information about aquatic species[24], including information about species distribution. (Source: manual modeling based on competency questions (Presutti and Gangemi 2008) and the fact sheets schema)
2. Water areas (Source: reengineering of reference data)
   - FAO fishing areas. It includes 27 major areas, divided into a system of subareas, divisions, and subdivisions.
   - Large marine ecosystems. Identified by the National Oceanic and Atmospheric Administration (NOOA) of the USA.
   - Exclusive economic zone.
3. Aquatic resources (Sources: reengineering of reference data, data extracted from fact sheets)
4. FAO geopolitical ontology. Contains information about land areas (countries)
5. Commodities ISSCFC HS (Source: reengineering of reference data). Contains fragments of classification of commodities relative to fisheries: the International Standard Statistical Classification of Fishery Commodities (ISSCFC) and the Harmonized Classification (HS)[25]
6. Fishing gear ISSCFG[26] (Source: reengineering of reference data)
7. AGROVOC thesaurus, covering a variety of thematic areas relevant to fisheries
8. ASFA thesaurus, covering a variety of thematic areas relevant to fisheries
9. Catch records (Source: FAO time series and NAFO)

Most of the ontologies produced are based on single repositories of data, as it is the case with the reference data used to identify the "dimensions" of a piece of statistical data collected by FAO. For example, any data about catch or production is identified by a "what" (which species or group of species), a "where" (which FAO fishing areas), and "by whom" (which country). However, the network also included some ontologies populated with data coming from different sources, as for example, the ontology on stocks, which includes data coming from both reference data and fact sheets.

[24] Any aquatic species relevant to fisheries, including some aquatic birds and plants.

[25] http://193.43.36.238:8181/fi/website/FIRetrieveAction.do?dom=ontology&xml=sectionR.xml

[26] http://www.fao.org/fishery/cwp/handbook/M/en

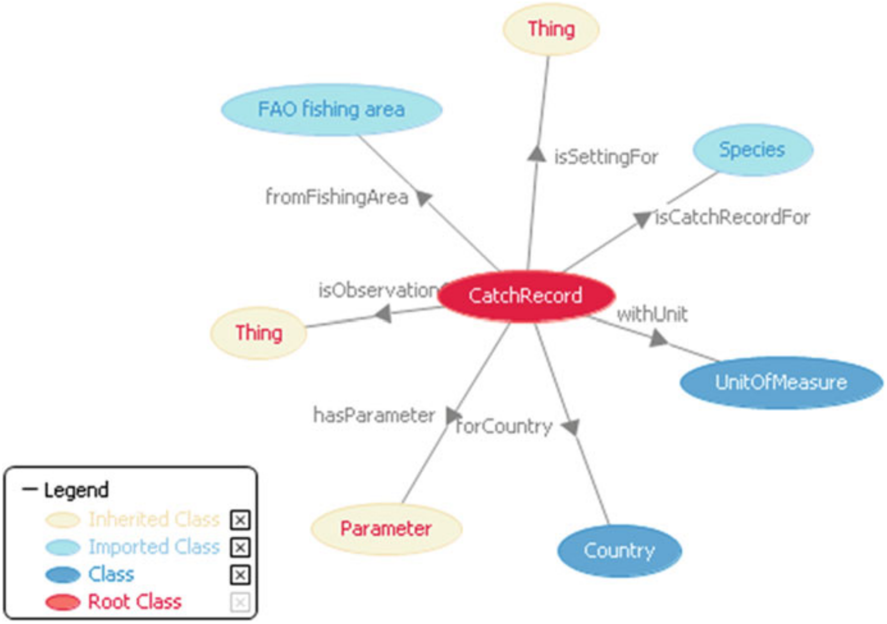


Fig. 18.4 The CatchRecord knowledge pattern shared by FAO and NAFO data (Picture generated by the ontology visualizer of NTK)

The case of the ontology of catch record is an interesting one because it organizes metadata (with extensive linking to the dedicated ontologies of reference data) and data, i.e., pieces of statistical data about catch of aquatic species. We considered the data collected by the Northwest Atlantic Fishery Organization (NAFO)[27] and FAO. Since the level of details of the two data sets is different, it was required that a first step of harmonization be applied in order to reach a common catch record definition. The result of this harmonization step is represented in Fig. 18.4.

The picture shows the schema of a typical catch record, highlighting the types (boxes) of subjects and objects of the properties (arrows) holding between a record and the objects or data involved in that record. The catch record ontology[28] reuses a knowledge pattern (aka content design pattern, Gangemi and Presutti 2009) from the ODP repository that models observations, records, and statements of dynamic facts, with a specific temporal indexing[29], and a knowledge pattern that models spatial relations and places[30]. The class taxonomy of the catch record ontology is

[27] http://www.nafo.int/

[28] http://www.ontologydesignpatterns.org/cp/owl/fsdas/catchrecord.owl

[29] http://www.ontologydesignpatterns.org/cp/owl/observation.owl

[30] http://www.ontologydesignpatterns.org/cp/owl/place.owl

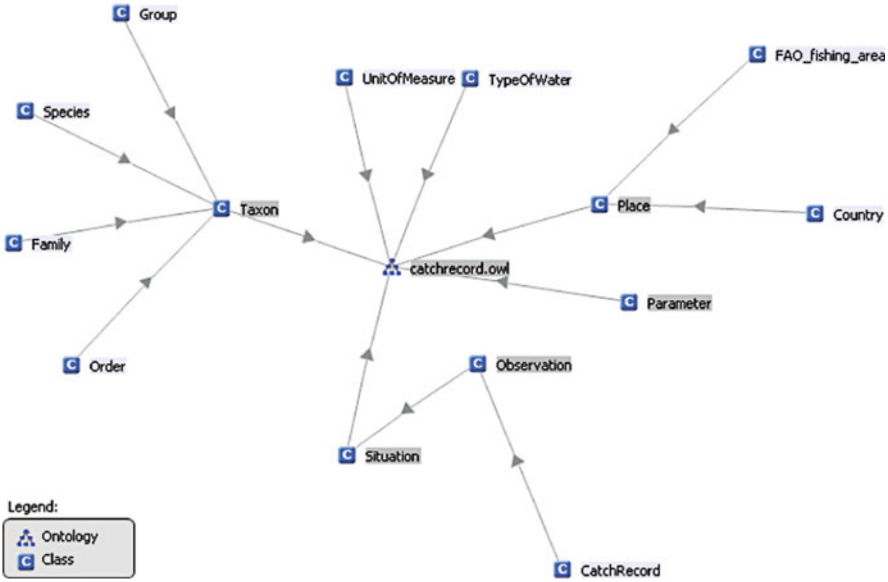


Fig. 18.5 The CatchRecord class taxonomy (Picture generated by the KCViz visualizer of NTK)

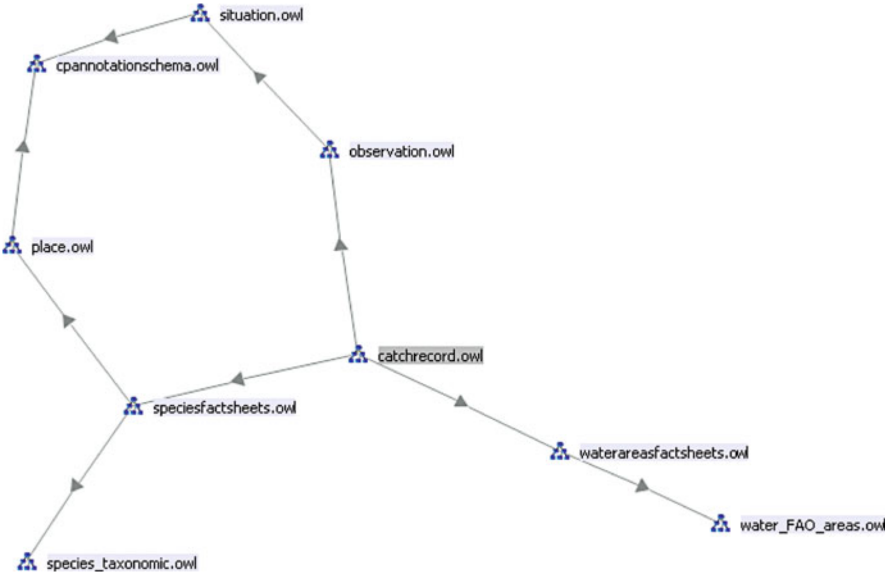


Fig. 18.6 The import graph for the catch record ontology (Picture generated by the ontology import visualizer of NTK)

shown in Fig. 18.5. Figure 18.6 shows the ontology import graph for the catch record ontology (arrows represent *owl:imports* statements). In particular, the *catchRecord* class is a subclass of the class Observation from the *observation.owl* pattern: an observation includes typically an observed entity (in this case, a species through the *isCatchRecordFor* property), some temporal reference (in this case, a date through *recordDate* and a year through *catchYear*), and some parameters (in this case, a fishing area through *fromFishingArea*, a country through *fromCountry*, an amount through *catchAmount*, and a unit of measure through unit). The catch record ontology also reuses the water areas[31] and species[32] ontologies extracted from the FI fact sheets.

The ontologies produced were networked by means of mappings of various natures between their classes and/or individuals. Here we concentrate on those mappings that are exploited by the FSDAS to provide users with richer entry points to the data than those provided by current information systems (as per the limitations mentioned in Sect. 18.1): we do not go in further details with the network of import statements, or prior versions, and concentrate on mappings modeled by using SKOS vocabulary and mappings expressing thematic, domain-oriented information.

In the following, we list the mappings included in the network, including the reason for having them in there. First, we list the mappings modeled as skos: exactMatch:

- Taxonomic – ASFA
  - Matching between species, to provide aquatic species, mainly described taxonomically with more information about common names
- ASFA – AGROVOC
  - Matching between species names, to exploit the multilinguality of AGROVOC for species names
- Fisheries commodities – ASFA
  - To provide ASFA commodities with the exact classifications

Many of these mappings were identified by using the experience (methodologies, tools) gained during the Ontology Alignment Evaluation Initiative (OAEI)[33]. Some of the ontologies included in the network were used as test bed for the OAEI (see Shvaiko et al. 2007; Shvaiko et al. 2008; Shvaiko et al. 2009).

Also, some domain properties were identified and new links created (see Caracciolo et al. 2010). Such links have been learned through typical matching techniques (e.g., Soundex, Levenshtein), linguistic matching (e.g., *headword matching*), and the structural properties emerging from the XML structure of FI

[31] http://www.ontologydesignpatterns.org/cp/owl/fsdas/waterareasfactsheets.owl

[32] http://www.ontologydesignpatterns.org/cp/owl/fsdas/speciesfactsheets.owl

[33] http://oaei.ontologymatching.org/

fact sheets. A manual evaluation of the linking has been performed. The list of domain properties extracted is reported here:

- ASFA – Fishing gear ISSCFC. Relation: species *caught by* gear.
  - To allow explicit grouping of species based on the gears used for their capture
- Taxonomic – Fishing gear ISSCFC. Relation: species *caught by* gear.
  - To allow explicit grouping of species based on the gears used for their capture
- Taxonomic – FAO fishing areas. Relation: species *found in* FAO water areas.
  - To allow explicit grouping of species based on the water areas where they are known to be found
- Taxonomic – FAO geopolitical ontology. Relation: species *in the vicinity of* country.
  - To allow explicit grouping of species based on the countries having a shore with the water areas where the animal is known to be found
- Taxonomic – Commodities ISSCFC. Relation: species *used for* commodity.
  - To provide a biological view on fisheries commodities
- Exclusive economic zones – FAO fishing areas. Relation: EEZ area *intersects* FAO area.
  - To provide a biological view on marine political boundaries
- Exclusive economic zones – FAO geopolitical ontology. Relation: EEZ area *is owned by* a country.
  - To associate water areas and the country ruling over it
- Large marine ecosystems – FAO fishing areas. Relation: LME area *intersect*s FAO area.
  - To have a biological view on FAO reporting areas for statistical purpose
- Species ISSCAAP – Commodities ISSCFC HS. Relation: ISSCAAP group of species *originates* commodity.
  - To provide correspondence between different classifications of aquatic species according to a commercial point of view
- Species ISSCAAP – Taxonomic. Relation: ISSCAAP group of species *includes* taxonomic entity.
  - To provide a biological view on a grouping of species based on commercial interest
- Aquatic resources – Taxonomic – FAO fishing areas. Relations: aquatic resource *consists of* aquatic species; aquatic resource *lives in* FAO area.
  - To express the composition of a stock in terms of species and its presence in a given FAO water area

As for how to expose the mappings between ontologies, i.e., if by means of dedicated ontologies or not, we decided on the basis of what is best suited for provenance and later maintenance. We created third entities in all cases where the links are extracted *after* the creation of the ontologies. While in all other cases, we preferred to leave the linking information inside the ontologies: this happened especially in the case of correspondences between classification systems (e.g., commodities), which have the same provenance as the reference data.

