NeOn-project.org

NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

---

D 7.4.1 Software architecture for managing the Fisheries ontologies lifecycle

---

Deliverable Co-ordinator: Óscar Muñoz-García (UPM), Soonho Kim (FAO)

Deliverable Co-ordinating Institution: UPM

Other Authors:

Soonho Kim, Marta Iglesias Sucasas, Caterina Caracciolo, Andrew Bagdanov (FAO);

Yimin Wang, Peter Haase (UKARL);

Óscar Muñoz-García, María-del-Carmen Suárez-Figueroa, Asunción Gómez-Pérez (UPM);

Document identifier:	NEON/2007/D 7.4.1/v1.0	Date due:	August 31, 2007
Class deliverable:	NEON EU-IST-2005-027595	Submission date:	August 31, 2007
Project start date:	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	final
		Distribution:	Public

NeOn Consortium

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

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta} @open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 28 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.upm.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SI-1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 655 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Koblenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Martino della Battaglia, 44 - 00185 Roma-Lazio, Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 1 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

The following partners were actively involved in the work that this document is based on:
CNR, FAO, ONTO, OU, SAG, UKARL, UPM.

Change log

Version	Date	Amended by	Changes
0.1	14/07/07	Soonho Kim	Fisheries ontologies lifecycle overview chapter
0.2	15/07/07	Marta Iglesias	Introduction, plan for next steps chapter; and revision of the fisheries ontologies lifecycle overview.
0.3	31/07/07	Yimin Wang	Architecture chapter and detailed architecture for first prototype D7.4.2
0.4	06/08/07	Soonho Kim	Conclusions chapter
0.5	06/08/07	Oscar Muñoz	Compilation of the completed document with input from different partners; first draft of use cases chapter; compilation of partner's input for Annex 1
0.6	09/08/07	Marta Iglesias	Revision of the completed document; feedback passed to all authors for improvements; and editing of various sections.
0.7	16/08/07	Marta Iglesias, Soonho Kim, Yimin Wang, Oscar Muñoz	Revision and update of Chapters 2, 3, 4 and 5
0.8	22/08/07	Marta Iglesias, Soonho Kim	Revision and update of various chapters based on Peter Haase comments
0.9	30/08/07	Romolo Tassone, Caterina Caracciolo, Andrew Bagdanov, Marta Iglesias, Soonho Kim	Linguistic revision. Revision of Charter 2, 3
0.9.1		Caterina Caracciolo, Marta Iglesias	Revision of Chapter 2, 3 in main document.
0.9.2		Coordinators: Andrew Bagdanov, Soonho Kim. Contributors: Michael Erdmann, Mathieu D'Aquin, Boris Villazón, Yimin Wang, Mauricio (UPM)	Revision of Annex I
0.9.5		Óscar Muñoz	Revision of Annex I
0.9.6		José Manuel Gómez Óscar Muñoz	Changes after QA
1.0	15/10/07	Angela Wilkinson	Final QA

Executive Summary

This document describes the software architecture for implementing a system for managing the fisheries ontologies lifecycle, cornerstone of WP7. The lifecycle described in this deliverable is an instantiation of the possible lifecycles that can be carried out with the NeOn toolkit, and has been selected according to FAO needs, style of work and organizational culture. This document is also an extension and refinement of D7.1.1 in terms of requirements that should be fulfilled by the general NeOn lifecycle management support. The architecture is based on the NeOn Toolkit and engineering components, and will be incrementally integrated into the system through subsequent T7.4 deliverables, i.e. deliverables D7.4.2 and D7.4.3.

Since D7.1.1 the requirements about the Fisheries Ontologies Lifecycle have evolved been more concrete (as in any software development process). Chapter 1 of the document describes the major requirements of the Fisheries Ontologies Lifecycle Management System. Chapter 2 provides an overview of the process required for building, populating, publishing and maintaining the fisheries ontologies.

Chapter 3 includes use case diagrams and a brief description of each use case required for implementing the system. These use cases are based on the requirements described in early WP7 deliverable D7.1.1, and each use case provides references to the relevant sections in D7.1.1. A detailed description of each use case and the engineering components to be integrated in the system are included in Annex I of this document.

Chapter 4 presents the system architecture derived from the use cases, as a selection of components of the NeOn Toolkit and system architecture. Chapter 5 describes the plan for subsequent T7.4 deliverables. Finally, Chapter 6 highlights the major conclusions of deliverable D7.4.1.

Table of Contents

List of tables

List of figures

1 Introduction

The ontologies produced within WP7 will be used by the Food and Agriculture Organization of the United Nations (FAO) in various medium-to-large web-based applications and in particular, they will underpin the Fisheries Stock Depletion Assessment System (FSDAS).

To implement these ontologies, it is necessary to create mechanisms that allow the actors involved to create and maintain distributed ontologies (and ontology mappings), in the fishery knowledge community, by deploying NeOn methodologies and technologies. To support the continuous growth of the networked ontologies customized to the fisheries domain, these mechanisms must include:

- creation and maintenance of data models, content, relationships and mappings, modularization and versioning;
- functionality for checking ontology validity, integrity, and relevance; and
- methods for ontology-learning and population.

For the successful implementation and service delivery of the FSDAS, it is crucial to support ontology editing and maintenance, because the nature and concepts used in a domain like fisheries are continuously evolving. Therefore, it is essential that these changes are continuously applied to the ontologies.

It is important to note that most of the ontologies in the fisheries domain will not be developed from scratch, but migrated from legacy systems. Therefore, the lifecycle management system must provide adequate support to either migrate or map to ontologies data, from relational databases or XML schemas.

In addition, while requirements for ontology design, population and validation are common in ontology engineering environments, the WP7 case study looks for a more articulated approach, paying special attention to the editorial workflow, key to ensure that users can modify and update ontologies in a controlled and coherent manner, especially for those ontologies already deployed on the Internet. At the same time, this controlled environment for the editorial workflow will support the appropriately version ontologies on the Internet, to ensure semantic web applications reliability on the ontologies exploited.

Ontology editors will be involved in the editorial workflow – mainly subject experts and not ontology engineers. Ontology editors will control the development of fragments of ontologies, revise the work of others, or develop multilingual versions of ontologies. Therefore, intuitive interfaces that hide the purely ontological and engineering options must be integrated with the toolkit.

Thus, one of the major objectives of the WP7 case study is to have a strong, reliable and intuitive ontology management system to edit the networked ontologies that the applications will use as their basis.

1.1 Interactions with other deliverables and WPs

D7.4.1 "Software architecture for managing the fishery ontologies lifecycle" is the first deliverable of task T7.4 "Software for managing the fishery ontologies lifecycle". It is intended to design, implement, test and deploy an instantiation of the NeOn Toolkit and infrastructure tailored to the fisheries domain.

The initial requirements for the Fisheries Ontologies Lifecycle Management System have been gathered from its future users (FAO's ontology engineers and ontology editors) and were summarized in D7.1.1 "User requirements specifications for the fishery ontologies lifecycle, knowledge tools and alert system".

D7.4.1, takes into consideration D6.2.1 "Specification of NeOn reference architecture and NeOn APIs"; D6.3.1 "First Implementation of critical Infrastructure Components"; and other deliverables from technical work packages WP1-WP4, which will provide plug-ins and engineering components to deliver D7.4.1 use cases.

In addition, the subsequent deliverables of T7.4 will implement the software for managing the fishery ontologies lifecycle through various prototypes in an iterative manner, by designing, deploying, testing and passing recommendations to the following iteration, allowing users to provide early feedback on the prototypes. This will be detailed in Chapter 5 “Plan for next steps”.

1.2 Overview of this deliverable

The rest of this deliverable is organized in the following way:

Chapter 2 is an in-depth description of the process for managing the fisheries ontologies; their conceptualization, development, validation, release and maintenance. This chapter also describes the users involved in the ontology lifecycle; their type of knowledge and skills and their roles in each step of the lifecycle.

After introducing the major steps in the ontology lifecycle, two subsections contain the steps to populate ontologies from legacy systems, and the editorial workflow. These steps and the related use cases are of special importance in the WP7 case study. The methodological aspects explained in these two sections must be implemented in the NeOn toolkit in the sense that the tool must guide the users through the process described.

Chapter 3 presents the use case diagrams with a brief description. The use case diagrams display the various actors and their interrelationships. Each use case has a one-paragraph summary, typically of the main success scenario, that is the scenario the most frequent scenario of those that the complete use case description describes. The complete use cases descriptions are detailed in the annex of this document. The use case descriptions were complied during early requirement analysis. The full description of use cases and architectural components are included in Annex I. The model of the system's functionality and environment was developed following the software engineering Unified Process methodology (Larman, 2005).

Based on the use case model, the architecture for the Fisheries Ontologies Lifecycle Management System is presented in Chapter 4 as an instantiation of the general NeOn architecture proposed in WP6. According to this, the lifecycle described in this deliverable will be one of the possible that could be carried out with the NeOn toolkit.

Chapter 5 provides an overview of the T7.4 task, the task’s subsequent deliverables, and feedback and constraints based on other WP7 deliverables and other NeOn project deliverables. This chapter also highlights the detailed architecture for the first system prototype to be delivered by M24 (D7.4.2).

Finally, Chapter 6 presents the conclusions.

2 Overview of the fisheries ontologies lifecycle

2.1 Users

There are two different profiles of the users that will manage the fisheries ontologies lifecycle: ontology engineers and ontology editors.

2.1.1 Ontology engineers

These users are specialized in ontology modelling techniques. They possess basic to advanced knowledge of ontology engineering tools and inference engines, but may know little about the domain to be modelled. Usually, ontology engineers are in charge of defining the initial ontology skeleton and hence they take into account the purpose of the ontology, possible interactions with legacy systems and other relevant issues.

2.1.2 Ontology editors

These users are mainly domain experts, although they can include information management specialists, terminologists or translators. Ontology editors are in charge of the daily editing and maintenance of the networked multilingual ontologies. They can control the development of specific ontology fragments, revising the work of others, and develop multilingual versions of ontologies. Ontology editors know about their ontology's domain, but usually know little or nothing about ontology software or ontology design issues. For this reason, they need to be provided with simpler interfaces than those available to ontology engineers (to hide most of the purely ontological and engineering options).

2.2 Roles

Only authorized users will be able to participate in the fisheries ontologies lifecycle – system administrators would determine access. Authorized users will be assigned roles to various ontology modules as ontology engineers, subject experts, validators or viewers, depending on their user rights and assigned tasks.

Subject expert, validator and viewer correspond to the possible roles of the ontology editors within the editorial workflow:

- **Subject experts** are the editors inserting or modifying ontology content.
- **Validators** revise, approve or reject the changes made by subject experts. Only validators can copy changes into the production environment for external availability.
- **Viewers** are users who are authorized to view approved information about ontologies, but they cannot edit the ontologies.

Feedback from WP4 will be obtained in order to take into account the users rights derived from the role description,

2.3 Major steps

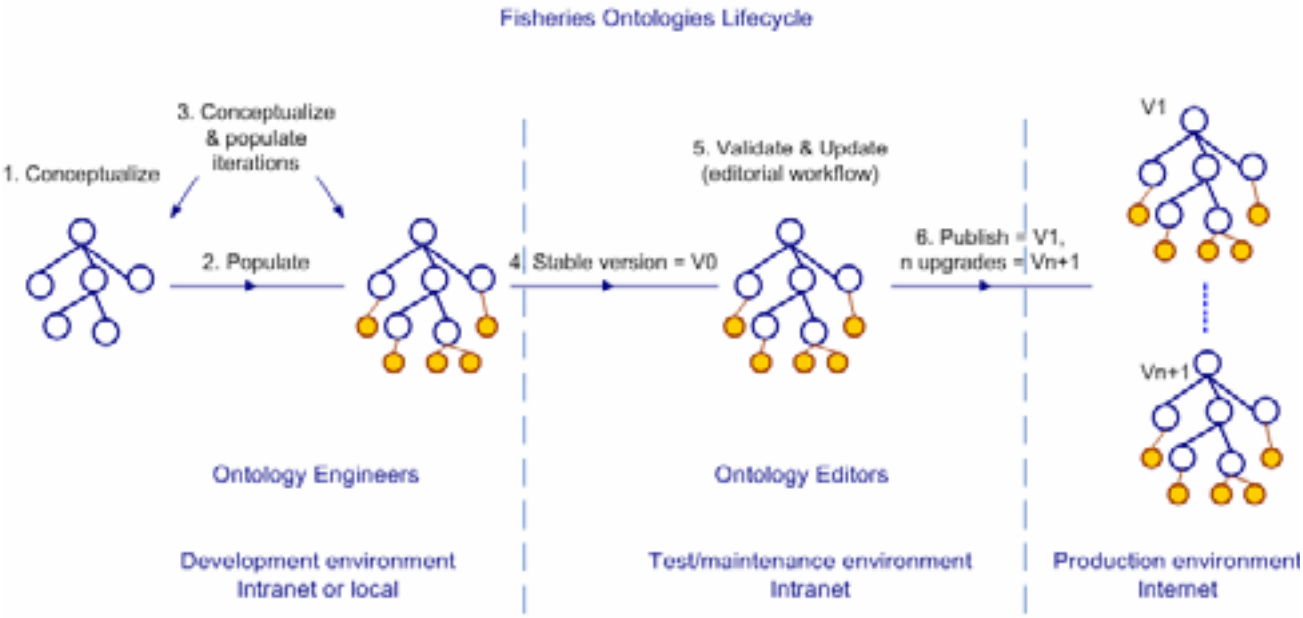


Figure 1. Major steps in the fisheries ontologies lifecycle

As illustrated in Figure 1, the fisheries ontologies lifecycle consists of the following major steps:

2.3.1 Ontology conceptualization

Ontology engineers organize and structure the domain information into meaningful models at the knowledge level. In the fishery domain, information is collected from the fisheries' FIGIS [FIGIS07] databases, fisheries' fact sheets and other information systems and documents, and analyzed with fisheries domain experts in FAO [FAO07]. The conceptualization (knowledge acquisition) process results in an ontology model with most of the concept level entities, such as classes, properties and restrictions.

2.3.2 Ontology population

Ontology engineers perform the ontology population, annotation or aggregation from unstructured information sources activities with various manual or (semi)automatic methods to transform unstructured, semi-structured and/or structured data sources into ontology instances. In the fisheries domain, this process consists mainly of converting semi-structured data sources (fishery fact sheets in XML format) and structured data source (from RTMS [RTMS07] relational database) into corresponding instances in the conceptualized fisheries ontology. The ontology population methods are explained in detail in the following sections.

2.3.3 Iteration of conceptualization and population process until getting a stable version

Ontology engineers will iterate the conceptualization and population processes until a populated ontology is achieved that satisfies all requirements and is considered stable. Once achieved, the ontology will enter into the test and maintenance environment, implemented through the editorial workflow. Since population depends on the model represented in the ontology, this iterations could

be seen as doing prototypes, evaluating them and, using the experience obtained, making changes to the conceptualization in order to repeat the population process with a different conceptualization till a ontology that satisfies all requirements is achieved.

2.3.4 Ontology validation and update through editorial workflow

The editorial workflow will allow ontology editors to consult, validate and modify the ontology and its instances, keeping track of all changes in a controlled and coherent manner. Any ontology that is to be released in the production environment needs to pass through the editorial workflow (including all upgrades). The editorial workflow is explained in detail in the following sections.

2.3.5 Ontology publication

Some ontology editors are in charge of validation and can make advises on their technical soundness for use in IT systems. Once the ontology editors that are in charge of validation consider the ontology final, they are authorized to release it on the Internet and make it available to end users and systems. A release will consist of copying the ontology from the maintenance environment to the production environment, which for FAO will be the Internet. Since ontologies published on the Internet will be always consider stable they will always be versioned; V1 for the first published version and Vn+1 for each subsequent upgrade. All versions will be continuously available (with necessary metadata) to ensure that third party semantic web applications that rely on a previous version will continue to function until they are adapted to the last version.

2.4 Fisheries ontology population

Fisheries ontologies will be manually populated from legacy systems, or by (semi)automatic methods. For manual population, ontology engineers transform real objects from a variety of data sources into instances in the fishery ontologies. This conversion is done once and will require manual updating each time the corresponding data sources change. Manual transformation has a high cost of labour and time, and thus will only be used in WP7 when no other (semi)automatic method is possible.

For (semi)automatic population, there are two different approaches:

- Population based on lifting (migration): each real object in the data sources would be migrated into the ontologies as a formal instance.
- Population based on mapping: each real object would stay in the data sources themselves. Mapping is only done by the ontologies and data sources. At the run-time, the mapping will convert the ontology-based query into a data source-based query to provide a variety of activities for retrieving instances.

Population based mapping is suitable when the schema of data sources is similar to ontology models, in terms of the complexity of the schema. However, population based on lifting would be best when the schema of data sources is different from the ontology models. However, it is most important that any (semi)automatic methods support all kinds of data sources, such as RDBMS or XML. This activity is processed by semi-automatic methods on the intranet or local environment.

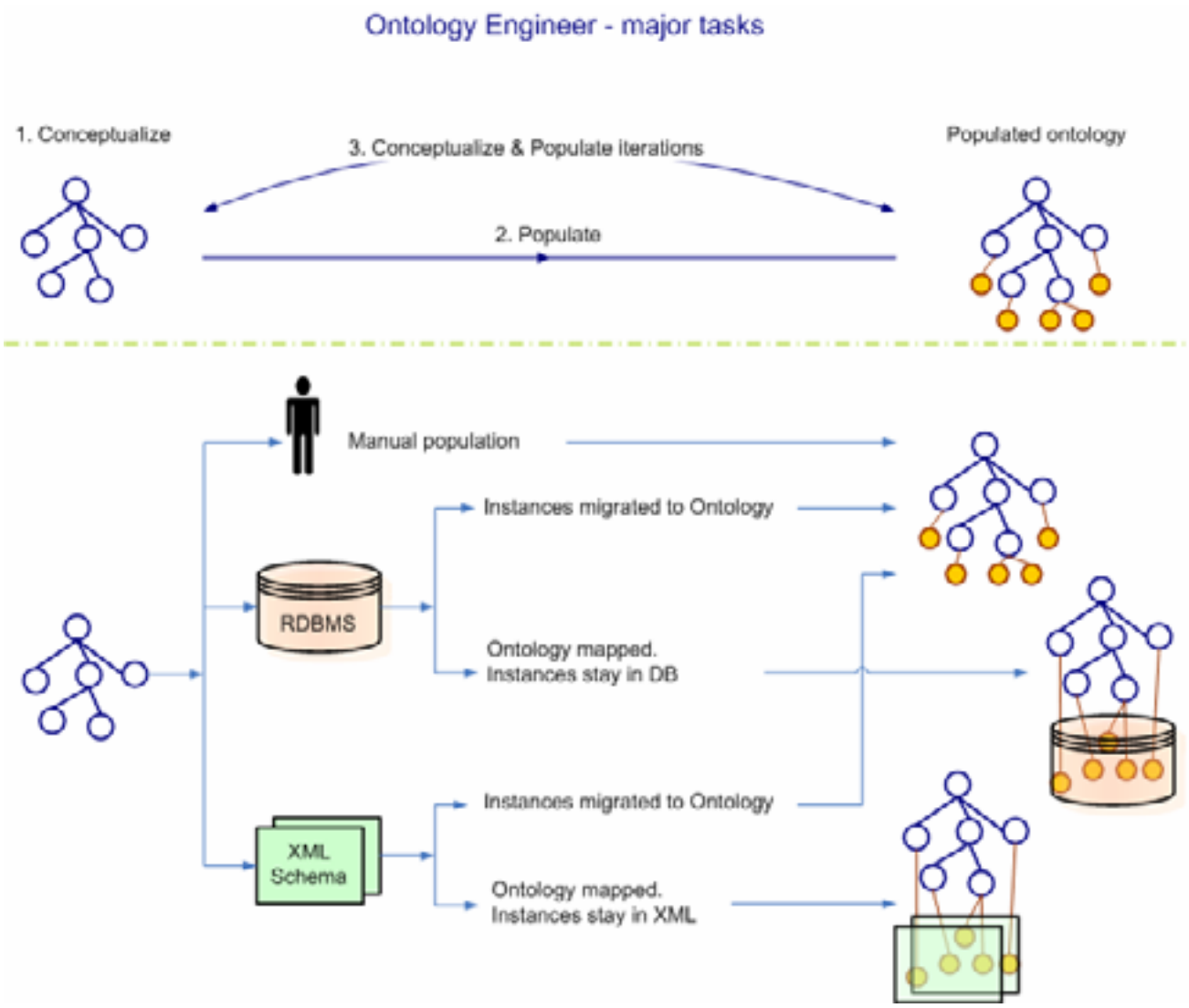


Figure 2. Methods for fishery ontologies population

2.5 Editorial workflow

The fisheries ontologies editorial workflow will allow ontology editors to consult, and if authorized, validate and/or modify ontologies in a controlled and coherent manner, ensuring that only fully validated ontologies will be released on the Internet (for external availability). The editorial workflow is important in the fishery ontologies, because it allows for the controlled access, verification and final approval of the fishery ontology elements. In this document ontology elements refer to classes, properties instances or modules of the ontologies.

To assure the ontology quality, only if each element is approved through the whole workflow, can it be released to the public? If an item were rejected at any stage, it would need to be revised, corrected and then resubmitted to the approval process.

The whole editorial workflow is illustrated in next figure. States are denoted by rectangles and actions by arrows

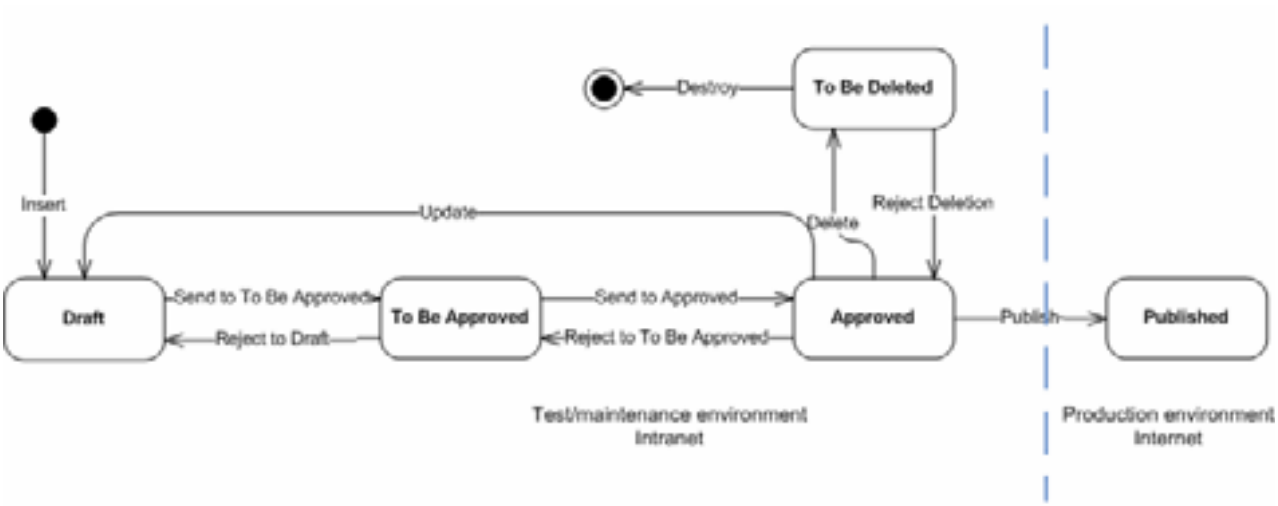


Figure 3. Overview of the whole editorial workflow.

2.5.1 Element status

The workflow is based on assigning a status to each element of the ontology. The ontology is published or upgraded only when all the elements have an approved status.

The possible statuses for each element are:

- **Draft**: this is the default status automatically assigned to each new element inserted by subject experts. Draft status is also assigned to elements that were previously in the approved status, and then modified (updated) by a subject expert. The draft status is the first status of an element entering the editorial workflow.
- **To be approved:** when a subject expert is satisfied with the element he/she have in draft status and consider them ready to be approved by validators, the subject expert sends the elements to the “to be approved status”. The element will remain in that status until the relevant validator approved (or rejects) it.
- **Approved**: this status is attained when a validator approves an element in the “to be approved” status.
- **Published**: this is the status of all elements, in any version of an ontology that has been released on the Internet.
- **To be deleted**: if a subject expert considers that an element is to be deleted, the item would be assigned the “to be deleted” status. A validator is required to delete the element.

2.5.2 Actions and roles

The workflow will allow the system to define who (depending on the role of the user) can do what (actions explained below) and when (depending on the status of the element and the role of the user).

Subject experts will be able to:

- **Insert** a new element, or **update** an approved element: in both cases, the system will automatically assign a draft status to the element. These two actions trigger the start of the workflow. When an element has a draft status, subject experts can update it as required. The

item will keep its draft status until the subject expert decides that is ready for approval – it then moves to the next status.

- Send "**to be approved**": the subject expert changes the status of an element from draft to "to be approved". This automatically moves the responsibility of the item from the subject expert to the validator. While an item has the "to be approved" status the subject expert cannot modify it.
- **Delete**: deleting an approved element will assign it the "to be deleted" status. Deleting an element with a draft status will automatically delete and remove the element.

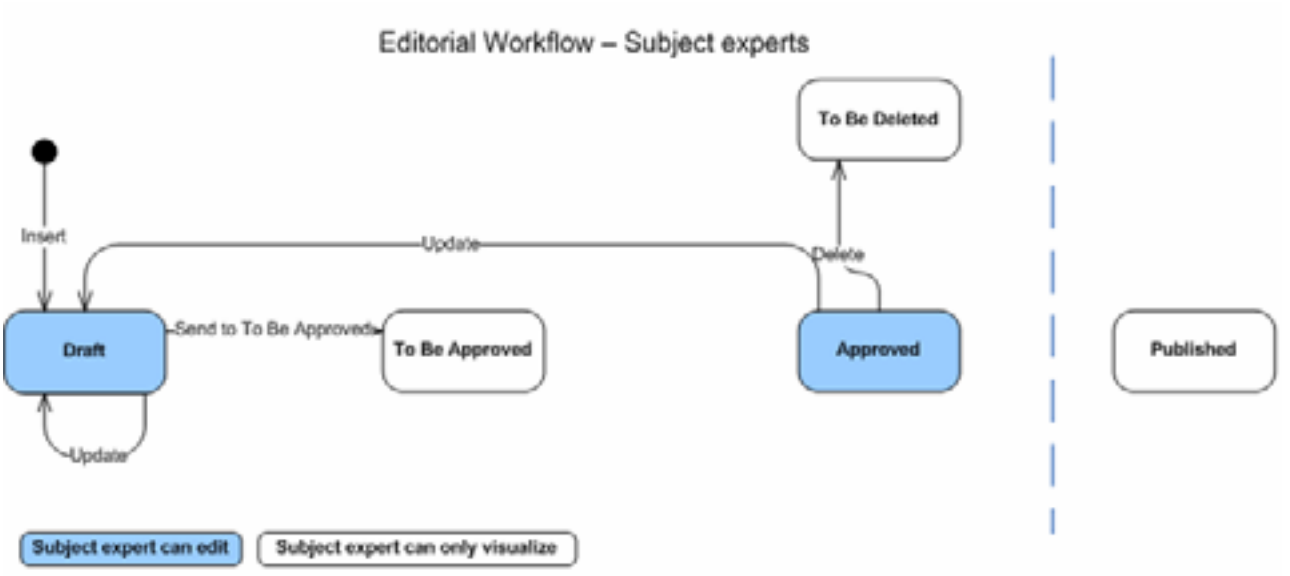


Figure 4. Subject expert allowed actions and sequences.

The workflow for any element starts with a draft version created by a subject expert. A draft is automatically created whenever the subject expert creates a new item or updates an approved item. The responsibility of the item is passed to the validator when the subject expert assigns an element the "to be approved" status. The item will remain with this status until the validator approves or rejects it.

Validators revise, approve or reject changes made by subject experts to the ontology modules and they are the only users who can copy changes into the production environment.

Validators will be able to:

- **Update**: an approved or a "to be approved" element. When a validator makes a modification, it does not need to be approved by others, so the element keeps its status.
- If an element is assigned the "to be approved" status, the validator can either **approve**, **reject** (to draft status), or **modify** it.
- If an element is assigned the approved status, the validator can either **reject** it (returns to the "to be approved" status), **delete** it (sends the element to the "to be deleted" status) or modify it.
- Send an approved element to the "to be deleted" status and even **destroy** an element in the "to be deleted" status.

- A validator can **reject the deletion** of an element that has been assigned the "to be deleted" status. If deletion is rejected, the element returns to the approved status.
- When all the elements of an ontology have been approved, the validator can **publish** it. This copies the approved ontology into the production environment and automatically assigns the correct version (V1 for the first release and Vn+1 for each subsequent release).

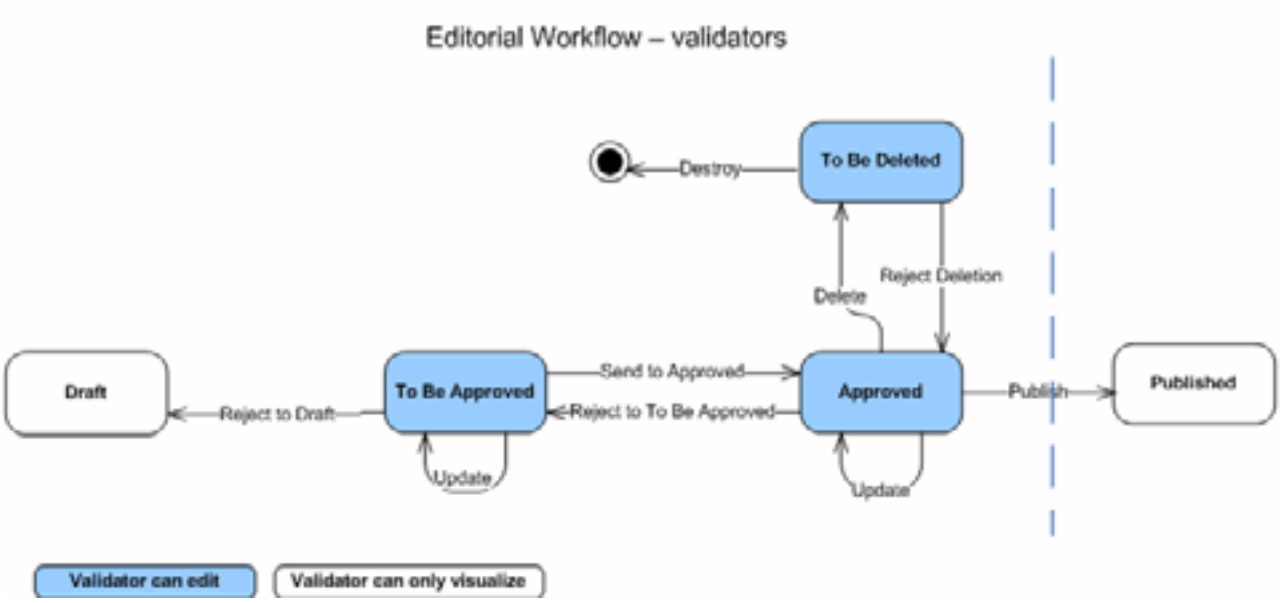


Figure 5. Validator's actions and sequences

2.5.3 Workflow visualization (interface)

It is important to provide a simple intuitive interface for each of the user roles of the editorial workflow, so that tasks are performed efficiently and effectively. Four specific views are required, based on the user roles and element status:

- **Approved** view: approved information only.
- **Draft** view: approved information plus changes made by the current editor, with the difference between the two states clearly visible.
- **To be approved** view: approved information plus all the pending elements to be approved by validators, with the difference between the two states clearly visible
- **To be deleted** view: approved information plus elements that are assigned "to be deleted", with the difference between the two states clearly visible.

3 Use cases

The use cases diagrams and their brief descriptions are inferred mainly from D7.1.1. Full details of each use case is provided in Annex I.

The numbers listed together with the brief descriptions as requirements refer to the corresponding section in the D7.1.1. For example, requirement 4.4.6 refers to Section 4.4.6 in the D 7.1.1.

Analogously, the numbers listed as use cases refer to the corresponding section in the D7.1.1. For example, use case 4.6.12 refers to Section 4.6.12 in the D7.1.1.

Additional information on the activities described in the use cases is provided in the NeOn glossary of activities (D5.3.1).

3.1 Use cases diagrams

The following diagrams are a UML representation of the use cases. There the user goals are reflected (for example “Search”). For each user goal there should be some sub functions that extends it providing more concrete functionalities (for example “Search Using Free Text” and “Search Using Advanced Input”). The same structure is used in the use cases description.