Linguistic information. A dedicated linguistic model has been developed within NeOn, the LIR model (see Chap. 4 in this book). The LIR model is a quite sophisticated framework meant to support creation of ontologies that are multilinguality aware, as is the case, for example, of resources such as AGROVOC, which is currently distributed in 19 languages (5 more to be released soon). However, our network of fisheries ontologies showed a limited degree of multilinguality (in most cases, names are available in no more than three languages), which made us adopt a simplified modeling of linguistic information but compatible with the LIR model.

URIs. We used hash URIs[34] for all ontologies based on reference data, forming their local part by concatenating key codes used in the relational database[35]. This type of URIs guarantees uniqueness; it is easy to check because the meta code allows one to recognize at a glance if a piece of data was correctly taken from the database and organized in the right class; given the stability of the source databases, the generation of new versions is always compatible with previous versions, and it is therefore of easy maintenance. Finally, this convention ensures uniformity throughout the reference tables, which implies that URIs may be built in the same way independently of the reference data set at hand.

The inconvenience of such a type of identifier is that it is very little informative to casual users who are not aware of the database behind. However, this drawback is partially overcome by having *rdfs:labels* that may be used for display, but still one may argue that better, i.e., more informative and user-friendly URIs may be used. Appropriate considerations should be made on a case-by-case approach[36]. The discussion above shows that no uniform approach to URIs can be taken in this

[34] http://www.w3.org/TR/2007/WD-cooluris-20071217/

[35] As in http://www.fao.org/aims/aos/fi/species_taxonomic.owl#ID_31005_2632

[36] Compare, for example, the following cases: For biological entities, the only names available (in FAO data) for all entities are the scientific names. In the case of FAO water areas for statistical reporting, the best option seems to be using the code itself (that vary in length and formal composition, which may represent a problem for maintenance) given to each area, as it is the only piece of information that each item adopts. Other entities in the future should be taken directly from the body in charge, as in case of the large marine ecosystems is maintained by the US National Oceanic and Atmospheric Administration (NOAA). The case of exclusive economic zones (EEZ) is more complex, as there is no single accepted way to model and manage this type of data. GIS technology provides a good tool to keep track of EEZ borders, but for our purposes, it is also important that a coding system, if possible standardized, be available. Similar issues apply to the case of vessel and gear types, while the case of commodities is even more controversial, as in

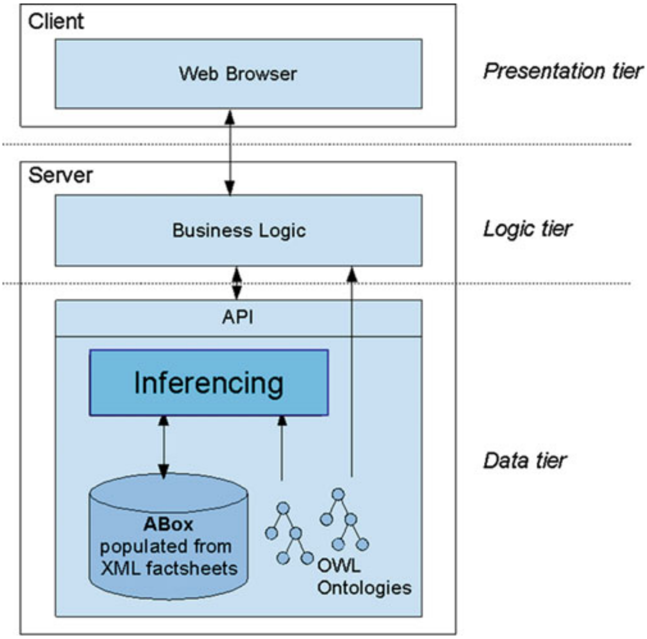


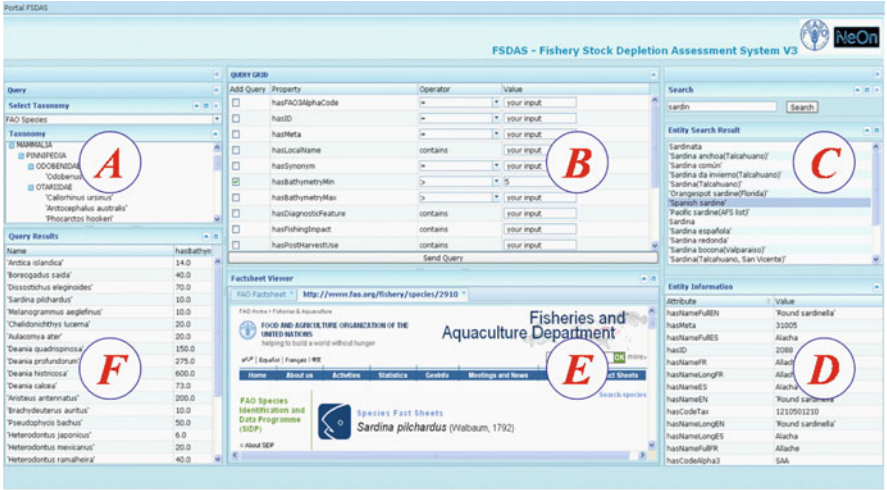
Fig. 18.7 FSDAS system architecture

domain: sometimes, a human-readable name is the best choice; in other cases, names are not available at all, or if they are, they are simply too long and cumbersome to use. Codes may be preferred; however, they follow a number of different formats, and often, they are revised and changed more often than names. For this reason, we kept numeric identifiers in the URIs, and in so doing, we privileged uniformity, uniqueness, and ease of maintenance over the possibility for a human user to grasp from the URI what it is about. However, when data is to be visualized, *rdfs:labels* are used instead of URIs.

18.5 Fisheries Stock Depletion Assessment System

The Fisheries Stock Depletion Assessment System (FSDAS) is a web-based prototype (Fig. 18.7 provides a high-level view of its system architecture) of what could be achieved by a software supported by a network of ontologies. It provides access to a selection of full-text documents and fact sheets, in addition to the data

that domain descriptions are way more important than codes, but unfortunately descriptions tend to be extremely long.


Portal FSDAS
FSDAS - Fishery Stock Depletion Assessment System V3
NeOn
Query
Select Taxonomy
FAO Species
Taxonomy
MAMMALIA
PINNIPEDIA
ODOBENIDAE
'Odobenus
OTARIIDAE
'Callorhinus ursinus'
'Arctocephalus australis'
'Phocarctos hookeri'
Query Results
QUERY GRID
Search
Entity Search Result
Entity Information
Factsheet Viewer
FAO Factsheet
http://www.fao.org/fishery/species/2910
FAO Home > Fisheries & Aquaculture
FOOD AND AGRICULTURE ORGANIZATION OF THE UNITED NATIONS
helping to build a world without hunger
Fisheries and Aquaculture Department
Home
About us
Activities
Statistics
GeoInfo
Meetings and News
Fact Sheets
FAO Species Identification and Data Programme (SIDP)
About SIDP
Species Fact Sheets
Sardina pilchardus (Walbaum, 1792)
A
B
C
D
E
F

Name	hasBathym
'Arctica islandica'	14.0
'Boreogadus saida'	40.0
'Dissostichus eleginoides'	70.0
'Sardina pilchardus'	10.0
'Melanogrammus aeglefinus'	10.0
'Chelidonichthys lucerna'	20.0
'Aulacomya ater'	20.0
'Deania quadrispinosa'	150.0
'Deania profundorum'	275.0
'Deania histricosa'	600.0
'Deania calcea'	73.0
'Aristeus antennatus'	200.0
'Brachydeuterus auritus'	10.0
'Pseudophycis bachus'	50.0
'Heterodontus japonicus'	6.0
'Heterodontus mexicanus'	20.0
'Heterodontus ramalheira'	40.0

Add Query	Property	Operator	Value
□	hasFAO3AlphaCode	=	your input
□	hasID	=	your input
□	hasMeta	=	your input
□	hasLocalName	contains	your input
□	hasSynonym	=	your input
☑	hasBathymetryMin	>	5
□	hasBathymetryMax	>	your input
□	hasDiagnosticFeature	contains	your input
□	hasFishingImpact	contains	your input
□	hasPostHarvestUse	contains	your input

Send Query

sardin Search

Sardinata
'Sardina anchoa(Talcahuano)'
'Sardina común'
'Sardina da invierno(Talcahuano)'
'Sardina(Talcahuano)'
'Orangespot sardine(Florida)'
'Spanish sardine'
'Pacific sardine(AFS list)'
Sardina
'Sardina española'
'Sardina redonda'
'Sardina bocona(Valparaiso)'
'Sardina(Talcahuano, San Vicente)'

Attribute	Value
hasNameFullEN	'Round sardinella'
hasMeta	31005
hasNameFullES	Alacha
hasID	2088
hasNameFR	Allad
hasNameLongFR	Allad
hasNameES	Alacha
hasNameEN	'Round sar
hasCodeTax	1210501210
hasNameLongEN	'Round sardinella'
hasNameLongES	Alacha
hasNameFullFR	Allache
hasCodeAlpha3	SAA

Fig. 18.8 Main front-end panels in the FSDAS web interface

contained in the network, which also includes statistical data and metadata for textual and statistical data.

Functional requirements for the FSDAS were elicited by fisheries managers and domain experts, while non-functional requirements were carefully designed as a joint work with information managers in FAO.

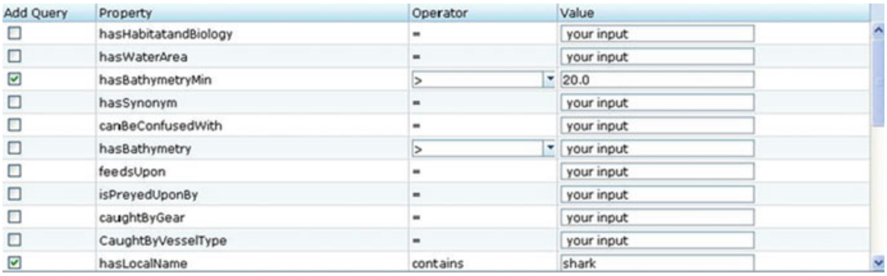
Users experience FSDAS as a browsable and queryable web application that returns organized, ranked, linked results, together with direct links to related stored documents or web pages. Figure 18.8 presents the user interface of FSDAS. It is organized in panels that we list using the letters shown in Fig. 18.8 (A) Taxonomy Panel for browsing taxonomies, (B) Query Composition Panel for the user to compose complex queries, (C) Search and Result Panel for free text search, (D) Resource Detail Panel, (E) Document Viewer Panel to inspect textual documents returned as search result, and (F) Query Result Panel.

The FSDAS also allows for navigation of various classification systems, linked to one another. This allows one to use a given classification system even if different from the one used for data storage. Also alternative names retrieved from networked resources are displayed to users. In summary, the large number of mappings available is used to allow users to search the data set according to multiple perspectives, for example, according marine area or gear type, as well as aquatic species.

The query panel (Fig. 18.9) allows users to formulate queries like as complex as:

“What are the aquatic species living below 20 m depth, and whose name contains the word-part shark?” In traditional systems, the answer to such a query could only be found after a sequence of queries to different resources.

FSDAS also supports the grouping of aquatic resources by (marine) area or other criteria, such as the fishing gear typically used for their capture.



Add Query	Property	Operator	Value
□	hasHabitatandBiology	=	your input
□	hasWaterArea	=	your input
☑	hasBathymetryMin	>	20.0
□	hasSynonym	=	your input
□	canBeConfusedWith	=	your input
□	hasBathymetry	>	your input
□	feedsUpon	=	your input
□	isPreyedUponBy	=	your input
□	caughtByGear	=	your input
□	CaughtByVesselType	=	your input
☑	hasLocalName	contains	shark

Fig. 18.9 FSDAS user interface: the query panel

18.6 Conclusions and Lessons Learned

In this chapter, we reported on our experience with the application of tools and techniques developed within the NeOn project to the area of fisheries. We have developed a network of ontologies covering the most important thematic areas needed to manage information about aquatic resources (i.e., aquatic species, water areas, and land areas) and massively included mappings between the ontologies in the network. The network also included a data set of statistical data: this is the first attempt to give a semantic modeling to numeric data of that sort. Based on the network, we built a prototypical application, called Fisheries Stock Depletion Assessment System, with the goal to show users how networked ontologies could be used.

The FSDAS prototype nicely shows the networked ontologies developed for this work and how they can be used to bridge the information gap currently existing between data sets. In particular, by using as a rich network of ontologies, it allows users to query and look at the data collected, in particular statistical data, according to more points of view than the one used for data storage.