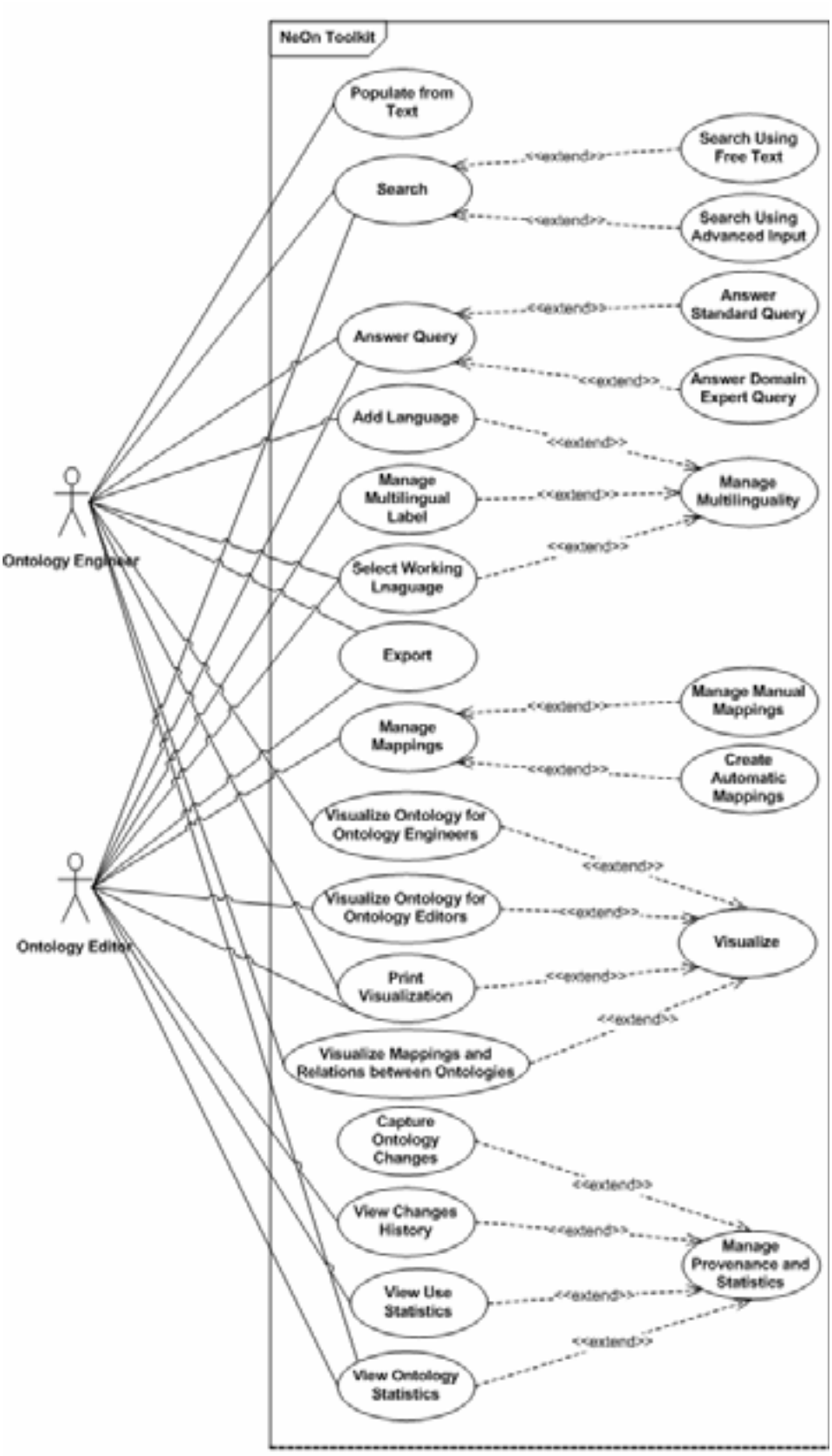


Figure 6. Use case diagram 1/2

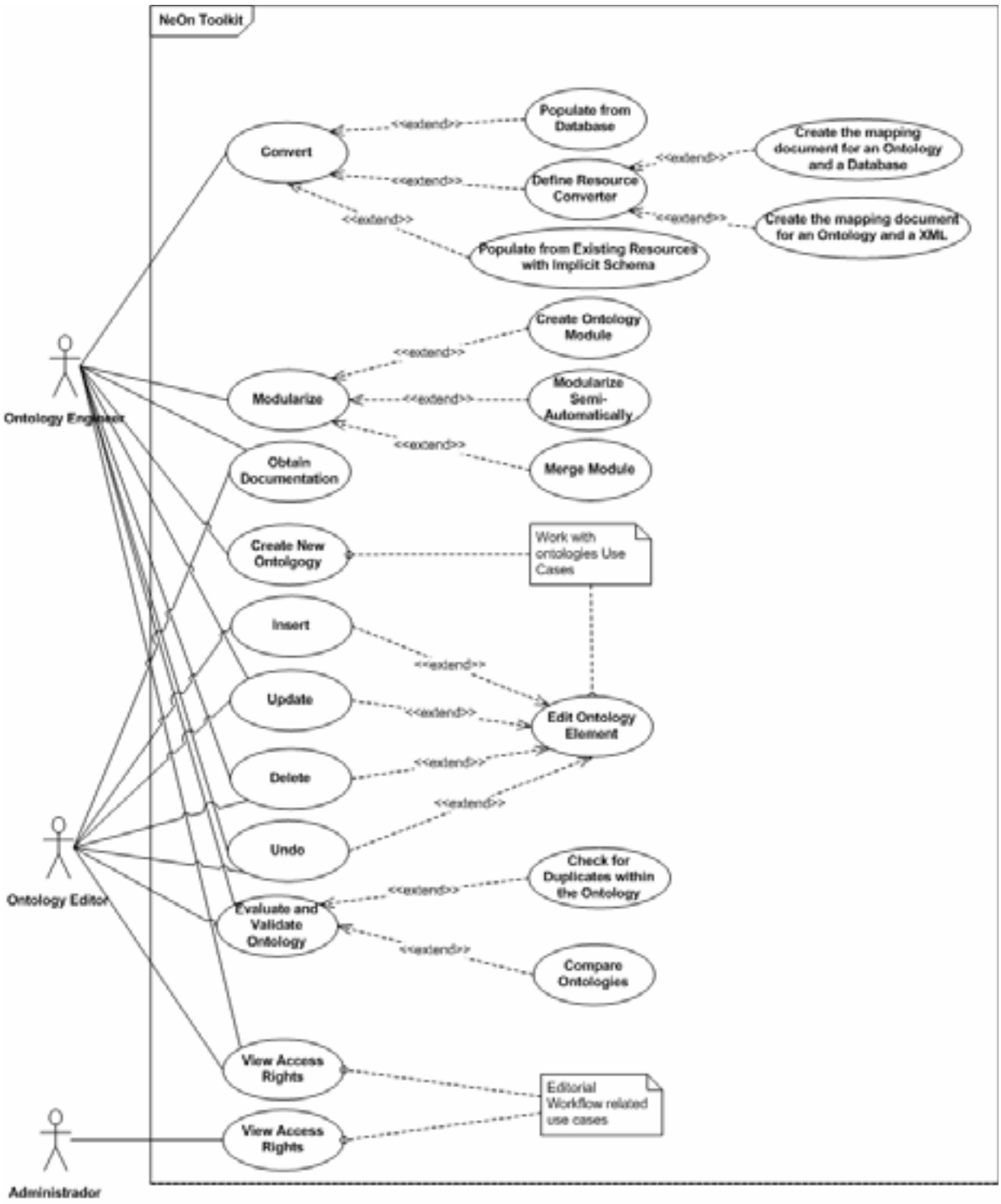


Figure 7. Use case diagram 2/2

3.2 Use cases description

3.2.1 UC-1: Search

Authorized users can perform various types of searches within the ontologies they are working on.

Requirement: 4.4.6

Use case: 4.6.12, 4.6.13, 4.6.14

UC-1.1: Search using free text

The users can search for any ontology element (i.e., concepts, instances, relations, properties) in which a given string appears. The user inputting a string to search and the system returns the search results. The search can be performed across all the ontologies being inspected.

Requirement: 4.4.6

Use case: 4.6.12

UC-1.2: Search using advanced input

Authorized users can search for any ontology element according to their “type”, and/or some constraints, and/or a given string contained in them. Search conditions are set by a template that allows users to impose search constraints. For example, it would be possible to search for:

- concepts having parent and/or that satisfy condition X (and/or Y);
- instances of classes that satisfy condition X;
- properties of data type X; and
- properties having domain X and/or range Y.

Requirement: 4.4.6

Use case: 4.6.13

3.2.2 UC-2: Answer query

While working with ontologies, authorized users are able to perform queries within the ontologies being edited.

Requirement: 4.4.6

Use case: 4.6.13, 4.6.14

UC-2.1: Answer standard query

Authorized users input queries in the formal query languages supported by the system. A graphical interface is available that masks the query code (for example SPARQL).

Requirement: 4.4.6

Use case: 4.6.13, 4.6.14

UC-2.2: Answer domain expert query

Authorized users input queries in natural language.

Requirement: 4.4.6

Use case: N/A

3.2.3 UC-3: Manage multilinguality

The ontology editors and ontology engineers manage the multilingual aspect of the ontologies.

Requirement: 4.4.7

Use case: 4.6.15,

UC-3.1: Add language

The ontology engineers can incorporate a new language for an entire ontology. The ontology engineers specify which elements of the ontology should be multilingual (classes, instances, properties or the entire ontology). The system incorporates the new language into the model of the ontology and the ontology engineers are now able to add translated labels (typically from the FAO's official translation service).

Requirement: 4.6.15

UC-3.2: Manage multilingual label

This use case refers to the ontology localization activity. The system allows ontology editors to add, edit, or delete multilingual labels in individual concepts.

Requirement: 4.4.7

UC-3.3: Select working language

Authorized users can select the language of the ontology information. In addition, ontology editors can choose the language to work with in editing mode while handling translations. More than one language should be displayed to help navigation and editing.

Requirement: 4.4.7

3.2.4 UC-4: Export

All authorized users can export an ontology in multiple formats. The system should be able to export ontologies into thesauri format, with conversion to: TagText, RDBMS, ISO2709, SKOS and TBX.

Requirement: 4.4.5

3.2.5 UC-5: Convert

The ontology engineers can convert ontologies from other resources. The system converts the ontologies from the resources chosen by the ontology engineers.

Requirement: 4.3.1, 4.7.1, 4.7.2

UC-5.1: Convert ontology

The ontology engineers are able to convert ontologies from one ontological format to other.

Requirement: 4.3.1

UC-5.2: Populate from database

This use case refers to the ontology population activity. The ontology engineers populate ontologies from databases. The system allows the ontology engineer to select the ontology to be populated, the source database and the related mapping document (resource converter) for the population. The system populates the ontology from the selected database.

Requirement: 4.3.1, 4.6.4, 4.7.1, 4.7.2

UC-5.3: Populate from existing resources with implicit schema

The ontology engineers populate ontologies from existing resources with implicit schema (XML). The system allows the ontology engineers to select the ontology to be populated, the XML file and the related mapping document (resource converter) for the population. The system populates the ontology from the selected XML file. This use case is also related to the ontology population activity.

Requirement: 4.6.5, 4.7.1, 4.7.2

UC-5.4: Define resource converter

This use case refers to the ontology aligning activity. The ontology engineers create mapping documents (resource converters) that describe the mappings between an external resource and ontologies.

Requirement: 4.7.1, 4.7.2

UC-5.4.1: Create a mapping document between ontologies and databases.

The system allows the ontology engineers to create mapping documents (resource converters) between ontologies and databases. The ontology engineers select the ontology and the database. The ontology engineers browse the schema of the selected ontology and the schema of the selected database. The ontology engineers then select specific elements of the ontology schema and specific elements of the database schema. The ontology engineers define transformations for the selected elements (ontology and database).

Requirement: 4.7.1, 4.7.2

UC-5.4.2: Create a mapping document between ontologies and XMLs.

The system allows the ontology engineers to create mapping documents (resource converters) between XML resources and ontologies. The ontology engineers select the ontology and the XML schema. The ontology engineers browse the schema of the selected ontology and the selected XML schema. The ontology engineers select specific elements of the ontology schema and

specific elements of the XML schema. The ontology engineer defines transformations for the selected elements (ontology and XML).

Requirement: 4.7.1, 4.7.2

3.2.6 UC-6: Create mappings

The ontology editors and ontology engineers may create mappings between ontology elements, i.e., classes, instances and relations inside different ontologies. Any mapping can be either created in a totally manual way or in a semi-automatic way (i.e., system-supported). If a mapping between elements in two ontologies is established, there is a mapping between the two ontologies. If the objects linked by mappings belong to two modules, there is a mapping between these two modules. This use case refers to the ontology aligning activity.

Requirement: 4.4.3

Use case: 4.6.8, 4.6.9, 4.6.10, 4.6.11

UC-6.1: Create mappings manually

The users are able to create mappings between concepts and instances inside different ontologies. The user opens the ontologies to be mapped, chooses the type of mapping to use and selects the ontology elements (or create them if they are not already defined) to map. The system creates and stores the mapping.

Requirement: 4.4.3

Use case: 4.6.8, 4.6.9

UC-6.2: Create mappings semi-automatically

The same mappings described above can be created in a semi-automatic way. The user selects the ontologies and the type of ontology elements to be mapped. She also selects the type of mapping to be created. The system returns the candidate mappings. The user inspects the proposed candidates, selects the appropriate ones and confirms them for creation. The system saves the created mappings.

Requirement: 4.4.3

Use case: 4.6.10, 4.6.11

3.2.7 UC-7: Visualize

Ontology visualization provides users with the possibility of checking their own ontologies while working with them and revising, evaluating and validating ontologies developed by other users.

Ontology engineers are familiar with ontology development and they have been exposed to many existing visualization formats for elements, such as tree view or graphic view.

Ontology editors (subject experts, validators, and viewers), who are in charge of evaluating, validating and publishing ontologies, may have little knowledge of the technical aspects of ontologies. Therefore, a more intuitive visualization is required for these users, compared to ontology engineers.

In addition, ontology editors need clear and simple visualizations of the status of ontology elements in the workflow and hence require functionalities to filter out elements on ontologies (or modules) based on their status and users' roles need to be implemented.

Requirement: 4.4.1, 4.4.3, 4.4.4

Use case: N/A

UC-7.1: Visualize ontology for ontology engineers

Ontology engineers choose the visualization mode that the system is to display the ontology. It is possible to display more than one visualization mode at the same time.

Requirement: 4.4.4

Use case: N/A

UC-7.2: Visualize ontology for ontology editors

Ontology editors can visualize one or more ontologies (modules) in several views, filtered by its status in the editorial workflow.

Requirement: 4.4.4

Use case: N/A

UC-7.3: Visualize mappings between ontologies

Authorized users are able to visualize several ontologies simultaneously, showing the mappings and relationship between the concepts and the modules.

Requirement: 4.4.3, 4.4.4

Use case: N/A

UC-7.4: Print visualization

Authorized users can print a selected visualization.

Requirement: 4.4.4

Use case: N/A

3.2.8 UC-8: Modularize

The ontology engineers can make ontology modules. Ontology modularization refers to the activity of identifying one or more modules in an ontology, with the purpose of supporting reuse or maintenance. A distinction exists between ontology module extraction and ontology partitioning.

Requirement: 4.3.2

Use case: 4.6.7

UC-8.1: Create ontology module

This use case is related to the ontology partitioning activity. The system allows the ontology engineer to create ontology modules. The ontology engineer specifies the branch of the ontology (or a set concepts), a set of properties and a set of instances to be included in the module (and the metadata of the module, name and comments).

Requirement: 4.6.7

UC-8.2: Modularize semi-automatically

This use case is related to the ontology partitioning and to the ontology module extraction activities. The system allows ontology engineers to modularize an ontology in a semi-automatic way. The system returns the resulting candidate modules and the related ontology elements. The ontology engineers select from the proposed modules, those that they want to create, including the ontology elements involved in each module.

Requirement: 4.3.2, 4.6.7

UC-8.3: Merge module

This use case refers to the ontology merging activity. The system allows ontology engineer to merge a set of modules. The system creates a unique module from the original modules.

Requirement: this use case was added during T7.4 analysis of requirements and was not originally included in D7.1.1.

3.2.9 UC-9: Manage provenance and statistics

Authorized users must obtain the provenance and other statistics about the ontologies selected.