The creation of the network was nicely supported by the NeOn Toolkit and its plugins, in particular for the reengineering of non-ontological data (i.e., the reference data for time series) and the ontology mapping. However, in the case of ontology mapping, also a number of NeOn technologies have been used, that were later developed as NeOn plugins. For the reengineering of reference data, we used extensively the NeOn plugin ODEMapster, which relies on ontologies for the extraction of data from relational database. This approach resulted to be very useful in preliminary phases but seems to be less convenient for data maintenance, as any minor change in the original database requires that a new transformation phase takes place.

The network of fisheries ontologies was produced within the scope of a large prototypical effort, and it was not used to replace current information systems and data sets. The implication of this state of affairs is that we essentially duplicated data, as data maintenance continued to take place in the original data sources which continued to feed existing information systems. In order for a complete replacement to take place, all information systems accessing the data included in the network of ontologies should have been reengineered so as to work with ontologies and RDF data. In fact, this is a process that is certainly going to happen but very likely in an incremental manner. Based on this and other considerations, we encourage the use of ontologies (OWL schema plus RDF instances) primarily for data sharing and dissemination. For example, the extraction of data from relational database and its reengineering as ontologies was extremely useful to make evident the many ad-hoc decisions made when storing data for internal use only. These lessons revealed to be important especially when dealing with standard classifications, such as the various coding systems we considered. As many of the coding systems considered are actually maintained by FAO and adopted as standards by the international community, the use of standard technologies (e.g., RDF, OWL) showed to be an appropriate approach to the publication of those standards to the public.

The work exposed in this chapter also contributed to a better understanding of the issue of converting traditional thesauri into modern formats to be compatible with current approach to web publications. For instance, the conversion of the AGROVOC thesaurus into a concept schema is now finalized, while the conversion of the ASFA thesaurus is about to be completed, also thank to the lessons learned during the making of this work (see considerations about AGROVOC and ASFA semantics in Sect. 18.3).

The activity of ontology mapping has had a remarkable follow up in the area of linked data, which has RDF and concept schemas (ontologies) at its very foundations. One of these results is that FAO is currently enhancing the network of ontologies and the conversion work carried out within NeOn to produce a larger data repository of open linked data (Caracciolo et al. 2011). One example of this is the linked data version of AGROVOC[37].

Acknowledgments We are pleased to thank our colleagues in FAO who have collaborated in various phases of the NeOn project: Marta Iglesias, Yves Jaques, Margherita Sini, Aureliano Gentile, Francesco Calderini, Soonho Kim, Fabrizio Sibeni.

References

Barrasa J, Corcho O, Gómez-Pérez A (2004) R2O, an extensible and semantically based database-to-ontology mapping language. In: Second Workshop on Semantic Web and Databases (SWDB2004), Toronto, Canada

Caracciolo C, Sini M (2007) Requirements for the treatment of multilinguality in ontologies within FAO. In: Proceedings of OWLED2007. Available at http://owled2007.iut-velizy.uvsq.fr/PapersPDF/submission_45.pdf

Caracciolo C, Heguiabehere J, Gangemi A, Presutti V (2009) NeOn deliverable D7.2.3. Initial network of fisheries ontologies. NeOn project

[37] http://aims.fao.org/website/Linked-Open-Data/sub

Caracciolo C, Heguiabehere J, Gangemi A, Peters W, Stellato A (2010) NeOn deliverable D7.2.4. Second network of fisheries ontologies. NeOn project

Caracciolo C, Morshed A, Stellato A, Johannsen G, Keizer J (2011) Thesaurus maintenance, alignment and publication as Linked Data. In: Proceedings of MTSR 2011

Gangemi A, Presutti V (2009) Ontology design patterns. In: Staab S et al (eds) Handbook of ontologies, 2nd edn. Springer, Berlin

Iglesias Sucasas M, Caracciolo C, Baldassarre C, Jaques Y (2007) NeOn deliverable D7.1.2. Revised specifications of user requirements for the Fisheries case study. NeOn project

Kim S, Iglesias Sucasa M, Caracciolo C, Viollier V, Keizer J (2009) Integrating country-based heterogeneous data at the United Nations: FAO's Geopolitical Ontology and services. In: Proceedings of semantic technology conference, 2009. http://semanticweb.com/integrating-country-based-heterogeneous-data-at-the-united-nations-fao-s-geopolitical-ontology-and-services_b10681

Levenshtein V (1966) Binary codes capable of correcting deletions, insertions, and reversals. Soviet Phys Doklady 10:707–710

Presutti V, Gangemi A (2008) Content ontology design patterns as practical building blocks for web ontologies. In: Spaccapietra S et al (eds) Proceedings of ER2008, Barcelona, Spain

Shvaiko P, Euzenat E, Giunchiglia F, He B (2007) Proceedings of 7th ontology matching workshop. http://oaei.ontologymatching.org/doc/Proceedings-OM-2007.pdf

Shvaiko P, Euzenat E, Giunchiglia F, Stuckenschmidt H (2008) Proceedings of 8th ontology matching workshop. http://disi.unitn.it/~p2p/OM-2008//om2008_proceedings.pdf

Shvaiko P, Euzenat E, Giunchiglia F, Stuckenschmidt H, Noy N, Rosenthal A (2009) Proceedings of 9th ontology matching workshop. http://disi.unitn.it/~p2p/OM-2009//om2009_proceedings.pdf

Winkler WE (1990) String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In: Proceedings of the section on survey research methods, American Statistical Association, Alexandria, pp 354–359

Chapter 19
Electronic Invoice Management in the Pharmaceutical Sector: The PharmaInnova Case

José Manuel Gómez-Pérez, Víctor Méndez, Joan Candini, and Juan Carlos Muñoz

Abstract Since the use of electronic invoicing in business transactions was approved by the EU back in 2002, its application in Europe has grown considerably. However, despite the existence of standards like EDIFACT (http://www.unece.org/trade/untdid/welcome.htm) or UBL, (http://www.oasis-open.org/committees/ubl) widespread take-up of electronic invoicing has been hindered by the enormous heterogeneity of proprietary solutions. In this chapter, we describe an approach toward addressing the interoperability problem in electronic invoice exchange. We especially focus on networked ontologies as the main enablers of such an approach, where networked ontologies serve as a semantic gateway for the transformation of invoice data between different formats and models.

19.1 Introduction

Since the new EU legislation on electronic invoicing was approved in 2002 and implemented by member countries in 2004, it is possible to send and receive invoices electronically, provided that they include a digital signature. The potential savings are enormous assuming that appropriate technological solutions exist. In parts of Europe, this has led to a proliferation of proprietary software products by vendors, e.g., SAP and ORACLE, aimed at tackling the e-invoicing problem.

J.M. Gómez-Pérez (✉) • V. Méndez
Intelligent Software Components (iSOCO) S.A., Avda. del Partenón, 16-18, 28042 Madrid, Spain
e-mail: jmgomez@isoco.com; vmendez@isoco.com

J. Candini
Laboratorios KIN, S.A, C. Ciudad de Granada, 123, 08018 Barcelona, Spain
e-mail: jcandini@kin.es

J.C. Muñoz
PharmaInnova, Avda. Torreblanca, 57, 08172 Sant Cugat del Vallés, Spain
e-mail: jcmunoz@pharmainnova.com

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*, DOI 10.1007/978-3-642-24794-1_19, © Springer-Verlag Berlin Heidelberg 2012

However, almost all current solutions are stand-alone applications, each with their own model of an electronic invoice.

Consequently, many industries suffer from migrating legacy systems to the formats required by the current e-invoicing solutions. This is obviously an entry barrier, especially for small and medium enterprises; large companies suffer less because they can "force" their providers to comply with a particular format, or else, they are out of business. Taking into account that a middle-sized organization processes around 100,000 invoices per year, the potential benefits are self-evident, with an estimated saving of almost 80% with respect to traditional paper invoices (Gómez-Pérez et al. 2006). However, the risks are also considerable, given the size of the investment required on one hand and the consequent vendor lock-in on the other hand. It is obvious that technologies with the potential to reduce the cost of migrating from one format to another are extremely attractive, especially for middle-sized companies.

Throughout industry, there is a large duplication of effort that could be significantly reduced if companies in the same sectors were willing to share models and infrastructure, a precondition which is made more complex by the competitive environments where they operate. The main limitations therefore include:

1. High investment (acquisition and maintenance) for industrial stakeholders to set up their own business IT infrastructure.
2. Difficulty in setting up business partnerships is due to high IT integration costs; this requires integration and communication across heterogeneous infrastructures, with additional investments to be made as each new partner joins such partnership. In practice, this implies the development of ad hoc transformation software between each pair of invoice formats and models potentially participating in economic transactions, which is time-consuming, expensive, and cumbersome.
3. Lack of possibilities to benefit from the fact that business partners of companies in a same sector are often shared and their invoice models could be common.

This scenario provides an opportunity for building semantic platforms for data interoperability among different stakeholders for business transactions in the form of invoice exchange. Our main objective is to facilitate interoperable invoice exchange between organizations following different formats and models, thus reducing entry barriers for companies and stimulating widespread adoption of e-invoicing, especially among small- and middle-sized companies.

In order to achieve such objective, users of e-invoicing systems need to be provided with (1) expressive, modular, and extensible means to represent invoice data observing both relevant standards and proprietary, local, and even tacit (not explicitly defined by a data model) representations of invoices and (2) usable tools that enable users to define correspondences between their invoice data and such formal models, which can be subsequently exploited for automated exchange of invoices between stakeholders following different representation models and/or formats.

In this chapter, we focus on the principled development of networked ontologies to address the first of the abovementioned issues, allowing for automatic transformation of large amounts of invoice data across formats and models.

19.2 Use Case Description

The main problem of interoperable invoice exchange is heterogeneity. Current technology and methods do not allow building generic solutions which allow the different peers involved in a commercial transaction to automatically process any type of invoice. The range of Enterprise Resource Planning (ERP) systems according to which invoices are emitted and the different formats that exist in the market are so wide that it has been necessary for organizations to take special measures in order to adopt electronic.

Nowadays, two main types of measures are applied: (1) to create clusters or sectorial associations that agree to define common invoicing infrastructure in terms of shared ERP platforms, invoice formats and models, and processes and (2) to identify the invoicing infrastructure that is necessary to automate invoice exchange with given stakeholders and build specific ad hoc plugins which implement gateways between the invoicing infrastructure of each peer.

Additionally, these two options can be mixed to a certain extent, as in the case of PharmaInnova[1], where a group of middle-sized laboratories have agreed to define common invoice models but keeping their ERP infrastructure. Such approach (illustrated in Fig. 19.1) requires an invoice interoperability middleware that ensures compatibility between invoice models of providers, wholesalers, and laboratories. Most laboratories have the same clients, so it is easier for them, as members of the cluster, to define, negotiate with clients, and implement common invoice models. This also allows the members of the cluster to share the expenses of building the abovementioned middleware.

However, all this process is costly in time and effort, and flexibility is low. For example, if a new member enters the cluster, the agreed invoice model needs to be revised not only within the cluster but also with the clients. This problem also applies to all the possible commercial transactions of pharmaceutical laboratories with suppliers, wholesalers, and pharmacies and, by extension, to those between wholesalers and pharmacies. Furthermore, knowledge about how to process invoice data in order to address the interoperability problem should be provided as directly as possible by subject matter experts (SMEs) in the field, leveraging their expertise on the domain to conduct this process. This allows to minimize the number of errors introduced by engineers with a limited knowledge of the e-invoicing models and

[1] http://www.pharmainnova.com

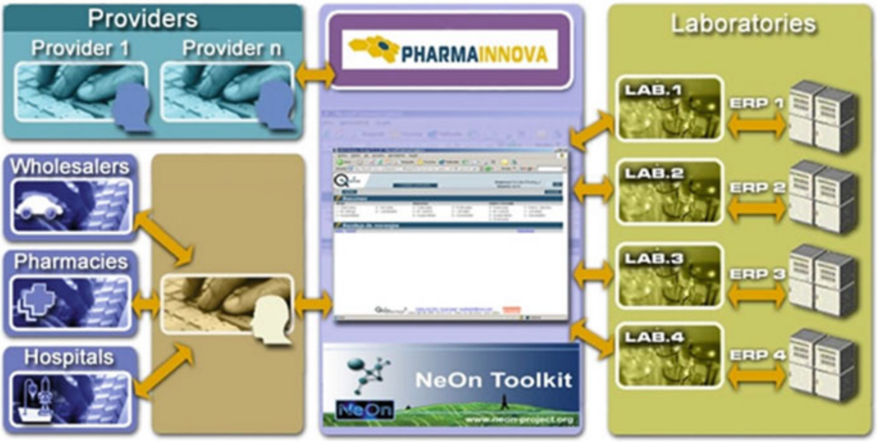


Fig. 19.1 Interoperable invoice exchange in PharmaInnova

formats of specific companies and to reduce the cost of implementing the transformation process by leaving additional engineers out of the loop.