Requirement: 4.4.1. 4.7.2 (Provenance service)

Use case: N/A

UC-9.1: Capture ontology changes

The system logs changes in all ontology elements. That is, the date of creation/editing, the editor that made the change, etc. All actions are logged with the following minimum fields:

- description
- operation executed
- timestamp
- URI
- username

Fields will be automatically completed by the system, except for the description field.

Requirement: 4.7.2 (Provenance service)

UC-9.2: View changes history

Ontology editors are able to view the logs about changes to ontology elements (cf. UC-9.1), including the creation/editing date and history notes (to track changes to a concept).

Requirement: 4.7.2 (Provenance service)

UC-9.3: View use statistics

Ontology editors can view important information about an ontology. At least: provenance; previous editors; frequency of changes; and the fragment/domain of the ontology changed most rapidly.

Requirement: 4.7.2

UC-9.4: View ontology statistics

Authorized users can view statistics of the ontology being edited. At least: depth of the class hierarchy; number of child nodes; number of relationships and properties; number of concepts per branch.

Requirement: 4.4.1

3.2.10 UC-10: Populate from text

This use case refers to the ontology population. Ontology engineers can populate an ontology from text. Ontology engineers choose the textual corpora. The system provides ontology engineers with a list of candidate elements of the ontology (classes, instances and relations between concepts). The system shows the documents and excerpts supporting the extracted terminology, including the document metadata such as the title of the document, author, data, owner and publication date. The ontology engineer inspects and selects the appropriate candidate and adds them to the ontology. The system populates the ontology with the new elements.

Requirement: 4.4.2

3.2.11 UC-11: Evaluate and validate ontology

This use case refers to the ontology evaluation activity. The ontology editors and the ontology engineers can check the quality of the ontology under development.

Requirement: 4.4.1

UC-11.1: Check for duplicates within the ontology

This use case refers to the ontology diagnosis and ontology repair activities. The ontology editors and ontology engineers can check for duplicates within the selected ontologies. The system returns duplicates and similar concepts.

Requirement: 4.4.1

UC-11.2: Compare ontologies

This use case refers to the ontology comparison activity. The ontology editors and ontology engineers can compare the ontology being edited with other ontologies. Users select the ontologies for comparison and open them to view. Users can make comparisons by either: searching for specific ontology elements to compare; inspecting the summary statistics of the selected ontologies; or navigating the ontologies.

Requirement: 4.4.1

3.2.12 UC-12: Undo

Ontology engineers and ontology editors are able to undo their last changes.

Requirement: N/A

3.2.13 UC-13: Obtain documentation

Authorized users can generate the documentation about the ontologies and modules they can access. The documentation generated should follow a similar style to those of JavaDoc or OWLdoc. This use case is related to the following activities in the NeOn methodology:

- Ontology documentation
- Ontology summarization

Requirement: 4.1, 4.4.8

3.2.14 UC-14: User rights

UC-14.1: View access rights

Authorized users are able to check their own access rights.

Requirement: 4.4.1

UC-14.2: Manage user rights

Administrators manage the users' rights for accessing modules.

Requirement: 4.4.1

3.2.15 UC-15: Editorial Workflow Related Use Cases

A number of operations are required in order to support the editorial workflow. Users may have different roles in the workflow, and assigned different operations to be performed in different moments. For a detailed description of the editorial workflow required for the ontology lifecycle see Section 2.5 in this deliverable.

Requirement: 4.4.1

Use case:

UC-15.1: Edit Ontology Element

Ontology editors (subject experts and validators) are allowed to edit ontology elements, depending on the status of the elements and their user rights and roles.

Requirement: 4.4.1

Use case: 4.6.3

UC-15.1.1: Insert an ontology element

Subject experts (see Figure 4) are allowed to insert new elements. When subject experts insert a new element, the new element is assigned the Draft status and triggers to start the editorial workflow.

Requirement: N/A

Use case: 4.6.3

UC-15.1.2: Update an ontology element

Subject experts and validators are allowed to update ontology elements, depending on their role and the status of the element: A subject expert can only update elements in Draft or Approved status. In both cases the status of the element is automatically reset by the system to “Draft” status, and the element will need to pass through the whole workflow again to be released. A validator can update elements in “To be approved” and “Approved” status; after the update the element keeps the status it had before the update.

Requirement:

Use case: 4.6.3

UC-15.2: Delete an ontology element

Ontology editors are able to propose for deletion and ontology element that is in the “Approved” status. In any case this is not a definitive action, and the element is automatically assigned the “To be deleted” status. Only validators are able to definitely destroy an element in the “To be deleted” status.

Requirement:

Use case: 4.6.3

UC-15.3: Change status of element

While inserting, updating and deleting elements, the status of those are automatically changed, when required, by the system. There are other cases where a specific action by Ontology editors is required to move an element from one status to another and make the editorial workflow to function. Section 2.5 of this deliverable provides additional information on support of these use cases, which are introduced here below.

Requirement:

Use case:

UC-15.3.1: Send to “To be Approved" status

Subject experts are allowed to select an element in Draft status and sent it to the validator for approval, by changing the status of the element from “Draft” to “To be approved”

UC-15.3.2: Send to the “Approved” status

Validators are allowed to approve elements, and thus to move them from the “To be Approved” status to the “Approved” status.

UC-15.3.3: Reject to “Draft” status.

Validators are allowed to reject element proposed to them to review in the “To be approved” status and send them back to the “Draft” status for the Subject Expert to check, update or complete.

UC-15.3.4: Reject to “To be Approved” status

Validators are allowed to move back to the “To be approved” status an element already approved, if they consider it necessary.

UC-15.3.5: Reject the “To be deleted” proposal for an element

Validators are allowed to reject a proposal for an element’s deletion that are on the “To be deleted” status. In this case the element goes back to the “Approved” status.

UC-15.3.6: Accept the “To be deleted” proposal for an element

Validators are able to accept the deletion and definitely destroy an element in the “To be deleted” status.

UC-15.4: Publish Ontology

Validators are allowed to copy an ontology where all its elements are in the “Approved” status, from the test and validation environment (editorial workflow in the Intranet) to the production environment (Internet). By doing so, the system automatically assigned the right version to the published ontology. From V1 for the first time a particular ontology goes live, to Vn+1 for the N upgrade of the ontology.

UC-15.5: Wiki based Editing of Ontology Elements

Validators are allowed to browse and modify the ontology by adopting the Wiki based visualization interface. The generated portal, representing ontology elements, allows subject experts and validators to perform a community based editing/validation.

4 Architecture

4.1 Introduction to NeOn architecture

In this chapter, the NeOn Toolkit architecture is introduced with references to D6.2.1 developed in WP6. Additionally, uses cases that were introduced in Chapter 3 are mapped to the NeOn Toolkit architecture. Lastly, it is described how the software architecture for the Fisheries Ontologies Lifecycle Management System, will be developed within other deliverables in T7.4.

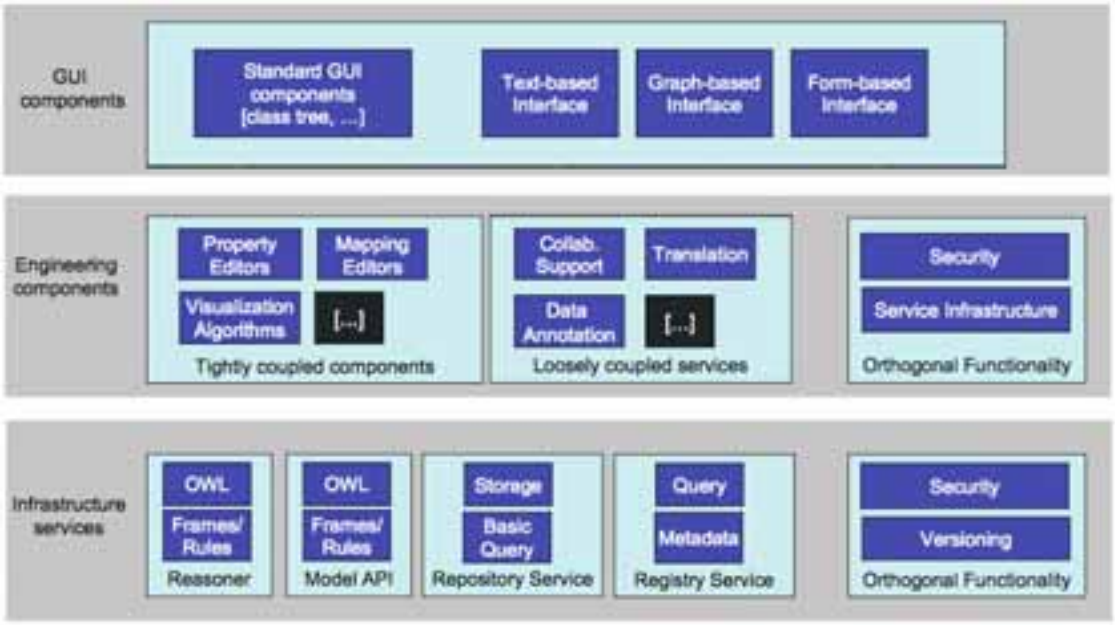


Figure 8. NeOn architecture

There are three layers in the NeOn Toolkit architecture, as displayed in Figure 8:

1. Infrastructure services: in this layer, basic services and functionalities are provided for most components.
2. Engineering components: this middle layer consists of tightly coupled components and loosely coupled services to provide major ontology engineering functionality.
3. GUI components: this layer is aimed directly at users. Both engineering components and infrastructure services may have GUI components. A set of predefined core GUI components also exists.

In NeOn deliverable D6.6.1, the plug-ins that fit in the above architecture has been introduced in the format of Java packages, here we briefly introduced the plug-ins that are related to use cases by properly naming them:

`Core Neon Toolkit – DataModel Plug-in`

This plug-in represents the core plug-in of the NeOn Toolkit since it represents the internal data model and thus the knowledge representation aspect of the application. It provides access to the underlying data model via an API.

`Core Neon Toolkit – GUI Plug-in`

This plug-in contains the core UI components and connects them with the underlying datamodel via the DataModel plug-in. The main UI elements include the ontology navigator and the property editors for the different ontology entities.

`Core Neon Toolkit – Import/Export Plug-in`

Provides functionality to import and export ontologies in different formats from and to the local file system or a WebDAV server (Web-based Distributed Authoring and Versioning).

`Core Neon Toolkit – OntoVisualize Plug-in`

The `ontovisualize` plug-in contains a view to graphically visualize ontologies using the JPowerGraph library[1].

`Core Neon Toolkit – JPowerGraph Plug-in`

Integration of the `jpowergraph` library and some extensions for the visualization plug-in.

`Core Neon Toolkit – Search Plug-in`

This plug-in provides some search functionalities for ontological entities like concepts, attributes, relations and instances.

Alternatively, to enhance the functionality of Core NeOn Toolkit, some plug-ins that are developed together with the above listed plug-ins, such as Query Answering Plug-in, OntoMap, are also provided.

[1] https://sourceforge.net/projects/jpowergraph/

Mapping between use cases and components

This section presents the alignment of the use cases to available tools, highlighting the level of the architecture that they belong to, so we can facility the implementation of the proposed software architecture for the Fisheries Ontologies Lifecycle Management System.

Note:

In the following tables each column denotes:

- **Category**: a use case in terms of user goal.
- **Sub category**: the sub function in the correspondent use case.
- **Tool name**: the name of the tool that will implement the use case sub function.
- **Status**: the status of development of the tool referred before.
- **Contact**: the partner in charge of coordinating the tool development.
- **Place in architecture**: an "I" denotes infrastructure services, "E" denotes engineering components and "G" denotes GUI components.
- **Functionality**: the functionality according to D6.1.1.

Table 1. Search

Sub category	Tool name	Status	Contact	Place in architecture	Functionality
1.1 Search Using Free Text	N/A	Not Implemented	Ontoprise	E G	Query Support
1.2 Search Using Advanced Input	Core NeOn Toolkit – Search Plug-in	Available	Ontoprise	E G	Query Support

Table 2. Answer query

Sub category	Tool name	Status	Contact	Place in architecture	Functionality
2.1 Answer Standard Query	Core NeOn Toolkit – Query Answering Plug-in	Available for Flogic Query	Ontoprise, UKARL	I E G	Query Support
2.2 Answer Domain Expert Query	ORAKEL (Natural language queries)	Available but not integrated	UKARL	I E G	Question Formulation

Table 3. Manage multilingualism

Sub category	Tool name	Status	Contact	Place in architecture	Functionality
3.1 Add Language	Core NeOn Toolkit – GUI Plug-in	Partially available	UPM	E G	Multilingual Support

3.2 Manage Multilingual Label	Term Translator / AGROVOC Web Services	Term Translator available but not integrated	UPM, USFD, FAO	E G	Multilingual Support
3.3 Select Working Languages	Term Translator / AGROVOC Services / NeOn Toolkit Core	Partially available	UPM	I E G	Multilingual Support

Table 4. Export

Sub category	Tool name	Status	Contact	Place in architecture	Functionality
N/A	ODEMapster	Partially available	UPM	E G	Support of heterogeneous knowledge

Table 5. Convert

Sub category	Tool name	Status	Contact	Place in architecture	Functionality
5.1 Convert Ontology	Core NeOn Toolkit – Import/Export Plug-in	Available	Ontoprise	E G	Reuse of ontological resources
5.2 Populate from Database	ODEMapster / R2O	Available	UPM	E G	Reuse of non-ontological resources
5.3 Populate from Existing Resources with Implicit Schema	XMapster / X2O	Not available	UPM	E G	Reuse of non-ontological resources
5.4 Define Resource Converter	ODEMapster / XMapster	Not available	UPM	E G	Reuse of non-ontological resources

Table 6. Manage mappings and integration

Sub category	Tool name	Status	Contact	Place in architecture	Functionality
6.1 Manage Manual Ontology Mapping	OntoMap	Partially available	Ontoprise	E G	Mapping support
6.2 Create Automatic Ontology Mappings	FOAM, Alignment Server	Available	Ontoprise, INRIA	E	Mapping support

Table 7. Visualize

Sub category	Tool name	Status	Contact	Place in architecture	Functionality

7.1 Visualize Ontology for Ontology Engineers	Core NeOn Toolkit – OntoVisualize Plug-in	Available	Ontoprise	E G	Networked ontologies Visualization
7.2 Visualize Ontology for Ontology Editors	Core NeOn Toolkit– GUI Plug-in	Available	Ontoprise	E G	Networked ontologies Visualization
7.3 Visualize Mappings and Relations Between Ontologies	OntoMap	Available	Ontoprise	E G	Networked ontologies Visualization
7.4 Print Visualization	Core NeOn Toolkit – Plug-in not applicable	Not Implemented	Ontoprise	E	Networked ontologies Visualization

Table 8. Modularize

Sub category	Tool name	Status	Contact	Place in architecture	Functionality
8.1 Create Ontology Module	Module Management	Not Implemented	OU	E G	Ontology modularization
8.2 Modularize Semi-automatically	Module Management	Not Implemented	OU	E G	Ontology modularization
8.3 Merge Module	Module Management	Not Implemented	OU	E G	Ontology modularization

Table 9. Use cases from 9 to 13

Category	Sub category	Tool name	Status	Contact	Place in architecture	Functionality
9 Manage Provenance and Statistics	9.1 Capture Ontology Changes	Oyster	Available	UPM	I E	Provenance Support
	9.2 View Changes History	Ontology Evolution Plug-in	Not implemented	UPM, UKARL	E G	Provenance Support
	9.3 View Use Statistics	Ontology Evolution Plug-in	Not implemented	UPM, UKARL	E G	Summarization
	9.4 View Ontology Statistics	Watson / Oyster	Available	OU, UPM	E G	Summarization
10 Populate from Text	N/A	GATE, Text2Onto	Available but not integrated	USFD, UKARL	GATE: E; Text2Onto: I,E	Text mining
11 Evaluate and Validate Ontology	11.1 Check for duplicates within the Ontology	Plug-in not applicable	Not available	UPM, UKARL, OU	E	Summarization
	11.2 Compare / Validate Ontologies	Watson / RADON	Partially available	OU, UKARL	E	Summarization

12 Undo	N/A	Editorial Workflow Plug-in	Not implemented	UPM, UKARL	E G	NeOn Editor
13 Obtain Documentation	N/A	OWLDoc	Not implemented	UPM, FAO	E G	Documentation

Table 10. Use case 14. User rights

Sub category	Tool name	Status	Contact	Place in architecture	Functionality
14.1 View Access Rights	N/A	Not implemented	Ontoprise	E G	Access management to Information resources
14.2 Manage User rights	N/A	Not implemented	Ontoprise	E G	Access management to Information resources

Table 11. Use case 15. Editorial workflow related use cases

Sub category	Tool name	Status	Contact	Place in architecture	Functionality
15.1 Edit Ontology Element (Insert, Update, Delete)	Editorial Workflow Plug-in	Not available	UKARL, UPM	E G	Ontology collaborative development
15.2 Delete	Editorial Workflow Plug-in	Not available	UKARL, UPM	E G	Ontology collaborative development
15.3 Change Status of Element	Editorial Workflow Plug-in	Not available	UKARL, UPM	E G	Ontology collaborative development

4.2 The integrated software architecture

The integrated software architecture for the fisheries ontology lifecycle below has been designed (according to D6.2.1) as an instantiation of WP6 architecture. For each sub category in each use case the place in the architecture of the components involved have been provided.