However, the knowledge of SMEs about invoice representations is usually constrained to their own e-invoicing systems. Therefore, it becomes necessary to provide them with a shared, formal conceptualization, e.g., an ontological representation of the e-invoicing domain, which SMEs can use to describe their invoice data. The election of an ontological framework for such purpose supports a threefold objective:

1. Provide a formal model for the representation of knowledge related with the e-invoicing domain, which observes both e-invoicing standards and sectorial specializations
2. Serve as a semantic gateway for invoice transformation during invoice exchange
3. Ensure the consistency of invoices exchanged between heterogeneous systems by leveraging the expressivity of the ontologies for automatic data and object type checks, observations of cardinality constraints, etc.

In this use case, we follow a learn-by-example approach where SMEs define correspondences between a sample invoice and the ontologies, which enable them to semantically annotate invoice data. The correspondences defined through such annotations are stored, recording metadata about each individual piece of invoice data annotated by the SME and the ontology entity it corresponds to. Subsequent invoices following the format and model of the sample can be automatically processed using the correspondences identified during the annotation phase, thus supporting their transformation into ontology entities and, from there, into whatever other invoice format and model treated in the same way.

19.3 Ontology Network Development

In the context of this use case, we have built an ontological framework in the form of expressive, modular, and extensible networked ontologies following the ontology development guidelines provided by the NeOn Methodology (Chap. 2) and, simultaneously, contributing to its development. The resulting invoicing networked ontologies are available as exemplary ontologies at the Ontology Design Patterns[2] portal. Next, we describe the application of the methodology to this particular case, focusing on:

- The observed ontology requirements specification
- The established ontology development life cycle, as part of the scheduling activity
- The most relevant processes and activities that have been performed for the development of the ontologies

19.3.1 Ontology Requirements Specification

Ontology requirements have been obtained fundamentally through competency questions, answered by SMEs in a number of business sectors, who covered a broad spectrum of the different roles in the invoicing process, including:

- U1. User of the invoicing application who is going to model a new invoice
- U2. User who emits invoices
- U3. User who receives invoices
- U4. User who administrates the invoicing system
- U5. Developers of invoicing applications

The complete set of competency questions and answers obtained can be found in Gómez-Pérez et al. (2007). Such competency questions can be classified as follows:

1. Competency questions about the e-invoicing workflow (11 CQs). Examples for this group are:

   CQ7: What is necessary to identify the emitter of the invoice? NIF/CIF.

   CQ9: What is necessary to identify the products in the invoice? Product description.

   CQ15: What is the address of the emitter of the invoice? The supplier fiscal address.

   CQ17: What is the status of the invoice X? The status can be imported, emitted, in process, accepted, in creation or disused.

[2] http://ontologydesignpatterns.org/wiki/Ontology:Main

2. Business rules applied during invoice exchange (4 CQs). Examples for this group are:

   CQ20: What is the total discount applied to this invoice? Discounts in payment date.
   CQ21: Is it possible to apply any special price to this invoice? Yes, if you describe the concept in the description line.
   CQ22: Is it possible to apply any business rule in this invoice? No, only the rules related to the amount to pay and supplier code.
   CQ23: What is the unitary price before applying discounts? The net price.

3. Information about the roles of e-invoice emitters and receivers (32 CQs). Examples for this group are:

   CQ29: How much is the total price of the invoice? Price in the specific invoice received.
   CQ33: When do we have to pay? Date in the specific invoice received.
   CQ49: Have we sold any other product in invoice X? Products in the specific invoice received.
   CQ60: Do we have to apply any specific rule in invoice X? Rules in the specific invoice received.

4. State-of-the-art technologies used in e-invoicing systems (5 CQs). Examples for this group are:

   CQ40: What invoicing technologies are using the emitters of the invoice? ERPs and small products for invoicing like Facturaplus or Contaplus, and customized applications.
   CQ42: In percentage, can you classify the invoicing technologies of each emitter? 65% CSV and FLF, 25% xml, and 10% EDI.
   CQ44: Is possible to classify the technologies depending on the business type of the emitter of the invoice? No.

5. Multilingualism needs (2 CQs). The CQs for this group are:

   CQ18: What is the language of the invoice? Currently only Spanish.
   CQ19: Where is the emitter of the invoice from? Spain.

6. Time modeling (15 CQs). Examples for this group are:

   CQ62: When do we have to pay the invoice X? Date that depends on the agreement.
   CQ63: When are the goods arriving bought in invoice X? On the agreed date.
   CQ67: What is the expiry date of invoice X? There is no expiry date.
   CQ75: How many products did we buy during the month? Number of products in the received invoices.

7. Currency representation (8 CQs):

   CQ76: In what currency are the receivers paying in invoice X? Euro
   CQ78: What taxes are applied in the invoice X? IVA, IGIC, or RE.
   CQ82: How much is the total amount in invoice X? Total line amount – total discounts + taxable amount.

8. General and composed competency questions (14 CQs):

   CQ4: What concepts are mandatory for a wholesaler/provider/laboratory? Two types of information: first regarding the identification of companies (names, addresses, bank accounts, etc.) and second information about the amounts of the products in the invoice and their prices
   CQ84: Given a set of invoices of different companies, is it possible to identify the common concepts used? Yes.
   CQ87: Given the information of a company, what products did it buy? Products in the received invoice.
   CQ91: Given the information of a product, how many units have been sold? Number of units in the received invoice.

Among the most relevant findings, interviews with SMEs showed the need of observing the main e-invoicing standards, which were identified as EDIFACT and UBL, and proprietary data models in business partnerships.

19.3.2 Ontology Development Life Cycle and Scenarios for Building the Invoicing Networked Ontologies

In the face of eventual changes in the requirements to fulfill by the invoicing networked ontologies, we have applied an iterative-incremental life cycle model (explained in Chap. 2) for the development of the invoicing networked ontologies. In addition to the activities performed as part of Scenario 1 (e.g., the ontology conceptualization activity), we have mainly combined Scenario 6 of the NeOn Methodology (reusing, merging, and reengineering ontological resources) and Scenario 2 (reusing and reengineering non-ontological resources) from those described in Chap. 2. The *ontology specialization* and *ontology localization* activities, respectively, from Scenario 8 and Scenario 9 have also been performed. See Fig. 19.2 for a detailed graphical representation.

19.3.3 Processes and Activities Performed

The following activities have been carried out for building the invoicing networked ontologies:

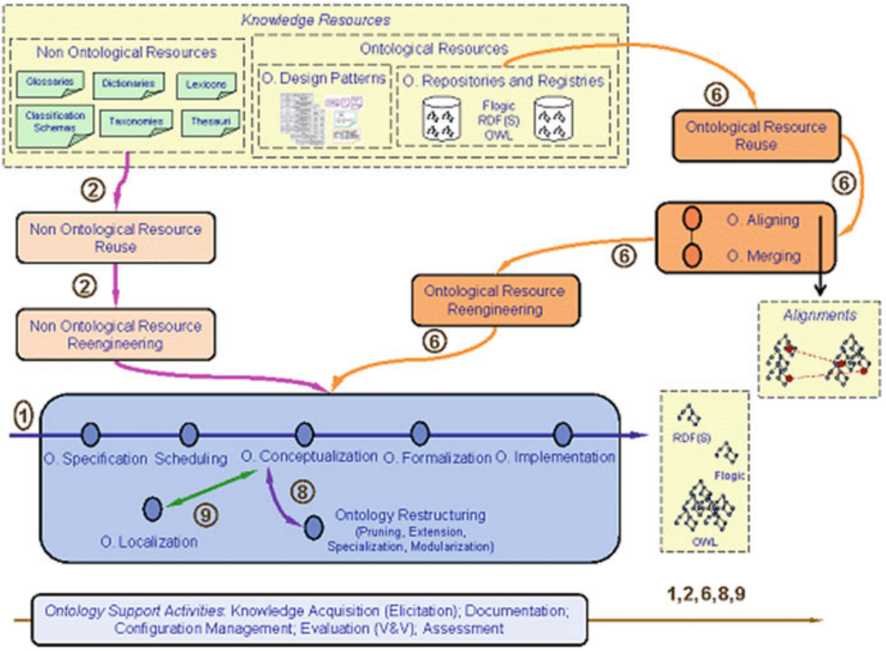

Knowledge Resources
Non Ontological Resources
Glossaries
Dictionaries
Lexicons
Classification Schemas
Taxonomies
Thesauri
Ontological Resources
O. Design Patterns
O. Repositories and Registries
Flogic
RDF(S)
OWL
⑥
Ontological Resource Reuse
⑥
②
Non Ontological Resource Reuse
O. Aligning
O. Merging
⑥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

Fig. 19.2 NeOn Methodology (focus on scenarios 2, 6, 8, and 9)

1. *Ontology elicitation*. In this support activity, the pharmaceutical domain was analyzed, with a focus on the invoicing life cycle, describing the steps an invoice goes through from the time it is emitted to the moment it is validated by the receiving company. This analysis also includes the actors that participate in the process (laboratories, wholesalers, and providers), and their requirements.
2. *Ontology requirements specification*. As described in Sect. 19.3.1, this activity is aimed at addressing the requirements that need to be fulfilled by the ontologies in order to effectively support applications implementing the approach described herein.
3. *Reuse of existing knowledge resources*. The knowledge resources used for creating the invoicing networked ontologies can be organized in the following groups:

   – *Upper-level ontologies and related projects*. The motivation for using upper-level ontologies comes from the need of reuse of the main reference ontology for invoicing. The purpose of this ontology is that it can be instantiated for different sectors of the industry. The first instantiation is for the pharmaceutical sector, laboratories mainly, but it will also be extended for providers of these laboratories or wholesalers. These providers provide from chemical products to energy or clean products, so they need different instantiations of the invoice reference ontology.

- *Invoicing resources*. These resources are mainly technologies for electronic invoicing. The technologies are UBL, EDIFACT, and the PharmaInnova approach.
- *Projects whose main goal is to integrate the invoice vocabulary into ontologies*. These include the ONTOLOG project[3] and the XBRL (eXtensible Business Reporting Language) Ontology project[4].

4. *Ontology conceptualization* (development of the invoicing networked ontologies). In this step, we conceptualized the resources analyzed in the previous activities. A result of this activity is the ontology design pattern *Invoice*[5], used to represent the core aspects of electronic invoices. Similarly to other ontology design patterns, this one is a conceptual model which encapsulates the knowledge representation given by SMEs, throughout an invoice template to ease the alignment of other heterogeneous models for invoices grouping best practices and hence helping to avoid mistakes in this step.
5. *Ontology specialization* (adaptation of the invoicing networked ontologies). The final invoice reference ontology was adapted to the cluster of companies that are going to use it, a laboratory for instance. The invoice reference ontology will be specialized to each cluster of companies needs (laboratories in an initial phase).
6. *Ontology localization* (localization of the invoicing networked ontologies). The users of the networked ontologies belong to different regions in Spain, in which different languages are used. Spanish is the official language, but in these regions, there are other co-official languages; therefore, localization was taken into account. Likewise, this activity was followed to anticipate future use of the ontology out of Spain.
7. *Ontology evaluation* (evaluation of the invoicing networked ontologies). Following this support activity, the invoicing networked ontologies were evaluated by the users of PharmaInnova as described in Candini et al. (2010).