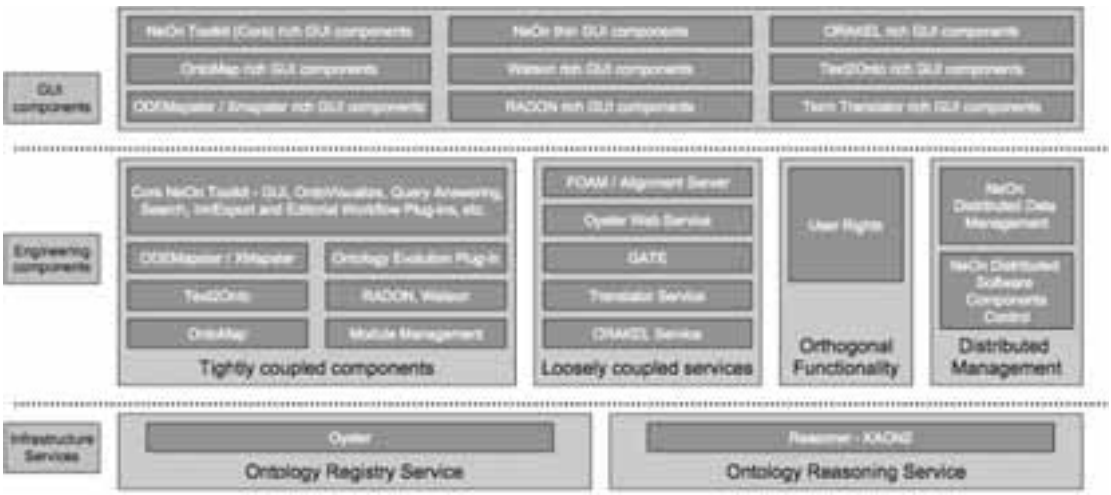


Figure 9. The integrated software architecture

Based on this architecture, taking the use case of editorial workflow introduced in Chapters 2 and 3 for example, the ontology editor may access the following engineering components in the architecture:

- Core NeOn Toolkit
- OntoMap
- Oyster Web Service
- Text2Onto
- Translator Service

The ontology editor may use Core NeOn Toolkit to bootstrap ontologies, OntoMap to create ontology mappings, Translator Service to translate ontologies into different languages, Oyster Web Service to access remote ontologies and Text2Onto to populate ontologies from text.

Another example could come from the task of ontology validator. If an ontology validator wants to check an ontology, the following components may be used:

- Core NeOn Toolkit
- Oyster
- RADON

The NeOn Toolkit will be used to visualize ontologies, Oyster to check the ontology history and RADON will provide advanced functionalities for checking ontology consistency.

5 Plans for next steps

Task T7.4 "Software for managing the fishery ontologies lifecycle" brings together a number of related deliverables (within WP7). These deliverables are necessary to design, test and deploy a system that enables ontology engineers and editors in the fishery domain, to manage the ontology's lifecycle, using NeOn technologies and methodologies.

A staged approach will be used for task T7.4, due to the:

- timing of the development of the NeOn Toolkit and its engineering components;
- fisheries system's dependence on the NeOn architecture;
- timing of the WP7 case study;
- vital importance of this task for the successful completion of the case study objectives; and
- need to incorporate users' feedback during the implementation of the prototypes.

The staged approach facilitates the development of the Fisheries Ontologies Lifecycle Management System through:

- a number of iterations of the design, implementation, testing and feedback;
- producing incremental prototypes;
- enabling users to be introduced to the NeOn Toolkit as it develops;
- providing users with NeOn tools for task T7.2 "Enhanced Fisheries Networked Ontologies"; and
- gathering early recommendations and feedback from these users.

T7.4 will produce, at least, three incremental and iterative deliverables:

1. D7.4.1 Software architecture for managing the fishery ontologies lifecycle (by M18);
2. D7.4.2 Prototype system for managing the fishery ontologies lifecycle and based on the NeOn Toolkit and tools provided by WP1-WP4 (by M24); and
3. D7.4.3 Integration of selected ontology learning and population tools in the fisheries ontologies lifecycle system (by M30).

Additionally, this task incorporates input from various deliverables in WP7 and deliverables produced in other work packages, mainly WP6, as illustrated in Figure 10.

5.1 T7.4 plan

5.1.1 D7.4.1 at M18

D7.4.1 "Software architecture for managing the fishery ontologies lifecycle" is the first deliverable of T7.4. It provides the architecture of the system, designed as an instantiation of the NeOn architecture and tailored to the fisheries domain. This deliverable reflects the initial user requirements for the Fisheries Ontologies Lifecycle Management System as summarized in D7.1.1 "User requirements specifications for the fishery ontologies lifecycle, knowledge tools and alert system".

D7.4.1, references D6.2.1 "Specification of NeOn reference architecture and NeOn APIs" and D6.3.1 "First implementation of critical infrastructure components".

With regard to other important deliverables related to D7.4.1, work has been coordinated with WP6 to gain input from early drafts of D6.4.1 "Realization & early evaluation of NeOn basic infrastructure services consisting of reasoner, repository and registry" and D6.7.1 "Realization of the basic NeOn Toolkit". Both have the same delivery date as D7.4.1 and are strongly related to each other.

Regarding the technical work packages WP1 to WP4 (they produce NeOn engineering components) information will not be collated into D6.10.1 "Realization & evaluation of core engineering components for NeOn Toolkit" until M24. Work has been coordinated with respective work packages or partners to align specifications of plug-ins with the architecture in D7.4.1.

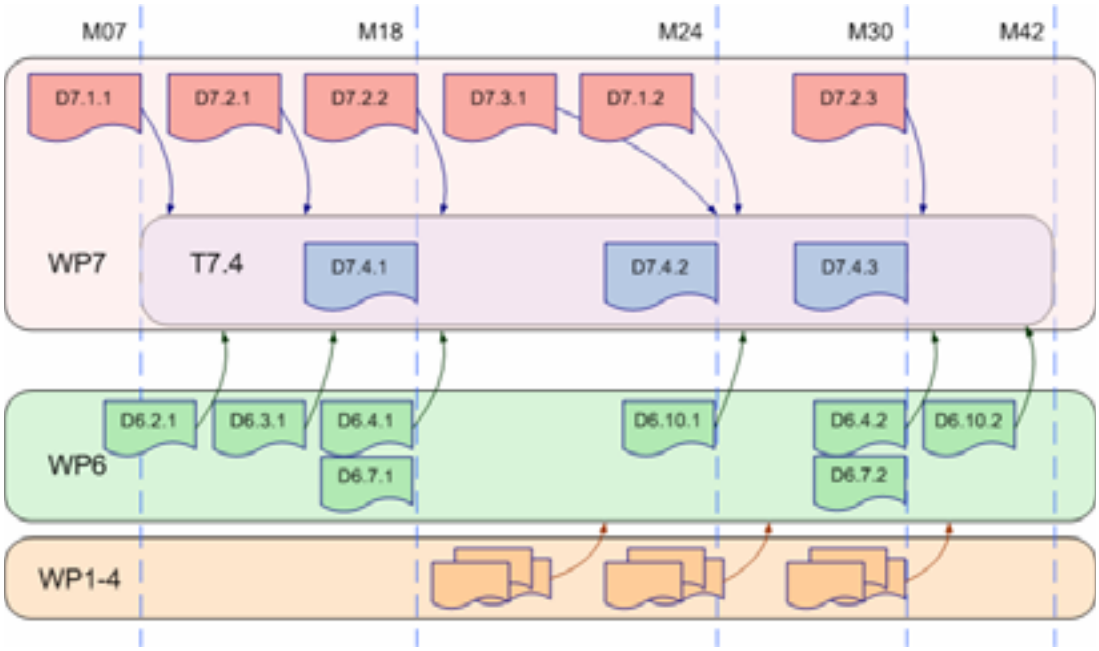


Figure 10. Plan for next steps

5.1.2 D7.4.2 at M24

D7.4.2 will be the first iteration prototype based on D7.4.1. This prototype will select a set of components from the NeOn architecture when available within M18 and M24 and will integrate various plug-ins developed by the technical work packages WP1 to WP4, that will be available before M24.

Alternatively, to support ontology development activities and the maintenance and mapping in T7.2, WP7 has prioritized the critical functionality to proceed with this task working with the large fisheries ontologies. Therefore, the first prototype will also focus on integrating the engineering components that support prioritized use cases.

In particular, the following functions have been identified as the most important for the continued improvement of the fisheries ontologies and to initiate the mappings:

- Browsing, visualization and editing functions. These are particularly important when working with large and rich ontologies. In addition, advanced print functionalities are highly desirable.
- Functions to convert, migrate and map legacy systems (mainly relational databases), into an ontological format.

- Support for modularization, to facilitate work with consistent chunks of ontologies. Again, this is important when working with large ontologies, such as the fisheries domain.
- Multilingual support (to manage the fisheries ontologies), which need to support (at least) English, French, Spanish, Arabic and Chinese.

Based on the criteria above and on the availability of components and priorities for the case study, the following use cases will be included in the first prototype:

- UC 1.2 Search using advanced input
- UC 2.1, 2.2, Query answering
- UC 5, Convert
- UC 6, Managing mapping
- UC 7, Visualize
- UC 9, Manage provenance and statistics
- UC 11, Evaluate and validate ontology
- UC 12 Undo
- UC 3, Manage multilingualism
- UC 4, Export
- UC 8, Modularize
- UC 15, Editorial Workflow related use cases

5.1.3 D7.4.3 at M30

D7.4.3 will be the second prototype of the Fisheries Ontologies Lifecycle Management System. It will integrate the engineering components covering the rest of use cases not included in the first prototype and made available by partners during the period M24-M30. Based on the results of the experiments on ontology learning and population carried out within T7.3 and evaluated in D7.3.1 (M20), if the tools and methods are found worthwhile to assist on the networked ontologies maintenance, the tools will be integrated into the lifecycle management system.

5.2 Detailed architecture for first prototype (D7.4.2)

According to the schedule of software development in T7.4, various iterative and incremental prototypes will be deployed for the Fisheries Ontologies Lifecycle Management System. In the first prototype, the following software components are planned to be integrated:

- Core NeOn Toolkit
- OntoMap
- ODEMapster / R2O
- FOAM
- Oyster
- Watson
- KAON2

Annex I has details on how these software components match the use cases selected for the first prototype, as indicated in section 5.1.2 of this document

The selected software components will be part of the architecture of the first prototype, as shown in Figure 11. This architecture is derived from the general NeOn Toolkit architecture produced by WP6, and introduced in this document in Chapter 4.

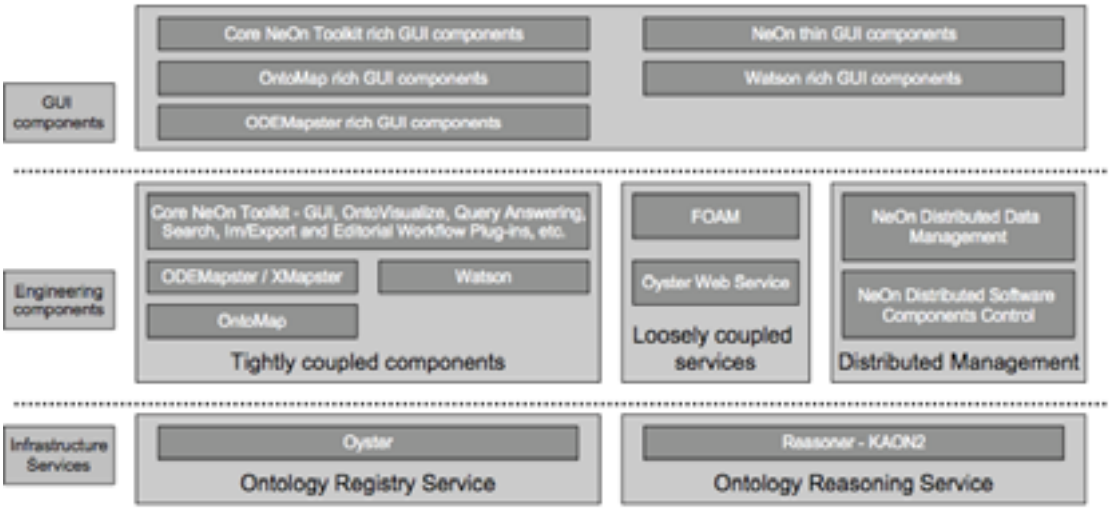


Figure 11. Detailed architecture of the first prototype

6 Conclusions and next steps

This deliverable describes the software architecture for the system that will manage the fishery ontologies lifecycle.

To design this software architecture, various deliverables were considered, such as

- D7.1.1, presenting WP7 user requirements;
- D7.2.1, listing the inventory of resources that will be used in the case study;
- D7.2.2, describing the first set of ontologies for the case study and highlighting the lessons learned and issues about developing those with the current state of the art tools; and
- D6.1.1, D6.2.1, D6.3.1, D6.4.1 and D6.7.1, providing the NeOn Toolkit and general architecture.

The use cases (Chapter 3 and Annex I) and the proposed architecture (Chapter 4) show that by integrating engineering components into the core NeOn Toolkit, it is possible to support ontology engineers developing single, small-to-medium sized ontologies. In addition, this document shows that the proposed architecture can support a real-life application, such as the one in the WP7 case study, with many data sources and large networked ontologies.

This deliverable focuses on the requirements and derived use cases (and their subsequent components – when identified) necessary to manage the mapping or upgrade of legacy systems to ontologies. That is; the editorial workflow for enabling users to modify and update ontologies in a controlled and coherent manner, especially for those ontologies already deployed on the Internet and thus being used by systems (semantic web applications).

Supporting the use cases and architecture and in order to depict and help understanding the whole lifecycle, this deliverable presents (Chapter 2) the set of steps ontologies need to pass through, from their conceptualization, development, validation, release and maintenance; paying special attention to the users involved in the ontology lifecycle, based on their knowledge and skills as well as the roles they will have in each particular step of the lifecycle.

It was decided to follow a phased approach for the deployment of the system through various incremental prototypes as described in Chapter 6, due to the;

- complexity of the case study;
- tight timeline of the WP7 case study;
- strong inter-linkage of the various work packages within the NeOn project; and
- necessity of continuously incorporating user feedback.

The first prototype D7.4.2 will be available by M24, and will allow users in FAO to provide early feedback to the NeOn project on various aspects of the toolkit.

References

FAO07 Food and Agriculture Organization of the United Nations (FAO), 2007, http://www.fao.org, last accessed the 13/08/2007

FIGIS07 Fisheries Global Information System (FIGIS), 2007, http://www.fao.org/fi/website/FIRetrieveAction.do?dom=topic&fid=2017, last accessed the 13/08/2007

D1.1.1 Haase, P., Rudolph, S., Wang, Y., 2006, D1.1.1 Networked Ontology Mode. NeOn Project Deliverable. http://www.neon-project.org/web-content/index.php?option=com_weblinks&catid=17&Itemid=35

D5.3.1 NeOn Glossary of Activities

D7.1.1 Iglesias, M., Caracciolo, C., Jacques, Y., Sini, M., Calderini, F., Keizer, J. Ward, F., Nissim, M., 2006, D7.1.1 WP7 User requirements. NeOn Project Deliverable. http://www.neon-project.org/web-content/index.php?option=com_weblinks&catid=17&Itemid=35

Larman05 Larman, C., 2005, Applying UML and patterns. An Introduction to Object-Oriented Analysis and Design and Iterative Development, [ed.] Don O'Hagan. Third. Upper Saddle River : Prentice Hall, Vol. 1. ISBN 0-13-148906-2.

RTMS07 Reference Tables Management System, 2007, http://www.fao.org/figis/servlet/RefServlet, last accessed the 13/08/2007.

D6.1.1 Gómez, J, Buil, C, Muñoz O, Waterfeld, W, 2006, Requirements on NeOn Architecture, NeOn Project Deliverable.

D6.2.1 Walter, W., Weiten, M., Haase, P., 2007, D6.2.1 Specification of NeOn reference architecture and NeOn APIs. NeOn Project Deliverable. http://www.neon-project.org/web-content/index.php?option=com_weblinks&catid=17&Itemid=35

D6.3.1 Weiten, M., Erdmann, M., 2007, D6.3.1 First Implementation of critical Infrastructure Components. NeOn Project Deliverable. http://www.neon-project.org/web-content/index.php?option=com_weblinks&catid=17&Itemid=35

Acronyms and definitions used in this deliverable

A-BOX	Asserted components - – a fact associated with a terminological vocabulary within a knowledge base
AGROVOC	FAO's Multilingual Agricultural Thesaurus
D1.1.1	Networked ontology model
D6.1.1	Requirements on NeOn Architecture
D6.2.1	Specification of NeOn reference architecture and NeOn APIs
D6.3.1	First Implementation of critical Infrastructure Components
D7.1.1	User requirements specifications for the fishery ontologies lifecycle, knowledge tools and alert system
D7.4.1	Software architecture for managing the Fisheries ontologies lifecycle
D7.5.1	Software architecture for the ontology-based Fisheries Stock Depletion Assessment System (FSDAS)
D7.X.X	Deliverable in WP7
DAML+OIL	DARPA Agent Markup Language + and Ontology Inference Layer
DB	Database
DMOZ	Directory Mozilla or Open Directory Project
FAO	Food and Agriculture Organization of the United Nations
FIGIS	Fisheries Global Information System
F-LOGIC	Frame Logic
FOAM	Framework for Ontology Alignment and Mapping
FSDAS	Fisheries Stock Depletion Assessment System
GUI	Graphical User Interface
I/O	Input and Output
ISO	International Organization for Standardization
ISO2709	ISO standard on Information and documentation: Format for Information Exchange
JavaDoc/OwlDoc	Description of java source codes or OWL elements
KAON2	An infrastructure for managing OWL DL, SWRL, and F-Logic

	ontologies
LSA	Latent Semantic Analysis
MySQL	My Structured Query Language (RDBMS)
NeOn	Network of contextualized ontologies
ODEMapster	Ontology Development Environment Mapster
OMV	Ontology Metadata Vocabulary
OntoMap	Ontology mapping
ORAKEL	natural language interface for knowledge bases
OWL	Web Ontology Language
OWL DL	Description logic version of Web Ontology Language
OWL MetamodelOWL DL	The meta-model for representing OWL in the context of the NeOn Toolkit Description logic version of Web Ontology Language
OWLDocOWL Metamodel	Description of OWL elements. The meta-model for representing OWL in the context of the NeOn toolkit
P2P	Peer to Peer
R2O	Relational to Ontology
RADON	Repair and Diagnosis for Ontology Networks
RDBMS	Relational Database Management System
RDF	Resource Description Framework Schema
RDF-S	Resource Description Framework Schema
RTMS	Fisheries Reference Table Management System
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SSD	System Sequence Diagram
T7.4	Software for managing the fishery ontologies lifecycle
T7.X	Task in WP7
TBX	TermBase eXchange
UC	Use Case

URI	Universal Resource Identifier
WP	Work Package
WP1	Dynamics of networked ontologies
WP2	Collaborative aspects for networked ontologies
WP3	Context sensitivity for networked ontologies
WP4	Human-ontology interaction
WP7	Ontology-driven stock depletion assessment system
X2O	XML to Ontology
XML	Extensible Markup Language

Annex

UC-1 Search

Overview

For a general overview please refer to section 3.2.1. This set of use cases will be mainly provided by the NeOn toolkit.

Detailed Description

UC-1 Search

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Stakeholders and Interests:

Authorized users want to retrieve certain entities within the ontologies

Preconditions: Some ontologies are opened in the toolkit.

Success Guarantee: Search results are returned to the user.

Fail Guarantee: No search results are returned to the user.

Extensions Points: Extended by:

- UC-1.1: Search using free text
- UC-1.2: Search using advanced input

Main Success Scenario: UC-1.1

Extensions: UC-1.2

Frequency of Occurrence: High

UC-1.1 Search Using Free Text

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Stakeholders and Interests:

Authorized users want to retrieve entities (concepts, properties, instances, or attribute values) within an ontology given a simple search-string.

Preconditions: Some ontologies are opened in the toolkit.

Success Guarantee: Search results are returned to the user.

Fail Guarantee: No search results are returned to the user.

Extensions Points: N/A

Main Success Scenario:

1. User opens search dialog
2. User enters search string
3. User initiates search

4. System presents a search result view containing a list of matches
5. User clicks on one of the matches
6. System opens selected element in the ontology navigator

Frequency of Occurrence: High

UC-1.2 Search Using Advanced Input

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Stakeholders and Interests:

Authorized users want to retrieve ontology elements within an ontology given some complex search expression.

Preconditions: Some ontologies are opened in the toolkit.

Success Guarantee: Search results are returned to the user.

Fail Guarantee: No search results are returned to the user.

Extensions Points: N/A

Main Success Scenario:

1. User opens search dialog
2. User clicks on “advanced search”
3. User selects the type of ontology element she wants to search
4. User enters search expressions
5. User initiates search
6. System presents a search result view containing a list of matches
7. User clicks on one of the matches
8. System opens appropriate entity in the ontology navigator

Frequency of Occurrence: Medium

Plugins/Tools

Both use cases UC-1.1 and UC-1.2 are partially implemented in the NeOn Toolkit already. The screenshots below show the current search dialog and the presentation of search results (cf. “D6.7.1 Documentation of NeOn Toolkit” for more details).

The available search-functionality of the NeOn Toolkit fully supports UC-1.1 with one exception. Currently attribute values are not considered for matching with the entered search-term.

The NeOn Toolkit supports UC1.2 to some extend, too. With it users can restrict the search to certain kinds of ontology elements, i.e. concepts, properties, or instances. Also the scope of search can be selected. In the future the Toolkit should support restricting the search to single ontologies. Other more complex search requests can be partly realized with a query tool, which supports complex queries against the ontology.

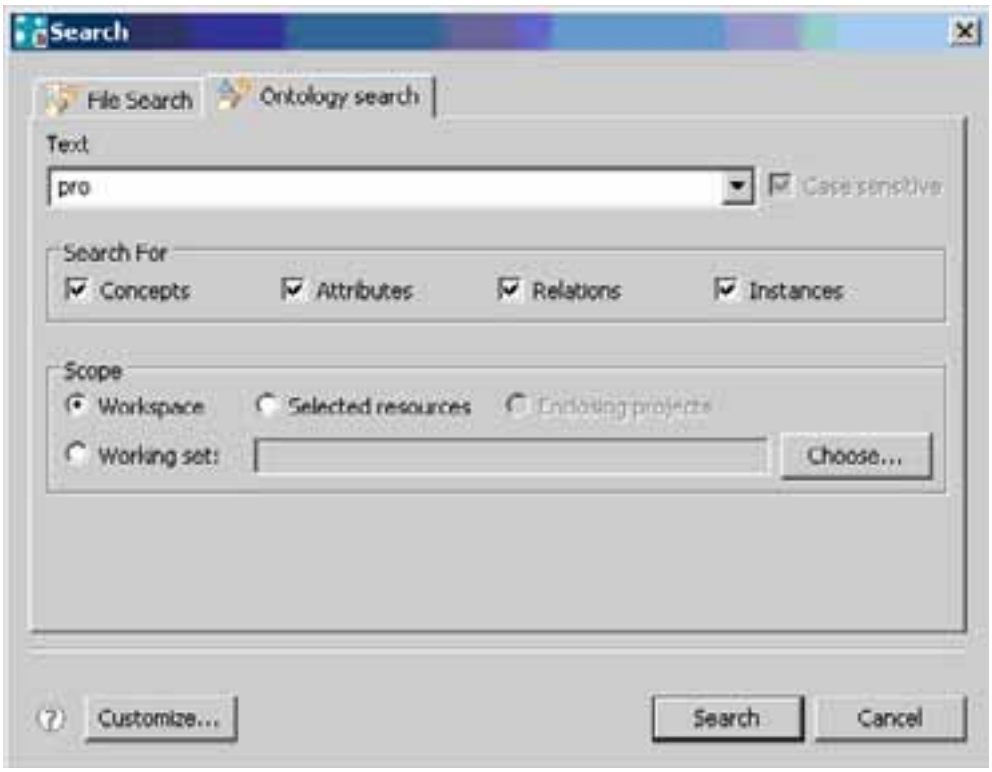


Figure 12. Current Search Dialog of the NeOn Toolkit

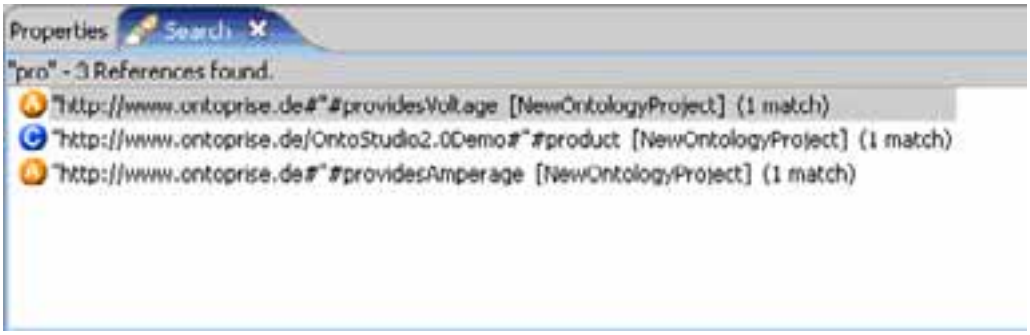


Figure 13. Search result view, displaying matching entities

UC-2 Answer query

These use cases are described in Section 3.2.2 in the main document.

Overview

While working with ontologies, authorized users are able to perform queries within the ontologies being edited. UC-2.1 deals with queries input with a formal query language (a graphical interface is available for users who do not want to write queries in a formal language), UC-2.2 with queries input with natural language.

The NeOn Toolkit (Core) will implement UC-2.1 and ORAKEL, a component integrated into NeOn Toolkit, will implement UC-2.2.

Detailed description

UC-2 Answer query

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Primary Actor: Authorized user

Stakeholders and interests:

Authorized user wants to perform a query within the ontology being inspected.

Preconditions: The user is logged in and has the permissions for viewing the ontologies.

Success Guarantee: Query results are returned to the user.

Fail Guarantee:

No query results are returned to the user. Appropriate warning and/or error message is returned.

Extensions Points: Extended by:

- UC-2.1: Answer Standard Query
- UC-2.2: Answer Domain Expert Query

Main Success Scenario:

1. The user selects an ontology for querying.
2. The user chooses to make a standard query (UC-2.1) or a domain expert query (UC-2.2).
3. Results are returned to the user.

Special Requirements: N/A

Frequency of Occurrence: medium

UC-2.1 Answer standard query

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Authorized user

Stakeholders and Interests:

Authorized user wants to perform a standard query to the ontology being inspected.

Preconditions: The user is logged in and has the permissions for viewing the ontologies.

Success Guarantee: Query results are returned to the user.

Fail Guarantee:

Message is returned to the user to warn that no results are found. In case of an error in the query, an appropriate error message should be given.

Extensions Points: N/A

Main Success Scenario:

1. The user introduces a formal query string, either manually or by means of a GUI.
2. The System performs the reasoning task according to the query and presents the answer to the user.

Special Requirements: N/A

Frequency of Occurrence: medium

UC-2.2 Answer domain expert query

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Authorized user

Stakeholders and Interests:

Authorized user wants to make a natural language query to the ontology being inspected.

Success Guarantee: Query results are returned to the user.

Fail Guarantee: No query results are returned to the user.

Extensions Points: N/A

Main Success Scenario:

1. The user introduces a query formulated in natural language.
2. The System performs the reasoning task according to the query and presents the answer to the user.

Special Requirements:

The query formulated in natural language is restricted only for ground facts as typically found in the ontology as answers, such as questions starting with “who”, “what”, “where”, “which”, “How many” etc., but not for explanations, such as questions starting with “Why”, “How”.

Frequency of Occurrence: medium

Plugins/tools

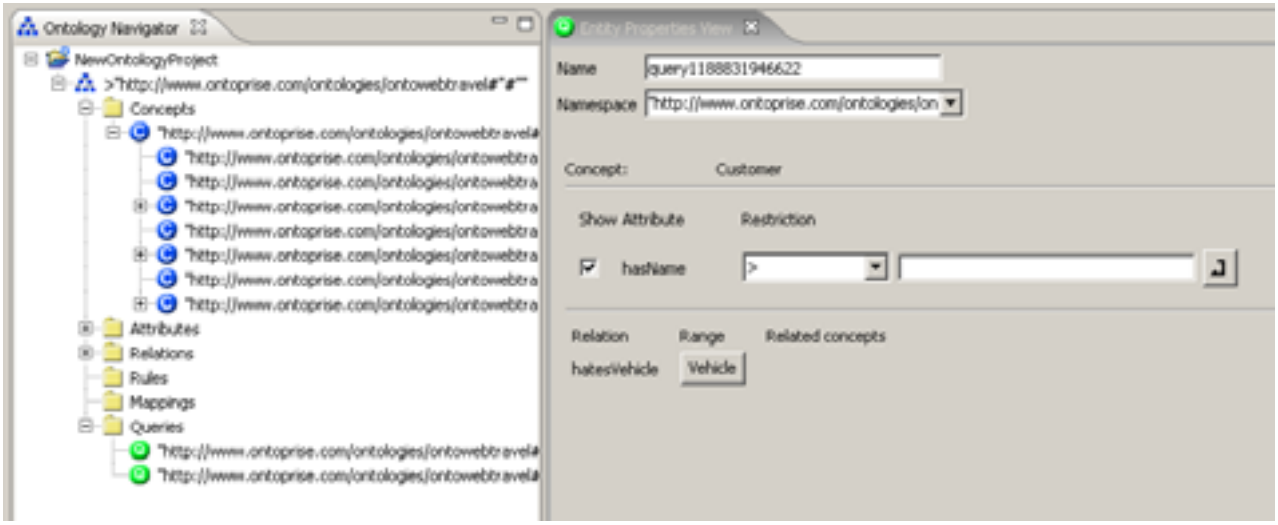


Figure 14. GUI prototype of standard query answering

The GUI prototype of standard query answering interface consists of an entity of “Query” and its corresponding property view. Users are able to edit a query based on the attributes of the querying target. The GUI prototype is show in the figure above.

ORAKEL is a natural language interface which translates natural language queries to structured queries. This translation relies on a lexicon for the underlying fishery ontology, which specifies the possible lexical representations of the ontology elements in the user queries. ORAKEL generates the lexicon partially automatically from the underlying ontology. The lexicon can be refined manually with appropriate tool support

From the user’s view, they are able to directly interact with FAO fishery department portal, by accessing the library data with natural language questions, which are translated into SPARQL queries by a component called ORAKEL. The underlying mechanism however is hidden from the users – the only thing user need to do is to input the query just as their normal questions and then get the result from the portal.

From the view of usability and human factor engineering, this interface has the big advantage of bringing the user out of the game of guessing and trying the keywords in the entry of the webpage portal. Obviously, most people have the experience of struggling with the keywords of the query, especially when their searching target is uncertain. This interface enables users to query the data by the relations among them without knowing any keyword included in the data.

As natural language interface for domain expert query, the development of ORAKEL is ongoing. We propose ORAKEL here as a future direction and plug-in of NeOn toolkit.

UC-3 Manage Multilinguality

Overview

For a general overview of these use cases please refer to section 3.2.3. The NeOn Toolkit (Core) will implement the use case UC-3.1 and TermTranslator will implement the use cases UC-3.2. and UC-3.3.

Detailed description

UC-3 Manage multilinguality

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Primary Actor: Authorized user

Stakeholders and Interests:

Authorized user wants to work with linguistic information of the ontologies.

Preconditions: The user is logged in and has the permissions for work with the ontologies.

Success Guarantee:

The System is configured for work with linguistic information of the ontologies.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions Points: Extended by:

- UC-3.1: Add Language
- UC-3.2: Manage Multilingual Label
- UC-3.3: Select Working Languages

Main Success Scenario:

1. The user incorporates a new language for work with the linguistic information of the ontologies (UC-3.1).
2. The user chooses an individual concept on which the multilingual information will be managed (UC-3.2).