19.4 Description of the Invoicing Networked Ontologies

As shown in Fig. 19.3, the invoicing networked ontologies[6] comprise a number of ontologies including the invoicing backbone ontology (IBO) and other ontologies for the subdomains addressed (one ontology module each). Such ontologies are the UBL Invoicing Ontology (UBLIO) for the UBL e-business standard, the EDIFACT Invoice Message Ontology (EIMO), which represents the subset of the EDIFACT

[3] http://ontolog.cim3.net/

[4] http://xbrlontology.com/

[5] http://ontologydesignpatterns.org/wiki/Submissions:Invoice

[6] Available at: http://ontologydesignpatterns.org/wiki/Ontology:Aggregated_Invoice_Ontology and http://www.isoco.com/ontologies/neon/AggregatedInvoiceOntology.owl

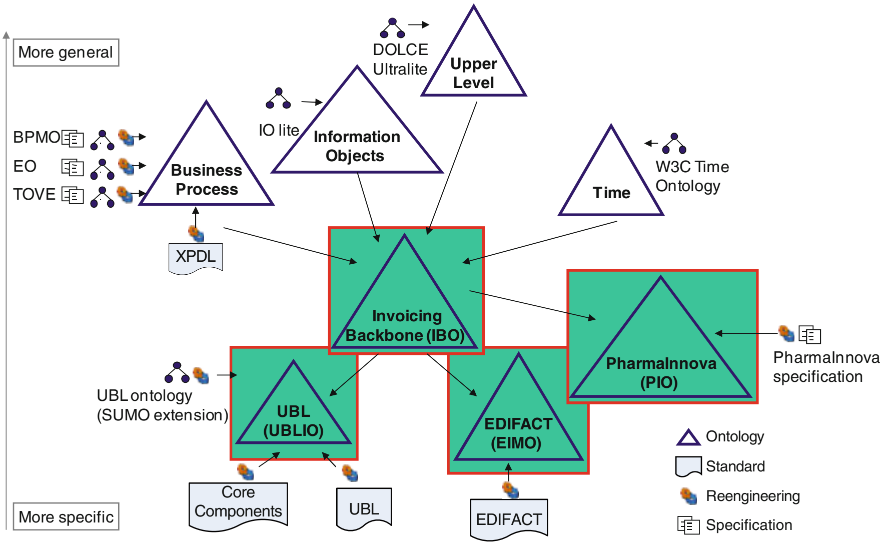


Fig. 19.3 The e-invoicing networked ontologies

Table 19.1 Metrics of the e-invoicing networked ontologies

Classes	697
Object properties	532
Data properties	295
DL expressivity	SHIQ(D)
Class axioms	1,922
Object property axioms	1,428
Data property axioms	643
Annotation axioms	3,224

standard describing the EDI messages used for electronic invoice exchange, and the PharmaInnova Ontology (PIO), which provides a formal representation of the invoicing model used in the PharmaInnova partnership for electronic invoicing in the pharmaceutical sector.

The coverage of the e-invoicing domain provided by these ontologies is extensive, with almost 700 classes, around 500 object properties, and 300 data properties (Table 19.1). Additionally, its design allows further extensions through modules for new standards or proprietary approaches.

IBO has been built through the reuse of a number of business process ontologies, like the enterprise ontology (EO) (Uschold et al. 1998) and TOVE[7], time

[7] http://www.eil.utoronto.ca/Enterprise-modelling/tove

ontologies, the W3C time and time zone ontologies, and upper-level ontologies (DOLCE ultralite and the information objects ontology[8]). While UBLIO is an extension of the UBL ontology, based on SUMO[9] and the core components[10] recommended by the standard, EIMO is the ontological version of the EDIFACT recommendation for electronic invoicing. Finally, PIO has been produced by reengineering the PharmaInnova XML schema describing their invoicing data model into an actual ontology. A detailed description of the ontologies can be found in Gómez-Pérez et al. (2007).

During ontology reuse and conceptualization, we have used the core functionalities provided by the NeOn Toolkit[11] together with those stemming from a number of plugins[12], fundamentally: the OWL modeling plugin, the alignment plugin for ontology alignment, RaDON for ontology repair, ontology relationship visualizer for ontology visualization and navigation, and CupBoard, the online ontology repository for sharing and reusing ontologies linked together and their alignments.

19.5 Application Description: i2Ont

The range of ERP systems managing invoicing information (SAP, ORACLE, PeopleSoft, Baan, Movex, openXpertya, etc.) and the different languages for exchange of electronic business documents that exist in the market (EDIFACT, UBL, Intermediate Document from SAP, etc.) are extremely diverse. NeOn Toolkit with i2Ont plugin[13] (Fig. 19.4) applies the invoicing networked ontologies to enable organizations involved in economic transactions to exchange arbitrary electronic business documents by automatically extracting the information contained in them out of the details of their particular representation formats and technologies, thus saving large amounts of money in the process, as shown in Candini et al. (2010). Figure 19.4 shows NeOn Tookit with i2Ont plugin, where its components are magnified: (1) relationship visualization provides a visualization paradigm to navigate based on ontologies relations, (2) attributes view shows information concerning data properties for the selected concept and information about relationships items of invoice and data properties, and (3) invoice view shows information concerning the loaded invoice and items of invoice which are mapped.

[8] http://www.loa-cnr.it/DOLCE.html

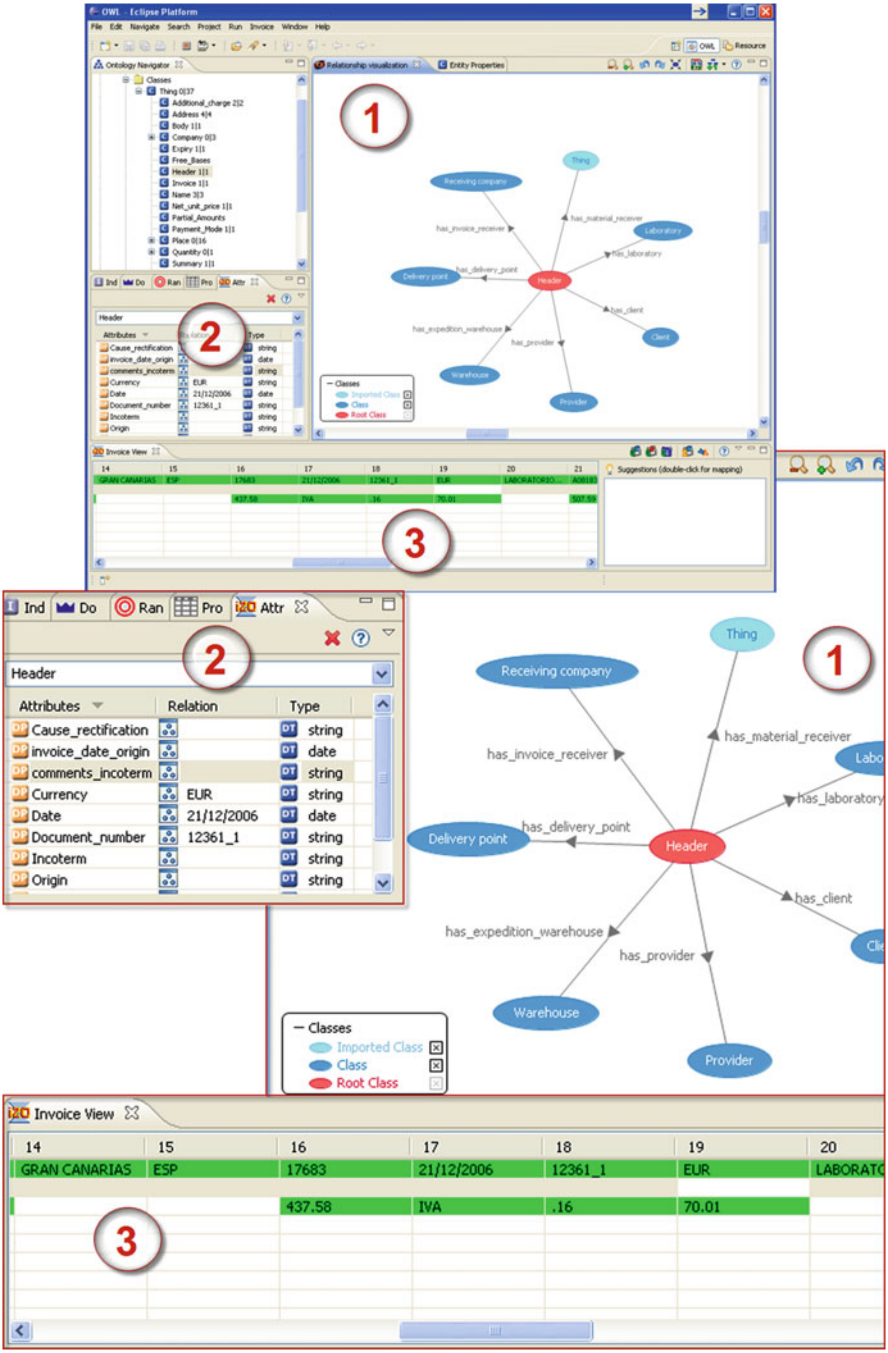
[9] http://www.ontologyportal.org

[10] http://ontolog.cim3.net/cgi-bin/wiki.pl?CctRepresentation

[11] http://neon-toolkit.org

[12] http://neon-toolkit.org/wiki/Neon_Plugins

[13] http://www.neon-project.org/nw/Movie:_i2Ont



Attributes	Relation	Type
Cause_rectification		string
invoice_date_origin		date
comments_incoterm		string
Currency	EUR	string
Date	21/12/2006	date
Document_number	12361_1	string
Incoterm		string
Origin		string

14	15	16	17	18	19	20	21
GRAN CANARIAS	ESP	17683	21/12/2006	12361_1	EUR	LABORATORIO...	A08183
		437.58	IVA	.16	70.01		507.59

Fig. 19.4 NeOn Toolkit with i2Ont plugin (views with numbers are magnified)

One of the most challenging entry barriers for uptake by real users in the domain, with no background on ontological engineering, is the gap between domain knowledge (e-business and economic transactions in the pharmaceutical domain) and the formalisms used to acquire and represent such knowledge. Inspired by Newell's definition of the knowledge level (Newell 1982) back in the eighties, we have intended to develop a highly usable, intelligent user interface that enables experts on e-business and financial staff to alleviate their invoice interoperability problems by means of networked ontologies, relieving them from caring about the way invoice knowledge is formally represented, stored, mapped and, in summary, processed. i2Ont allows domain experts to work and think exclusively at the level of their expertise, i.e., electronic invoices.

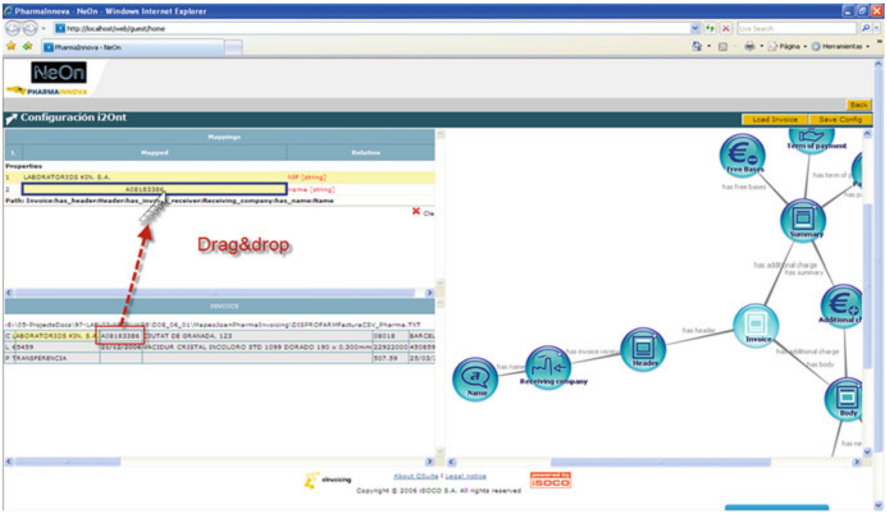
The solution proposed is grounded on a combination of networked ontologies and a graph-based visualization and navigation paradigm. Networked ontologies provide a formal, semantic backbone between different electronic invoicing formalisms and models, including support for the main invoicing standards, like EDIFACT and UBL, and sectorial approaches like PharmaInnova. The user interface allows for a simple navigation across the relevant invoicing concepts, and the formal invoice model described in the ontology network allows ensuring correctness and completeness of the correspondence between the different electronic invoice representations.

Previous approaches to the invoice interoperability problem required implementing specific transformations between the formats and models of each pair of organization exchanging electronic invoices. This was cumbersome and little scalable. On the contrary, i2Ont learns by example, i.e., sample electronic invoices are used to define the mappings between electronic invoice data and ontology concepts. Subsequent electronic invoices received by the system, with a format and model compliant with such sample invoices, are transparently imported as instances of the invoicing ontologies by means of applying the mappings defined during the learning phase. From that point on, invoices are automatically exported to whatever invoice format and model known by the system without needing to implement ad hoc (and costly) transformations.

19.6 Exploitation Roadmap

The exploitation roadmap of the approach described in this chapter starts from the integration of i2Ont technology with the PharmaInnova platform and its subsequent exploitation by PharmaInnova and its members. For this purpose, a new version of i2Ont has been developed in the form of a web application called PharmaInvoicing[14], currently accessible by PharmaInnova members only.