Extensions:

(In UC-3.1 above) the user chooses the languages for view the information and selects a language for work in editing mode (UC-3.3).

Special Requirements:

Frequency of Occurrence: Medium

UC-3.1 Add language

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and Interests:

Ontology Engineer wants to incorporate a new language to work with the linguistic information of the ontologies.

Preconditions: The user is logged in and has the permissions for add a new language.

Success Guarantee:

A new language is created for manage the linguistic information of the ontologies.

Fail Guarantee: The System remains as it was before the execution of this use case.

Extensions Points: N/A

Main Success Scenario:

1. The user selects the new language to add.
2. The user chooses that elements of the ontology must be multilingual.
3. The System configures the editors in order to incorporate the new language.

Extensions: N/A

Special Requirements: N/A

Frequency of Occurrence: Low

UC-3.2 Manage multilingual label

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Editor

Stakeholders and Interests:

Ontology Editor wants manage the multilingual information of an individual concept.

Success Guarantee:

The System updates the multilingual information of the concept selected by the user.

Fail Guarantee: The System remains as it was before the execution of this use case.

Extensions Points: N/A

Main Success Scenario:

1. The user selects the ontology label that he/she wants to manage.
2. The user introduces the linguistic information in the corresponding target language.
3. The System updates the multilingual information of the label under consideration.

Extensions:

(In 2 above) the user wants to edit/to add the linguistic information, he/she can use TermTranslator for looking for the relevant information in the lexical resources that it has: EWN MySql Databases, BabelFish, GoogleTranslate, IATE, Wiktionary, FreeTranslation and Agrovoc Web Services.

TermTranslator ranks the obtained linguistic information and returns to the user the results.

TermTranslator updates the multilingual information.

(In 2 above) the user wants to delete the multilingual information.

Special Requirements: N/A

Frequency of Occurrence: Medium

UC-3.3 Select working language

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Authorized user

Stakeholders and Interests:

Authorized user wants select the languages to show the linguistic information and work in editing mode.

Success Guarantee: The System configures the editors in order to work with some languages.

Fail Guarantee: The System remains as it was before the execution of this use case.

Extensions Points: N/A

Main Success Scenario:

1. The user selects the languages to view the ontology information.
2. The user chooses the working language to translate an ontology label.
3. The System configures the editors to work with some languages.

Extensions: N/A

Special Requirements: N/A

Frequency of Occurrence: Medium

Plugins / Tools

TermTranslator supports the translation of ontological terms using relevant information obtained from different lexical resources. More details about of the functionalities that characterizing the current 1st prototype and the innovations planned for the 2nd prototype are described in the Deliverable 2.4.1 Section 1st Prototype of the NeOn Multilingual Ontology Meta-model.

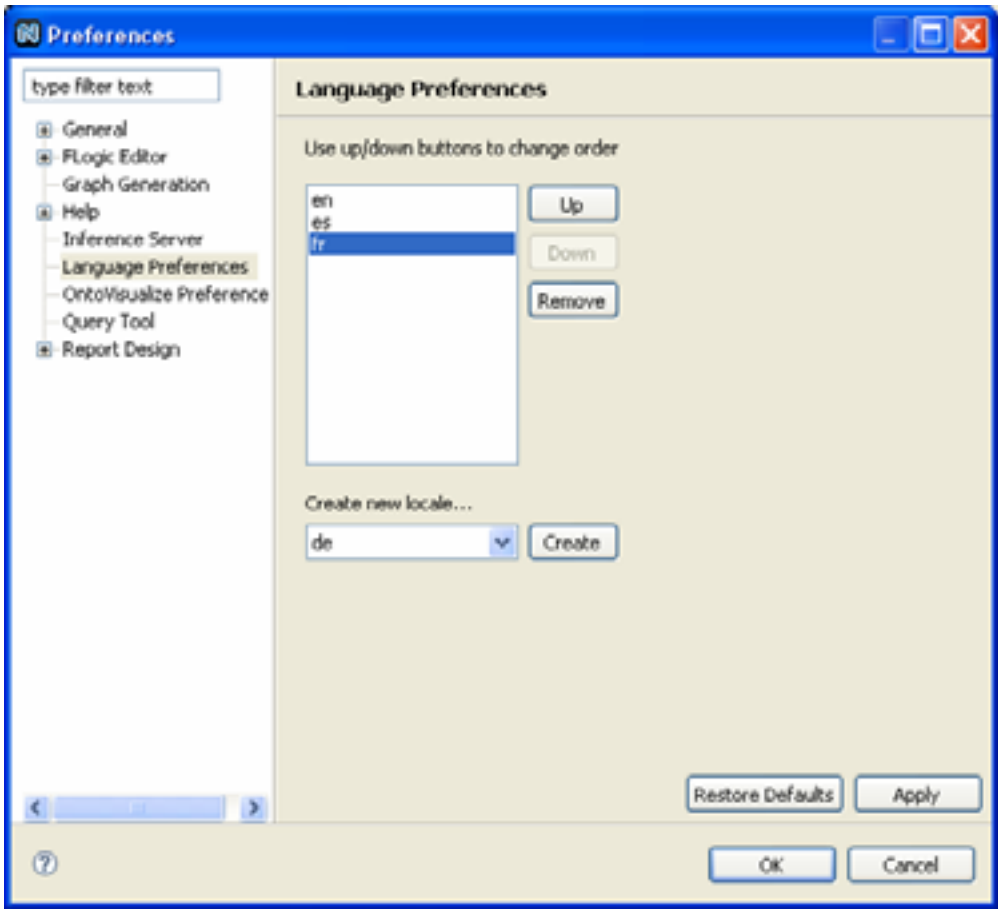


Figure 15. Add Language GUI Prototype

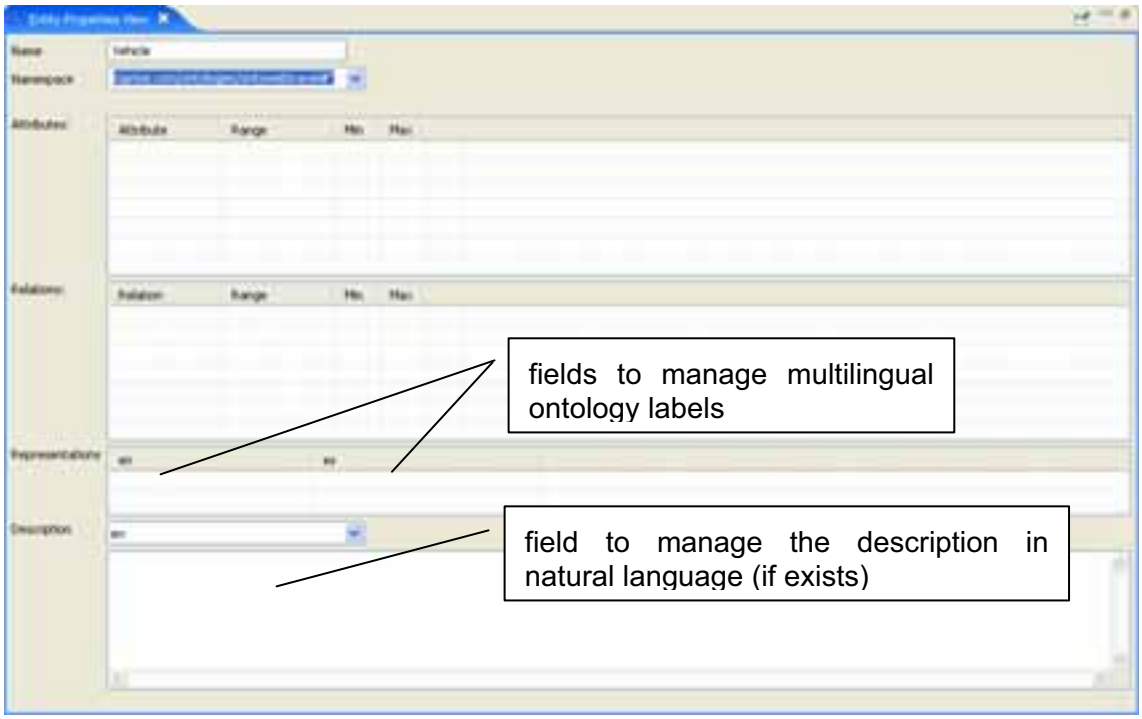


Figure 16. Entity Properties View for manage multilingual information

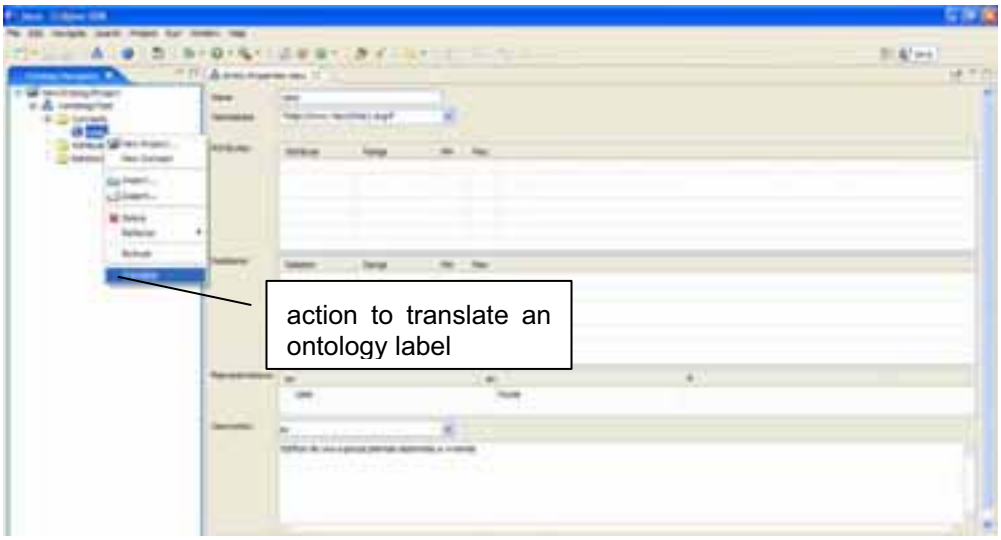


Figure 17. Popup action menu to translate a label (using TermTranslator)

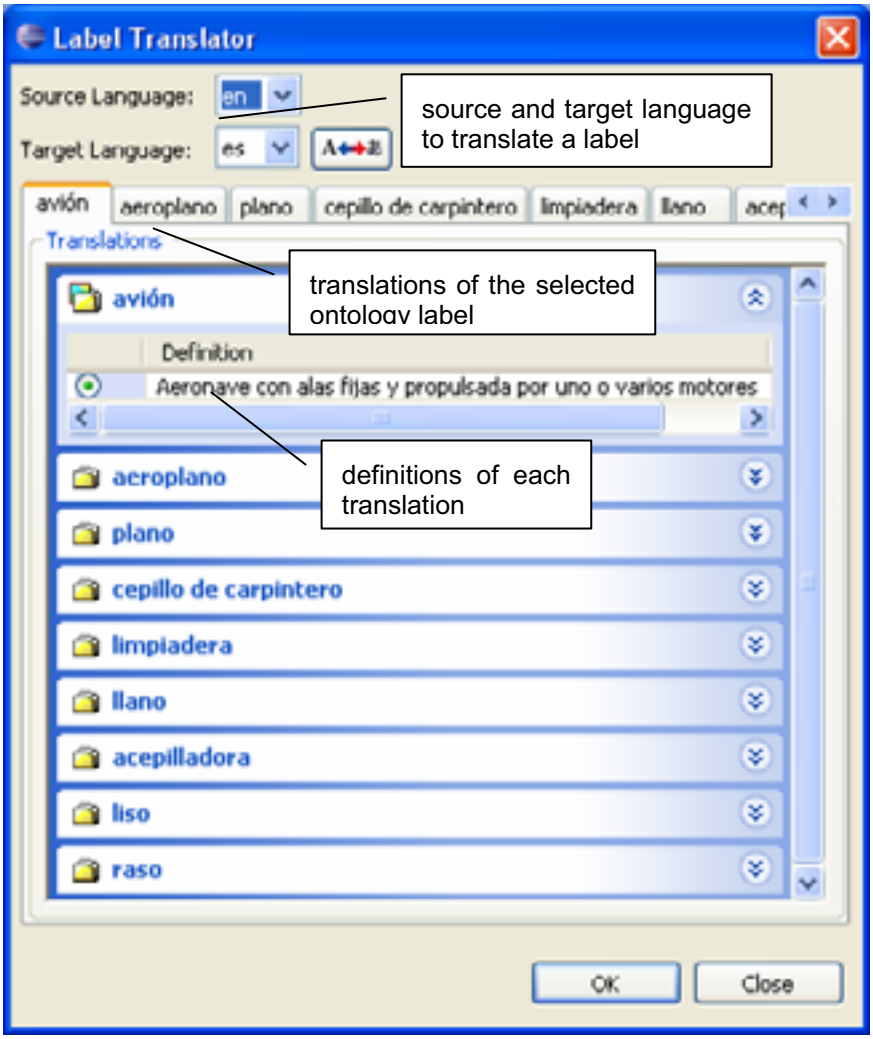


Figure 18. TermTranslator GUI dialog

Figure 19. TermTranslator GUI prototype to select both working and view languages

UC-4 Export

Overview

For a general overview of these use cases please refer to section 3.2.4. No engineering component has been identified for this use case yet.

Detailed description

UC-4: Export

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Primary Actor: Authorized users

Stakeholders and interests:

Users want to export ontologies to other formats (including legacy format, for backwards compatibility).

Preconditions: The user is logged and has at least view access to the selected ontologies.

Success Guarantee: Ontologies are exported to other formats.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions Points: N/A

Main Success Scenario:

1. The User selects to export the selected ontology.
2. The User chooses the new format; the available formats are: TagText, RDBMS, ISO2709, SKOS and TBX
3. The User selects the destination file/resource.
4. The system exports the ontology to the selected format.

Extensions: N/A

Special Requirements: N/A

Frequency of Occurrence: Medium

Plugins/tools

Since there is no engineering component identified for this use case yet; it is not possible to give a full description of the corresponding plugin. Next figure shows how the ontology expert interacts with the system for this use case.

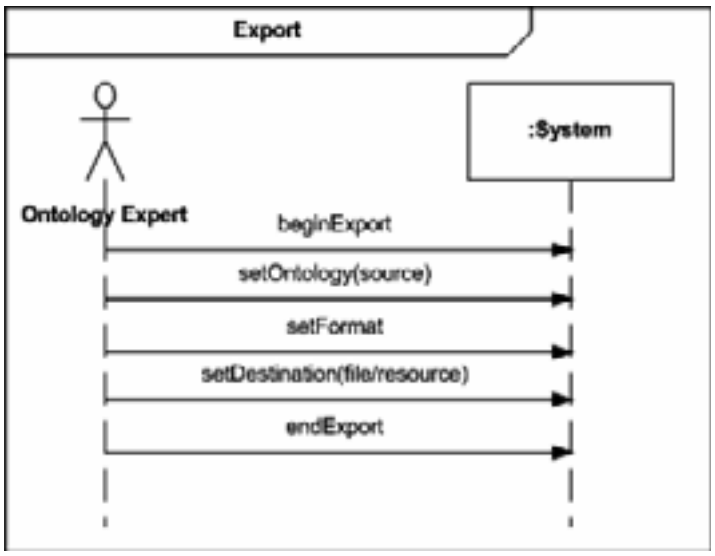


Figure 20. Export Sequence Diagram

UC-5 Convert

Overview

For a general overview of these use cases please refer to section 3.2.5.

This set of use cases will be implemented by integrating various plug-ins: UC-5.2 and UC-5.4.1 will be implemented with R_2O & ODEMapster; UC-5.3 and UC-5.4.2 will be implemented with X_2O & XMapster; while a plug-in or tool has not yet been identified for use case 5.1. There are other tools available for these use cases and they are: NeOn Toolkit (Core) and OntoMap.

Detailed description

UC-5 Convert

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Primary Actor: Ontology Engineer

Stakeholders and interests:

The ontology engineer wants to deal with the conversion of legacy data to ontologies

Preconditions: the Ontology Engineer has been logged in and has the permissions for manage the mappings between the ontologies and the legacy data.

Success Guarantee: Legacy systems are converted or the mapping documents are defined.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions Points: Extended by:

- UC-5.2: Populate from Database
- UC-5.3: Populate from existing resources with implicit schema
- UC-5.4: Define Resource converter
- UC-5.1: Convert Ontology

Main Success Scenario: UC-5.2

Extensions: UC-5.3, UC-5.4 and UC-5.1.