[14] http://www.neon-project.org/nw/Movie:_i2Ont_Web



L	Mapped	Relation
1	LABORATORIOS KIN. S.A.	NIF [string]
2	A08183386	name [string]

Path: Invoice:has_header:Header:has_invo[...]_receiver:Receiving_company:has_name:Name

E:\05-ProjectsDocs\97-LA[...]\D08_06_01\MapeoJoanPharmaInvoicing\DISPROFARMFacturaCSV_Pharma.TXT

C LABORATORIOS KIN. S.A.	A08183386	CIUTAT DE GRANADA, 123	08018	BARCEL
L 65459	21/12/2006	VACIDUR CRISTAL INCOLORO STD 1099 DORADO 190 x 0.300mm	22922000	450859
P TRANSFERENCIA			507.59	25/03/

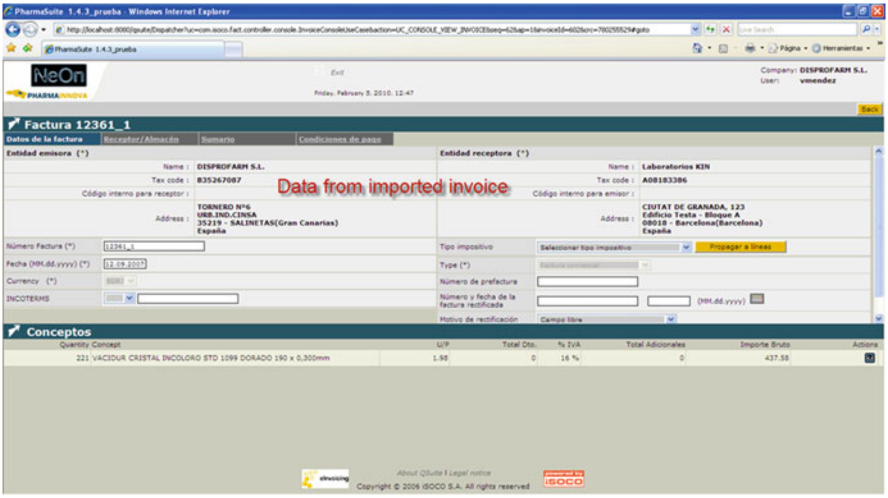
Fig. 19.5 PharmaInvoicing configuration back office

PharmaInvoicing presents functional and usability enhancements with respect to i2Ont, focused on improving import and export performance and user interaction throughout the electronic invoicing life cycle. It also includes a back office providing i2Ont's functionalities for SMEs to configure the correspondences (mappings) between their invoices, using a single sample invoice and PharmaInnova's model.

Figure 19.5 shows a screenshot of PharmaInvoicing's invoice configuration back office. On the right-hand side, the graph displayed is a graphic, interactive representation of our invoicing networked ontologies, which have been customized for the case of PharmaInnova with a look and feel more specific to the pharmaceutical domain. Usability improvements include new icons for the branches of the ontologies representing the different subdomains, e.g., geographical locations, currencies, etc., and invoice sections, e.g., header, summary, and body.

The use of PharmaInvoicing is completely transparent from the underlying knowledge representation formalism. Invoices can be imported, exported, accepted, rejected, and electronically signed, just like they are without the application of these technologies, the only difference being the savings in terms of saved money, time, and effort of IT experts in implementing ad hoc software for translating invoice data from one format and model to another. Figure 19.6 illustrates this for the case of invoice import.

As shown in Fig. 19.7, PharmaInvoicing is in the center of PharmaInnova's innovation roadmap, establishing a two-staged strategy toward exploitation of the tool. PharmaInvoicing has been deployed and is currently being used internally by PharmaInnova's members. The knowledge obtained during this stage helped validating the tool, supporting its refinement and eventual release as a product.


PharmaSuite 1.4.3_prueba - Windows Internet Explorer
Página
Herramientas
NeOn
PHARMAINNOVA
Exit
Back
Propagar a líneas
About QSuite | Legal notice
powered by iSOCO

Friday, February 5, 2010. 12:47

Company: DISPROFARM S.L.
User: vmendez

Factura 12361_1

Datos de la factura | Receptor/Almacén | Sumario | Condiciones de pago

Data from imported invoice

Entidad emisora (*)
Name : DISPROFARM S.L.
Tax code : B35267087
Código interno para receptor :
Address : TORNERO Nº6
URB.IND.CINSA
35219 - SALINETAS(Gran Canarias)
España

Entidad receptora (*)
Name : Laboratorios KIN
Tax code : A08183386
Código interno para emisor :
Address : CIUTAT DE GRANADA, 123
Edificio Testa - Bloque A
08018 - Barcelona(Barcelona)
España

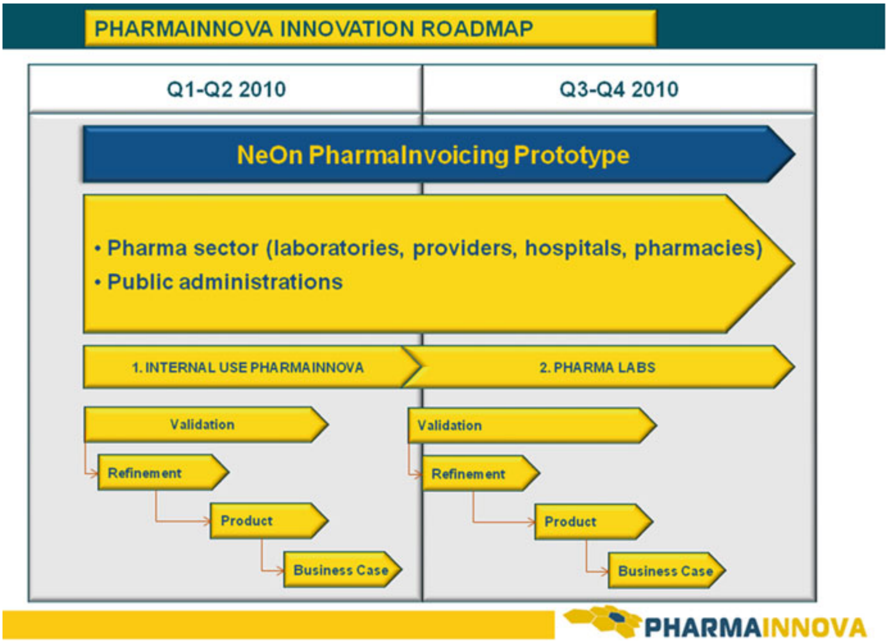
Número Factura (*): 12361_1
Fecha (MM.dd.yyyy) (*): 12.09.2007
Currency (*)
INCOTERMS
Tipo impositivo: Seleccionar tipo impositivo
Type (*): Factura comercial
Número de prefactura
Número y fecha de la factura rectificada (MM.dd.yyyy)
Motivo de rectificación: Campo libre

Conceptos

Quantity	Concept	U/P	Total Dto.	% IVA	Total Adicionales	Importe Bruto	Actions
221	VACIDUR CRISTAL INCOLORO STD 1099 DORADO 190 x 0,300mm	1.98	0	16 %	0	437.58	

Copyright © 2006 iSOCO S.A. All rights reserved

Fig. 19.6 Managing electronic invoices with PharmaInvoicing


PHARMAINNOVA INNOVATION ROADMAP

Q1-Q2 2010 | Q3-Q4 2010

NeOn PharmaInvoicing Prototype

- Pharma sector (laboratories, providers, hospitals, pharmacies)
- Public administrations

1. INTERNAL USE PHARMAINNOVA
2. PHARMA LABS

Validation → Refinement → Product → Business Case

Validation → Refinement → Product → Business Case

Fig. 19.7 PharmaInnova innovation plan

The second (still ongoing) stage benefits from this and observes the exploitation of the tool in a broader context, aiming for pharmaceutical laboratories outside of PharmaInnova.

In parallel to these two stages, exploitation opportunities for electronic invoicing out of the pharmaceutical sector will be pursued, especially in public administrations. PharmaInnova plans to offer i2Ont's functionalities in SaaS mode (software as a service), which provides computer-based services to customers over the network, reducing initial costs and avoiding maintenance tasks on the customer's side. This will allow the members of PharmaInnova to freely use the tool, while charging a fee to external users, e.g., laboratory providers, will be possible.

19.7 Conclusions

In this chapter, we have described how networked ontologies can be used to alleviate classical interoperability problems in electronic invoice exchange by means of (1) providing a formal model for the representation of knowledge related with the e-invoicing domain, which observes both e-invoicing standards and sectorial specializations, (2) serving as a semantic gateway for invoice transformation during invoice exchange, and (3) ensuring the consistency of invoices exchanged between heterogeneous systems by leveraging the expressivity of the ontologies for automatic checks. A detailed description of the developed networked ontologies has been provided, as well as of the application of the NeOn Methodology, which supported such development and simultaneously benefited from this use case as a comprehensive test bed. We have provided a short description of an application (i2Ont) implementing the approach and introduced a subsequent business roadmap for the resulting technology. Future work includes developing extensions of the invoicing networked ontologies, for a more thorough coverage of invoicing formats and models beyond EDIFACT and UBL, and the corresponding extensions of the application in order to benefit from such extensions.

References

Candini J, Gómez-Pérez JM, Méndez V, Melero R, Pariente T, Herrero G (2010) Ontologies for the pharmaceutical case studies. NeOn deliverable D8.6.1. http://www.neon-project.org/nw/images/e/e8/NeOn_2010_D861.pdf

Gómez-Pérez JM, Daviaud C, Morera B, Benjamins R, Pariente T, Herrero G, Tort G (2006) Analysis of the pharma domain and requirements. NeOn deliverable D8.1.1

Gómez-Pérez JM, Pariente T, Buil-Aranda C, Herrero G (2007) Ontologies for the pharmaceutical case studies. NeOn deliverable D8.3.1. http://tinyurl.com/32zxytz

Newell A (1982) The knowledge level. Artif Intell 18(1):87–127

Uschold M, King M, Morales S, Zorgios Y (1998) The enterprise ontology. Knowl Eng Rev 13 (1):31–89

Chapter 20
Integrating Product Information in the Pharmaceutical Sector

Tomás Pariente Lobo and Germán Herrero Cárcel

Abstract In recent years, increased attention has been paid to what is called semantic interoperability in eHealth, being the interoperable identification and description of drugs at its very core. In spite of the efforts toward having a common way to describe drugs, there is no universal nomenclature but several attempts like SNOMED CT (http://www.ihtsdo.org/snomed-ct/) or the biomedical ontologies in OBO Foundry (http://www.obofoundry.org/) and BioPortal (http://bioportal.bioontology.org/). This chapter describes an approach that applies NeOn technology to bridge the gap between different ontologies describing pharmaceutical products.

20.1 Introduction

In recent years, there has been an increasing interest in semantic interoperability in eHealth. In this domain, there is a clear need to link electronic health records (EHR) to other clinical data and biological evidence for multiple purposes. At the very core of this effort lies the need of having a common or interoperable identification and description of drugs and medical products.

Several strategies have been put in practice. Standards such as the CEN 13606[1] European norm or the HL7 v3[2] are part of the semantic interoperability efforts. The usage of SNOMED CT as a baseline for terminology is widely but not universally accepted. Besides, SNOMED CT is not a proper ontology, and the interoperability achieved using terms from this terminology is far from being complete. More interesting for us is the widely spread idea that ontologies are a very useful way

[1] http://www.iso.org/iso/catalogue_detail.htm?csnumber=40784

[2] http://www.hl7.org

T.P. Lobo (✉) • G.H. Cárcel
ATOS Origin SAE, Albarracín 25, 28037 Madrid, Spain
e-mail: tomas.parientelobo@atosresearch.eu; german.herrero@atosresearch.eu

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1_20, © Springer-Verlag Berlin Heidelberg 2012

to describe drug models. Initiatives such as BioPortal or the OBO Foundry are clear examples of the uptake of biomedical ontologies. It is in this direction where there is a clear need of a solution where different terminologies expressed in ontological form are mapped and connected in a way that the interoperability is ensured.

In order to achieve an interoperable nomenclature of drugs, a potential solution has to provide the means to (1) transform the different vocabularies and models into ontologies, (2) put all the ontologies together by creating the necessary mappings between different drug descriptions, and (3) create the infrastructure to query the ontologies using the terms that the different stakeholders are more familiar with.

In this chapter, we describe a proof of concept of this interoperable nomenclature built according to the NeOn approach. The chapter is focused on the application of different steps of the NeOn Methodology (Chap. 2) and the use of the NeOn Toolkit (Chap. 13) and some of the plugins recommended by the methodology. It is worth noting that the case study and the NeOn Methodology evolved together during the NeOn project, so the methodology received continuous feedback from real scenarios of usage in order to be eminently practical.

The chapter is structured as follows: First, we give a brief overview of the case study followed by a discussion about how we applied the NeOn Methodology to engineer the Semantic Nomenclature ontology network. The resulting ontology network is then presented, along with a brief overview of the application showcase developed to query the network. A conclusion section gives the general considerations of the chapter.

20.2 Semantic Nomenclature Use Case Description

One of the major problems in achieving a common drug description is the different stakeholders involved. Standardization bodies, governments (transnational, central, regional, or local), international public bodies such as the World Health Organization (WHO)[3] or the European Committee for Standardization (CEN)[4], private organizations like the International Health Terminology Standards Development Organisation (IHTSDO)[5], public and private hospitals, etc., provide drug data and standards for different purposes. Therefore, data heterogeneity and its frequent changes, linked to the huge amount of drugs existing nowadays, pose a major problem to solve the semantic interoperability issue.

In order to achieve semantic interoperability, we need to:

1. Enable the safe, meaningful sharing and combining of pharmaceutical data between heterogeneous systems

[3] http://www.who.int

[4] http://www.cen.eu

[5] http://www.ihtsdo.org

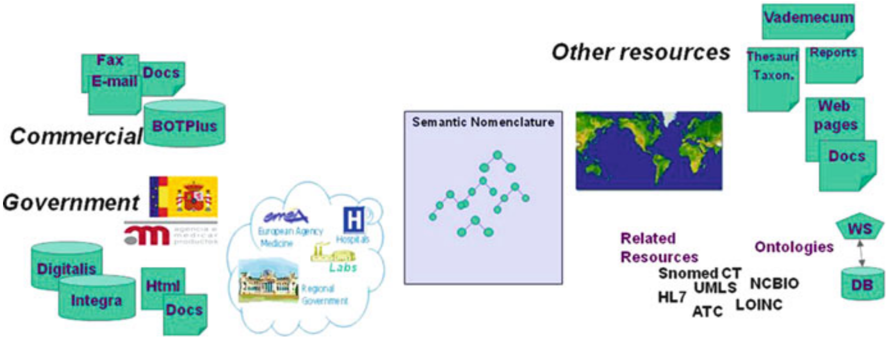


Fig. 20.1 Semantic Nomenclature stakeholders and resources overview

2. Enable the consistent use of modern terminology systems and medical knowledge resources
3. Ensure the necessary data quality and consistency to enable rigorous uses of heterogeneous data

Addressing all the previous issues needs more than a methodological and technological sound approach. It needs the agreement and collaboration of most of the named stakeholders in the overall life cycle, from the ontologies definition and mapping to the validation, evaluation, and continuous update of the results. This issue is clearly out of the scope of a project such as NeOn. The use case is therefore focused only on the methodological and technological aspects of the solution, offering a proof of concept of the NeOn approach toward the creation of a shared and consistent knowledge base about pharmaceutical products. Being part of the NeOn project, we used the NeOn Methodology and NeOn tools both at conceptual and implementation levels.

In order to test the benefits of this approach, the case study focused on a limited number of stakeholders and resources as shown in Fig. 20.1.

Figure 20.1 shows the main resources used in the case study:

- The Digitalis and Integra databases from the Spanish Agency of Medicine and Health Products (AEMPS)[6].
- The public part of the BOTPlus[7] commercial database from the General Council of Pharmacists in Spain.
- Public documents, such as official reports, public drug descriptions, HTML descriptions, etc.
- Classification of pharmaceutical terms, such as the Anatomical Therapeutic Chemical (ATC)[8] classification, a WHO recommendation mapped to many terminologies.

[6] http://www.aemps.es/

[7] https://botplusweb.portalfarma.com/

[8] http://www.whocc.no/atc_ddd_index/

- Official documents detailing description of drugs models, such as the Summary of Product Characteristics (SPC)[9] model from the European Commission. Some other non-ontological resources, like commercial nomenclatures (vademecum), pharma thesauri, widely used health standards or terminologies (HL7, SNOMED-CT, UMLS, etc.), or web pages and documents from different actors of the domain were also explored as candidate resources.

The approach followed was to model different resources and apply NeOn in order to align the models. The new models are then populated with real drug data, and the resultant knowledge base can be queried in order to obtain a more complete and interoperable drug description.

20.3 Applying the NeOn Methodology to Engineer the Ontology Network

Being both part of the NeOn project, the Semantic Nomenclature use case and the NeOn Methodology evolved together. The use case applied the NeOn Methodology (Chap. 2) and gave continuous feedback of the results of its usage. As a result of this process, the Semantic Nomenclature ontology network has been developed within this use case.

In this section, we describe a summary of the usage of the NeOn Methodology in the Semantic Nomenclature use case.

20.3.1 Ontology Requirements Specification

In the NeOn project, we started the use case definition with the requirements definition activity. Apart from gathering non-functional requirements, related mainly to issues such as scalability or reliability, the main functional requirements were gathered using competency questions (as explained in Chap. 5). These questions were proposed and answered by some of the stakeholders mentioned in the list below, which are the possible users for the ontology:

- U1: Pharmacist who is interested in searching for drugs information
- U2: BOTPlus technician whose main interest is to complete information of their commercial database with drug data from other nomenclatures
- U3: AEMPS expert who analyzes the situation of the information about drugs or updates its content

[9] http://ec.europa.eu/enterprise/sectors/pharmaceuticals/files/eudralex/vol-2/c/spcguidrev1-oct2005_en.pdf

The competency questions gathered were enumerated and classified into groups suggesting the initial list of the different concepts mentioned by the user groups. The main groups are concepts about pharmaceutical products, laboratories, and active ingredients.

As a matter of example, see below some competency questions about pharmaceutical products:

- CQ1. What is the drug commercial name?
- CQ13. Which is the drug composition?
- CQ17. Which route of administration is used?
- CQ18. What is the drug pharmaceutical form?

The complete set of competency questions, answers, and requirements obtained can be found in Gómez-Pérez et al. (2007).

20.3.2 Scenario Selection

The NeOn Methodology is scenario-based, meaning that instead of prescribing a rigid workflow, it suggests activities for a variety of scenarios. The NeOn Methodology presents and describes nine common scenarios that may arise during ontology development as described in Chap. 2.

Based on the relation between life cycle model and the scenarios, for the first iteration of the ontology development, we have followed Scenarios 1, 2, 3, and 8, while in the second iteration, when the methodology and our initial network was more advanced, we also followed Scenarios 6, 7, and 9. Not all the phases specified on the methodology for the scenarios selected have been addressed in detail, especially in the first iteration.

According to the NeOn Methodology, the ontology network life cycle model defines in an abstract way how to develop an ontology network project and how to organize the processes and activities into phases or stages. Due to the incremental development of the ontologies followed in the project, the ontology network life cycle model selected for the use case was the Iterative-Incremental Ontology Network Life Cycle Model. This model organizes the ontology development in a set of iterations (or short mini-projects with a fixed duration). Each iteration is scheduled as a single ontology project using a waterfall model. Requirements specified in the Ontology Requirements Specification can be divided in different subsets, and implements throughout the different iterations.

Two main iterations have been selected for the development of the Semantic Nomenclature ontology network. Six-phase Waterfall Model was selected for the both iterations. This model allows the reengineering of ontological resources and non-ontological resources (NORs), which was in the scope of the case study.

20.3.3 Ontological Processes and Activities Performed

In this section, we summarize the most interesting processes and activities followed according to the NeOn Methodology and the scenarios selected in the use case. Figure 20.2 shows the main activities followed by the Semantic Nomenclature case study and included in the incremental life cycle model.

It is worth noticing that the processes and activities followed in both iterations were basically the same. This is due to the fact that the result of the first iteration was a preliminary version of the ontology network that was further elaborated and extended in the second version.

- Support activities: The use case started with a survey of the domain. As part of Scenario 1 of the NeOn Methodology, we followed the methodological support activities, such as the Ontology Environmental Study and Ontology Feasibility Study, in parallel to the Ontology Requirements Specification activity (as explained in Sect. 20.3.1). In this phase, we decided that a network of ontologies was the best approach for the use case. It is worth noting that in the second iteration of the case study, we did a second round of the Ontology Requirements Specification activity adding new requirements, especially from hospitals toward the differentiation between commercial and clinical drugs. However, the overall objective and approach of the use case remained unchanged. The

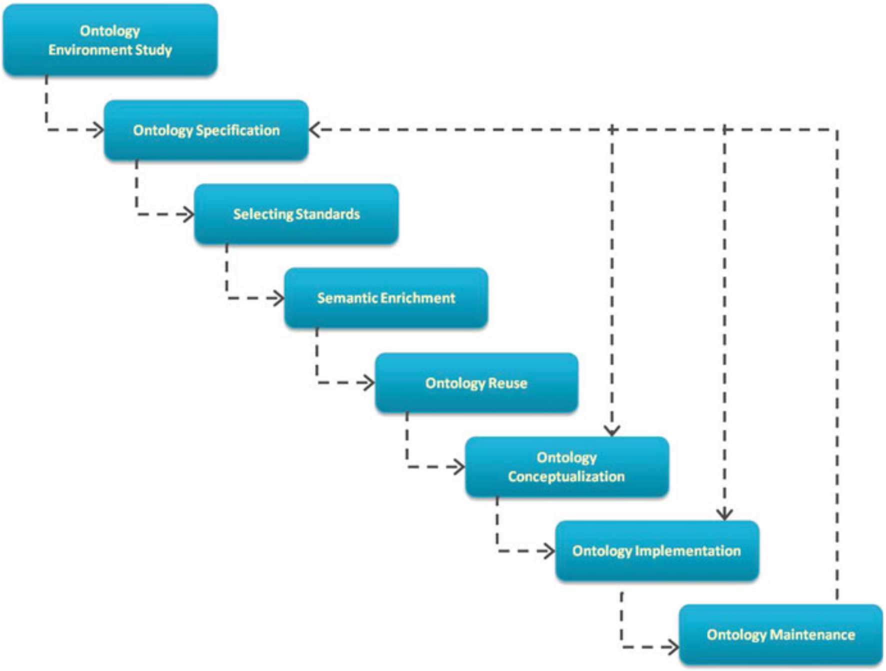


Fig. 20.2 Semantic Nomenclature use case ontological activities

second iteration was closer to the incremental approach "produce and deliver" new ontologies to the case study, but the already available ones were not discarded but improved. We did not use the template for ontology requirements specification document (ORSD) provided by the methodology because it was drafted during the last period of the project, but we used a similar approach by gathering requirements, competency questions, etc. In the second iteration, we did a proper scheduling activity by using the tooling support provided by the NeOn Toolkit (the gOntt plugin explained in Chap. 14) in order to generate a plan for the iteration.

- Reusing resources: Following Scenarios 2 and 3, we tried in parallel to reuse as much as possible already available ontologies in the domain and model new ontologies from existing non-ontological resources. For the non-ontological resources, we carried out several activities proposed by the NeOn Methodology, such as resource search, resource assessment, or resource selection. The result of this process was the selection of several resources: the ATC classification schemas created manually and populated automatically from the WHO ATC XML version, the Digitalis and BOTPlus ontologies created from the abovementioned databases using the R2O-ODEMapster plugin, and the SPC ontology created manually from the SPC standard specification.
- In parallel, we tried to reuse as much as possible already available ontologies in the domain. We followed several activities proposed by the NeOn Methodology, such as ontology search, ontology assessment, ontology comparison, and ontology selection for ontology reuse. The results of Scenarios 2 and 3 can be found in Gómez-Pérez et al. (2007).
- In the second iteration, we used the ontology design patterns to better define the ontologies. Apart from generic content patterns such as "part of", we released a candidate pattern for the description of drugs to distinguish between clinical drugs, prescription drugs, and pharma-marketed products.
- Semantic enrichment: In the second iteration, as part of Scenario 8, we carried out a semantic enrichment activity, including new concepts and relations especially to define clinical drugs in our Semantic Nomenclature ontology.
- Ontology conceptualization: The goal of the conceptualization activity is to organize and structure the knowledge into meaningful models at the knowledge level. We defined the main concepts and relations and decided on the overall model. In this activity, the Semantic Nomenclature ontology network took shape. As a result of the activity, we defined a new ontology (the Semantic Nomenclature ontology) that acts as a bridge of the different ontologies present on the network. We also decided on the type of alignments to be made between that ontology and the rest of the ontologies selected or defined for the ontology network.
- Ontology implementation: The main goal of the ontology implementation activity is to create a computable model implemented in an ontology language from the conceptual model created in the ontology conceptualization activity. In the Semantic Nomenclature, we used OWL2 and the NeOn Toolkit for the implementation. In this activity, we used several NeOn Toolkit plugins such as the

alignment plugin (and alignment server) to generate candidate alignments between some of the ontologies of the ontology network, R2O and ODEMapster to map the BOTPlus and Digitalis public databases to our ontology schemas in order to generate the initial set of individuals, Cicero to develop the ontology network in a collaborative fashion, Watson plugin to search for available reusable ontologies, RaDON to continuously verify the coherence of the ontology network, SPARQL for internal tests, and finally OWL-Doc for documentation purposes.

- Ontology evaluation: At the end of the project, we applied this support activity in order to evaluate the resulting ontology network and application. On the one hand, we used RaDON in order to evaluate the soundness of the ontologies of our network. On the other hand, from the domain perspective, we counted with several users to perform the evaluation. This final evaluation consisted of several interviews with users and a training and evaluation hands-on session performed at the end of the project.

20.4 Semantic Nomenclature Ontology Network

The Semantic Nomenclature ontology network comprises a set of ontologies organized in four levels: the representation ontology (OWL), general ontologies, domain ontologies, and application ontologies.

The Semantic Nomenclature (SN) ontology plays a central role in the network (see Fig. 20.3), acting as a bridge between the different knowledge representations from the application and domain ontologies.

The domain level comprises ontologies defining the main notion and concepts of the pharmaceutical domain. At this level, we include ontologies providing a classification of pharmaceutical terms, such as the ATC and SPC ontologies created within the case study, or the RxNorm[10], the UMLS Metathesaurus[11], MeSH[12], OpenGALEN[13], and NCI[14], all reused or reengineered from existing ontologies or resources. Some of these ontologies have been just partially mapped to the network as a proof of concept. We did some tests using the alignment plugin in order to perform these mappings. In some cases, the candidate alignments were good

[10] RxNorm terminology produced by the National Library of Medicine (NLM) http://www.nlm.nih.gov/research/umls/rxnorm/

[11] UMLS Metathesaurus http://www.nlm.nih.gov/pubs/factsheets/umlsmeta.html

[12] National Library of Medicine's (USA) controlled vocabulary thesaurus http://www.ncbi.nlm.nih.gov/mesh

[13] OpenGALEN http://www.opengalen.org/

[14] NCI Metathesaurus http://ncim.nci.nih.gov/

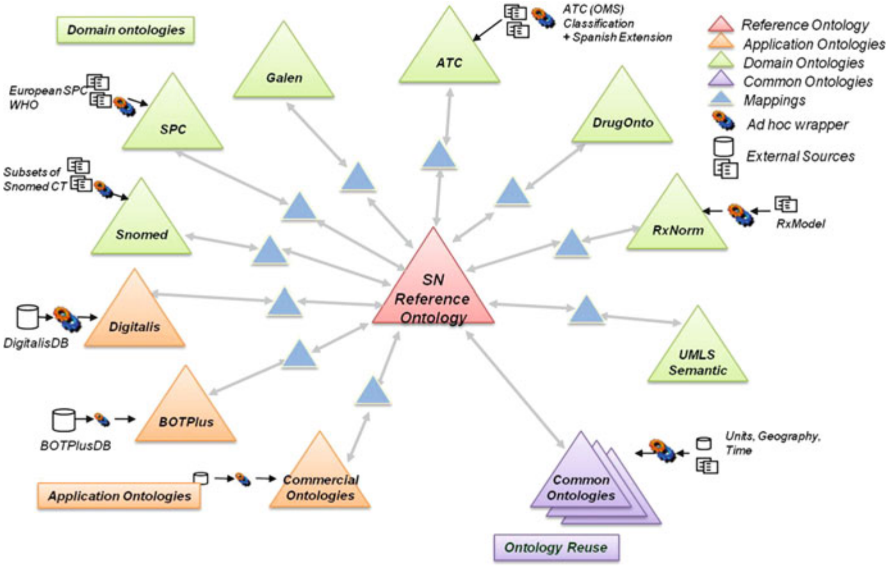


Fig. 20.3 Nomenclature ontology network

enough, although the overall impression was that a lot of manual work had to be done (see Chap. 12 for more details on ontology alignment). New domain ontologies can be added to the network by creating the necessary mappings.

At the Application level, we have ontologies representing knowledge of real-world resources used for a specific purpose or application. This is the case of the Digitalis and BOTPlus ontologies, containing governmental and private views of commercial pharmaceutical products in Spain.

We have reused existing ontologies for defining common domain elements. After looking at different ontologies, we chose the W3C time ontology for managing dates, parts of the Galen ontology for the definition of units of measurement, and the geographical module from the Simile Ontology.

The Semantic Nomenclature ontology network is shown in Fig. 20.3. A detailed description of the ontologies can be found in Candini et al. (2010).

The ontology network makes possible the easy interoperability and integration of the distributed resources for the description of pharmaceutical products. Moreover, the ontology network facilitates the aggregation of drug-related information connecting new ontological resources via mappings to the SN ontology. This solution makes possible searching for aggregated information by querying the knowledge base using SN ontology elements or domain elements. Apart from the obvious usage on the semantic interoperability area, these queries allow the different stakeholders to potentially keep up to date their back-office systems or take better decisions based on the aggregated information presented.

20.5 Semantic Nomenclature Application

The Semantic Nomenclature application is eye catching for the pharmaceutical community as a new nomenclature (compendium) based on semantic web technologies. The application is targeting mainly pharmaceutical-knowledge experts. The main result is focused on the runtime aspects of NeOn, but it relies on the work done at the design time on the ontologies using the NeOn Toolkit, different NeOn plugins, and the NeOn Methodology. Apart from being a test bed of the NeOn runtime services, the goal is to offer a view over a set of networked ontologies, allowing functionalities such as querying, adding new ontologies to the network, rating of ontology elements, or adding new ontology mappings.

The Semantic Nomenclature application is supported at the infrastructure level by a knowledge base (KB) populated according to the Semantic Nomenclature ontology network model. This KB contains relevant information about pharmaceutical products and associated knowledge about other types of entities like active ingredients, diseases, laboratories, etc. The Semantic Nomenclature application provides a feature set in different dimensions: (a) accurate query answering mechanism by using a rich web client form that abstracts the end user from the underlying SPARQL construction of queries, (b) access to aggregated data using the Cupboard NeOn service, and (c) collaborative and social functionalities using the Cicero NeOn Service integrated from the web application.

There is also a different angle to consider, which is related with the Open Linked Data initiative. Linked Data[15] is a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF. Among the current Linked Data datasets currently available there are several health-related open resources, such as Drugbank[16], Dailymed[17], or Diseasome[18], and also several other generic-purpose resources containing information about drugs, such as DBPedia[19]. Although using Linked Data was not a requirement for the use case, we also included in the current prototype a possible link to make use of Linked Data from the Semantic Nomenclature application and a small attempt to allow the creation of mappings between our ontologies and some of the Linked Data datasets. This is nevertheless a work in progress aimed at showing a possible path for future enhancements.

The implementation of the application deals with the integration of the ontology network presented before in a user-friendly web application. The implementation has two main different layers: a business processing layer at server side and a

[15] http://linkeddata.org/

[16] http://www.drugbank.ca/

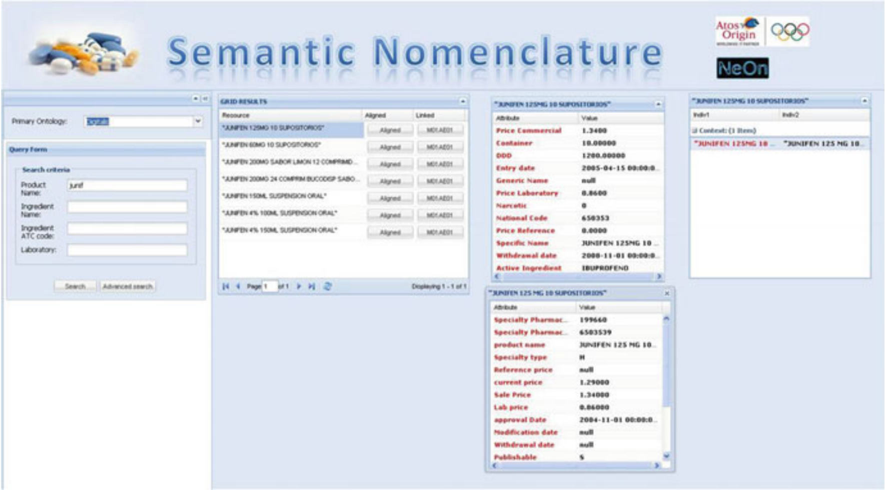
[17] http://dailymed.nlm.nih.gov

[18] http://diseasome.eu/

[19] http://dbpedia.org/

presentation layer at the client side. On the one hand, the server is essentially dedicated to data processing and management of the functional process, and its architecture is generic to interact with several software components provided by NeOn or other third parties. On the other hand, the client side is dedicated to the information presentation and the data interaction. The application uses Google Web Toolkit (GWT)[20] to allow a rich presentation layer. The software integration was facilitated by the communication mechanism provided by the GWT technology. For example, the Remote Procedure Call (RPC) mechanism permits to interoperate with different kind of software components as a semantic repository, or the NeOn plugins through web services. Moreover, the JSON[21] communication language between the server and client sides allows the exchange of structured data.

The users of the Semantic Nomenclature web prototype are mainly pharmaceutical-knowledge experts with a limited knowledge of ontologies. It is not intended for users with no knowledge about ontologies at all, but they do not need to be ontology experts to get benefits from using the prototype. But the prototype is not closed to those domain actors; it is open to any other kind of users, such as people who want to retrieve semantic-enriched information about pharmaceutical products, personnel from hospitals, governmental agencies, etc. Moreover, the prototype provides functionalities to biomedicine, pharmacy, or health-care researchers to assess about the models of the ontologies used by the prototype, add new models, or discuss with other colleagues.



Primary Ontology: [illegible]

Query Form

Search criteria

Product Name: junif

Ingredient Name:

Ingredient ATC code:

Laboratory:

Search | Advanced search

GRID RESULTS

Resource	Aligned	Linked
"JUNIFEN 125MG 10 SUPOSITORIOS"	Aligned	M01AE01
"JUNIFEN 60MG 10 SUPOSITORIOS"	Aligned	M01AE01
"JUNIFEN 200MG SABOR LIMON 12 COMPRIMID...	Aligned	M01AE01
"JUNIFEN 200MG 24 COMPRIM BUCODISP SABO...	Aligned	M01AE01
"JUNIFEN 150ML SUSPENSION ORAL"	Aligned	M01AE01
"JUNIFEN 4% 100ML SUSPENSION ORAL"	Aligned	M01AE01
"JUNIFEN 4% 150ML SUSPENSION ORAL"	Aligned	M01AE01

Page 1 of 1 Displaying 1 - 1 of 1

"JUNIFEN 125MG 10 SUPOSITORIOS"

Attribute	Value
Price Commercial	1.3400
Container	10.00000
DDD	1200.00000
Entry date	2005-04-15 00:00:0...
Generic Name	null
Price Laboratory	0.8600
Narcotic	0
National Code	650353
Price Reference	0.0000
Specific Name	JUNIFEN 125MG 10 ...
Withdrawal date	2008-11-01 00:00:0...
Active Ingredient	IBUPROFENO

"JUNIFEN 125 MG 10 SUPOSITORIOS"

Attribute	Value
Specialty Pharmac...	199660
Specialty Pharmac...	6503539
product name	JUNIFEN 125 MG 10...
Specialty type	H
Reference price	null
current price	1.29000
Sale Price	1.34000
Lab price	0.86000
approval Date	2004-11-01 00:00:0...
Modification date	null
Withdrawal date	null
Publishable	S

"JUNIFEN 125MG 10 SUPOSITORIOS"

Indiv1	Indiv2
Context: (1 Item)	
"JUNIFEN 125MG 10 ...	"JUNIFEN 125 MG 10...

Fig. 20.4 Semantic Nomenclature web default perspective

[20] http://code.google.com/webtoolkit/

[21] http://www.json.org/

A screenshot of the application can be shown in Fig. 20.4. This screenshot shows different widgets used for selecting ontologies and querying and displaying the results of the query.

20.6 Conclusion

In this chapter, we described how we applied the NeOn Methodology and the NeOn Toolkit and some of its plugins on the development of an ontology network in the scope of the Semantic Nomenclature use case developed within the NeOn project. We explained the current situation and the interoperability problems posed by the existence of multiple stakeholders, the heterogeneity of the different solutions for drug description, and the huge amount of data involved, being this issue at the very core of the semantic interoperability in eHealth effort. A detailed description of the methodological activities carried out in the use case toward the definition of the Semantic Nomenclature ontology network has been presented. As the NeOn Methodology and the use case evolved in parallel, both received continuous feedback from each other during the project life span. We have also briefly presented an overview of the Semantic Nomenclature web application that takes advantage of the underlying ontology network.

References

Candini J, Gómez-Pérez JM, Méndez V, Melero R, Pariente T, Herrero G (2010) Ontologies for the pharmaceutical case studies. NeOn deliverable D8.6.1. http://www.neon-project.org/nw/images/e/e8/NeOn_2010_D861.pdf

Gómez-Pérez JM, Pariente T, Buil-Aranda C, Herrero G (2007) Ontologies for the pharmaceutical case studies. NeOn deliverable D8.3.1. http://tinyurl.com/32zxytz

Index

M.C. Suárez-Figueroa et al. (eds.), *Ontology Engineering in a Networked World*,
DOI 10.1007/978-3-642-24794-1, © Springer-Verlag Berlin Heidelberg 2012