Special Requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-5.1 Convert Ontology

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and interests:

The ontology engineer wants convert an ontology from another format.

Preconditions:

The ontology engineer has been logged in and has the permissions for converting and ontology.

Success Guarantee: The ontology is converted.

Fail Guarantee: The System remains as it was before launching the conversion process.

Extensions Points: N/A

Main Success Scenario:

1. The Ontology Engineer informs the System that he wants to convert an ontology.
2. The Ontology Engineer chooses the ontology to be imported.
3. The ontology is imported into the System.

Extensions: N/A

Special Requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-5.2 Populate from Database

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and interests:

The ontology engineer wants to populate an ontology from a selected database.

Preconditions:

Ontology and database models pre-exist, and also the related resource converter (mapping document) for the ontology and database.

Success Guarantee: Ontology is populated with instances from the database.

Fail Guarantee: The system comes back to the state before initiating the use case.

Extensions Points: N/A

Main Success Scenario:

1. The ontology engineer tells the system to populate an ontology from a database.
2. The system asks to the ontology engineer to select the ontology that he wants to populate.

3. The ontology engineer selects the ontology.
4. The system asks to the ontology engineer to select the database.
5. The ontology engineer selects the database.
6. The system asks to the ontology engineer to select the related resource converter (mapping document) for the population.
7. The system shows to the ontology engineer the ontology that has been populated.

Extensions: N/A

Special Requirements: N/A

Technology and Data Variation List:

The database should be stored in ORACLE or MySQL; and the Ontology should be represented in OWL or RDF(S).

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-5.3 Populate from Existing Resources with Implicit Schema (XML)

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and interests:

The ontology engineer wants to populate an ontology from a selected XML resource.

Preconditions:

Ontology and XML models pre-exist, and also the related resource converter (mapping document) for the ontology and XML.

Success Guarantee: Ontology is populated with instances from the XML data.

Fail Guarantee: The system comes back to the state before initiating the use case.

Extensions Points: N/A

Main Success Scenario:

1. The ontology engineer tells the system to populate an ontology from a XML resource.
2. The system asks to the ontology engineer to select the ontology that he wants to populate.
3. The ontology engineer selects the ontology.
4. The system asks to the ontology engineer to select the XML resource.
5. The ontology engineer selects the XML resource.
6. The system asks to the ontology engineer to select the related resource converter (mapping document) for the population.

7. The system shows to the ontology engineer the ontology that has been populated.

Extensions: N/A

Special Requirements: N/A

Technology and Data Variation List: The Ontology should be represented in OWL or RDF(S).

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-5.4 Define Resource Converter

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and interests:

The ontology engineer wants to define a resource converter between legacy data and an ontology.

Preconditions:

The ontology engineer has been logged in and has the permissions for manage the mappings between the ontologies and the legacy data.

Success Guarantee: The mapping document is created.

Fail Guarantee: The system remains as it was before the execution of the use case.

Extensions Points:

- UC-5.4.1: Create a mapping document between ontologies and databases.
- UC-5.4.2: Create a mapping document between ontologies and XMLs.

Main Success Scenario: UC-5.4.1

Extensions: UC-5.4.2

Special Requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Low

Miscellaneous: N/A

UC-5.4.1 Create a mapping document between ontologies and databases

Scope: Fisheries Ontologies Lifecycle Management System.

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and interests:

The ontology engineer wants to define a resource converter (mapping document) for an ontology and a database schema.

Preconditions: Ontology and database schema pre-exist.

Success Guarantee: A resource converter (mapping document) is created.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions Points: N/A

Main Success Scenario:

1. The ontology engineer tells the system to define a resource converter (mapping document) for an ontology and a database schema.
2. The system asks to the ontology engineer to select the ontology schema.
3. The ontology engineer selects the ontology schema.
4. The system asks to the ontology engineer to select the database schema.
5. The ontology engineer selects the database schema.
6. The system asks to the ontology engineer to enter the mapping name.
7. The ontology engineer enters the mapping name.
8. *Include Edit mapping document for a database*.
9. The System creates the mapping.

Extensions: N/A

Special Requirements: N/A

Technology and Data Variation List:

The database should be stored in ORACLE or MySQL; and the Ontology should be represented in OWL or RDF(S).

Frequency of Occurrence: Low

Miscellaneous: N/A

UC-5.4.1.1 Edit mapping document for a database

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and interests:

The ontology engineer wants to edit information associated to the mapping document (resource converter) for an ontology and a database.

Preconditions: Ontology and database schema pre-exist.

Success Guarantee: Mapping document is edited.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions Points: N/A

Main Success Scenario:

1. The ontology engineer selects a specific concept of the ontology.
2. The ontology engineer selects a specific table of the database.
3. The ontology engineer informs the system that he wants to add a condition.
4. The ontology engineer selects a specific operation.
5. The ontology engineer introduces a table column or a constant value as an argument of the operation.

The ontology engineer repeats the steps 3...5 until he does not want to add more conditions.

6. The system creates a relation for the specific term and table.
7. The ontology engineer informs the system that he wants to add a transformation for one relation.
8. The ontology engineer selects an operation.
9. The ontology engineer selects a table column or a constant value as an argument of the operation.
10. The system creates the transformation.

The ontology engineer repeats the steps 7...10 until he does not want to add more conditions.

11. The ontology engineer tells the system that he has finished.

Extensions: N/A

Special Requirements: N/A

Technology and Data Variation List: The Ontology should be represented in OWL or RDF(S).

Frequency of Occurrence: N/A

Miscellaneous: N/A

UC-5.4.2 Create a mapping document between ontologies and XMLs

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and interests:

The ontology engineer wants to define a resource converter (mapping document) for an ontology and a XML schema.

Preconditions: Ontology and XML schema pre-exist.

Success Guarantee: A resource converter (mapping document) is created.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions Points: N/A

Main Success Scenario:

1. The ontology engineer tells the system to define a resource converter (mapping document) for an ontology and a XML schema.
2. The system asks to the ontology engineer to select the ontology schema.
3. The ontology engineer selects the ontology schema.
4. The system asks to the ontology engineer to select the XML schema.
5. The ontology engineer selects the XML schema.
6. The system asks to the ontology engineer to enter the mapping name.
7. The ontology engineer enters the mapping name.
8. *Include Edit mapping document for a XML.*
9. The System creates the mapping.

Extensions: N/A

Special Requirements: N/A

Technology and Data Variation List: The Ontology should be represented in OWL or RDF(S).

Frequency of Occurrence: Low

Miscellaneous: N/A

UC-5.4.2.1 Edit mapping document for a XML

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and interests:

The ontology engineer wants to edit information associated to the mapping document (resource converter) for an ontology and a XML.

Preconditions: Ontology and XML schema pre-exist.

Success Guarantee: Mapping document is edited.

Fail Guarantee: The system remains as it was before the execution of the use case.

Extensions Points: N/A

Main Success Scenario:

1. The ontology engineer selects a specific concept of the ontology.
2. The ontology engineer selects a specific table of the XML.
3. The ontology engineer informs the system that he wants to add a condition.

4. The ontology engineer selects a specific operation.
5. The ontology engineer introduces a table column or a constant value as an argument of the operation.

The ontology engineer repeats the steps 3...5 until he does not want to add more conditions.

6. The system creates a relation for the specific term and table.
7. The ontology engineer informs the system that he wants to add a transformation for one relation.
8. The ontology engineer selects an operation.
9. The ontology engineer selects a table column or a constant value as an argument of the operation.
10. The system creates the transformation.

The ontology engineer repeats the steps 7...10 until he does not want to add more conditions.

11. The ontology engineer tells the System that he has finished.

Extensions: N/A

Special Requirements: N/A

Technology and Data Variation List: The Ontology should be represented in OWL or RDF(S).

Frequency of Occurrence: N/A

Miscellaneous: N/A

Plugins/Tools

Metamodel for conversion

[Barrasa, 2007] proposes model for defining mappings between ontologies and databases. We will extend this model in order to include the NeOn networked ontology model proposed in WP1, and also to include other kind of resources, not only databases. Next figure shows the first draft of this metamodel so called metamodel for conversion.

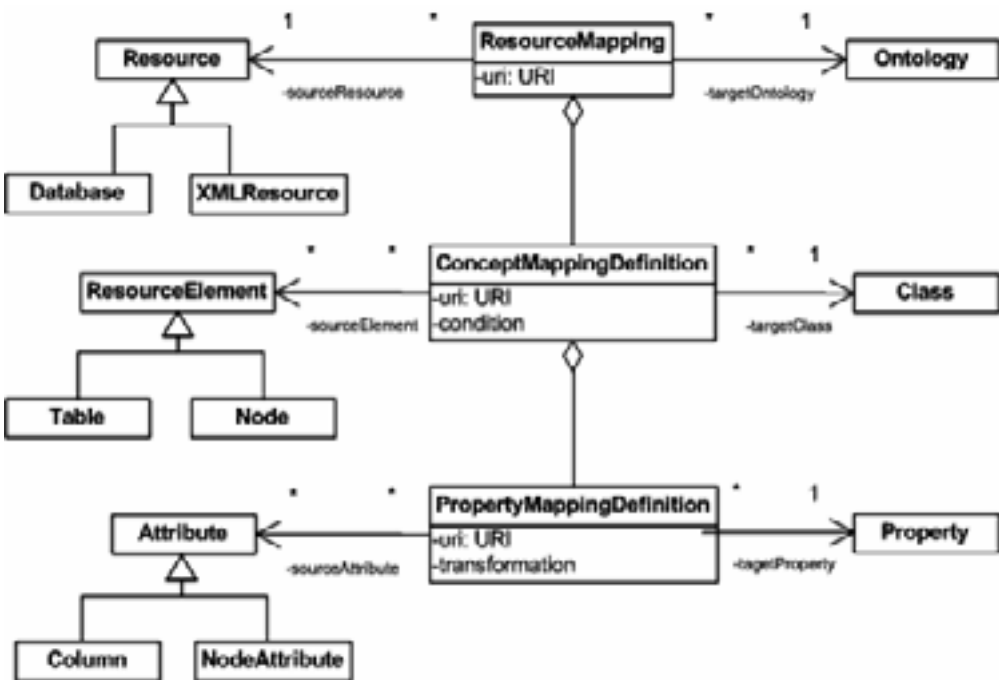


Figure 21. Metamodel for conversion

A ResourceMapping is defined between an ontology and a resource. An ontology is represented by the class Ontology in the OWL DL metamodel. A resource is represented by the class Resource; this resource can be a Database, represented by the class Database, or XML resource, represented by the class XMLResource. One association from ResourceMapping to Ontology, targetOntology, specifies the target ontology of the ResourceMapping. The association sourceResource specifies the source resource of the ResourceMapping. Cardinalities on both associations denote that to each ResourceMapping instantiation, there is exactly one Ontology connected as target and one Resource as source.

A ConceptMappingDefinition is defined between a resource and a class. A class is represented by the class Class in the OWL DL metamodel. A resource element is represented by the class ResourceElement; it can be a database table, represented by the class Table, or XML node, represented by the class Node. The association from ConceptMappingDefinition to Class, targetClass, specifies the target class of the ConceptMappingDefinition. The association sourceElement specifies the source element of the ConceptMappingDefinition. Cardinalities on both associations denote that to each ConceptMappingDefinition instantiation, there is exactly one Class connected as target and there are many Resources as source.

A PropertyMappingDefinition is defined between an attribute and a property. A property is represented by the class Property in the OWL DL metamodel. An attribute element is represented by the class Attribute; it can be a table column, represented by the class Column, or node attribute, represented by the class NodeAttribute. The association from PropertyMappingDefinition to Property, targetProperty, specifies the target property of the PropertyMappingDefinition. The association sourceAttribute specifies the source attribute of the PropertyMappingDefinition. Cardinalities on both associations denote that to each PropertyMappingDefinition instantiation, there is exactly one Property connected as target and there are many Attribute as source.

R_2O and ODEMapster details

R_2O is an extensible and declarative language to describe mappings between relational database (DB) schemas and ontologies, and ODEMapster which is the processor in charge of carrying out

the exploitation of the mappings defined using R_2O, performing both massive and query driven data upgrade. Also ODEMapster exports the ontology instances to database ones.

Implementation details

The implementation details for ODEMapster are presented here; including its package structure. Next figure shows the package structure of ODEMapster. The main package `net.barrasa.mapping` contains the following subpackages:

- `dbconn` package which provides access to the databases
- `engine` package which manages the execution of the R_2O mappings.
- `ontoClasses` package which provides access to the ontologies.
- `r2o` package which manages the R_2O mapping files.

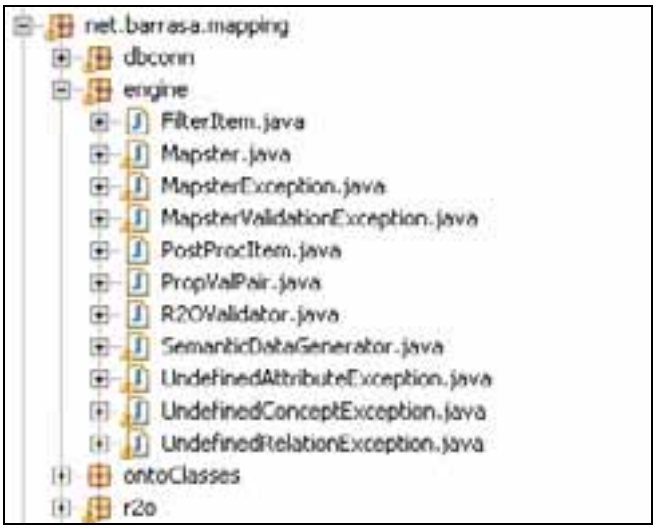


Figure 22. ODEMapster package structure

Configuration File

ODEMapster has a configuration file; in this configuration file we can set the following parameters:

```
r2o.file.path=examples/vessels/r2oVessel.xml
query.file.path=examples/vessels/qryVessel.xml
database.url=jdbc:mysql:///figis
database.driver=com.mysql.jdbc.Driver
database.user=root
database.pwd=root
output.file.path=vessels.rdf
onto.file.path=examples/vessels/vessels_model.rdf-xml.owl
```

These parameters include the R_2O document file path; the database information; and the ontology information.

X_2O and XMapster details

This section presents X_2O, an extensible and declarative language to describe mappings between XML schemas and ontologies, and XMapster which is the processor in charge of carrying out the exploitation of the mappings defined using X_2O, performing both massive and query driven data upgrade. XMapster also exports the ontology instances to XML.

Implementation details

This section present the implementation details of XMapster. This tool is under development; Next figure shows the current package structure of XMapster. The main package is **`es.upm.fi.dia.ontology.xmapster`**.

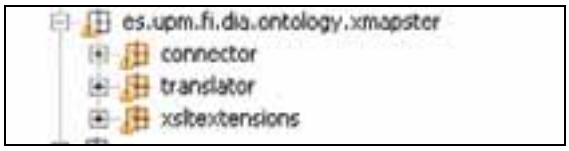


Figure 23. XMapster package structure

There are other available tools within, NeOn project, to be used for these use cases and they are:

- NeOn toolkit core for UC-5.2 and UC-5.3
- OntoMap for UC-5.4.

UC-5 Convert

The following figures show the system sequence diagram for this use case.

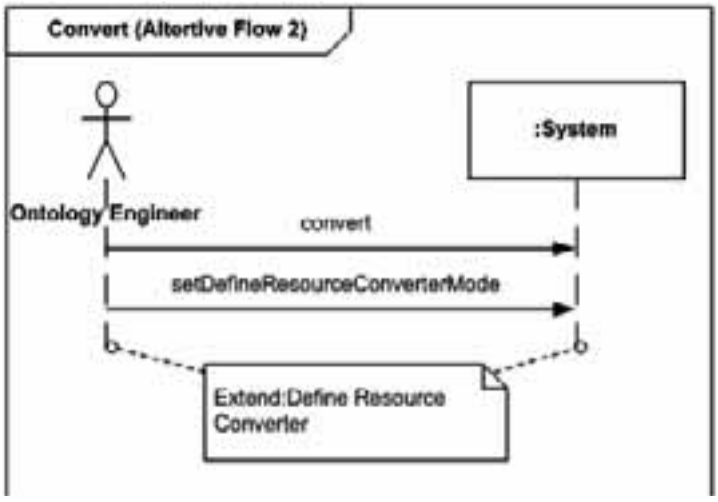


Figure 24. Convert Sequence Diagram (Define Resource Converter Mode)

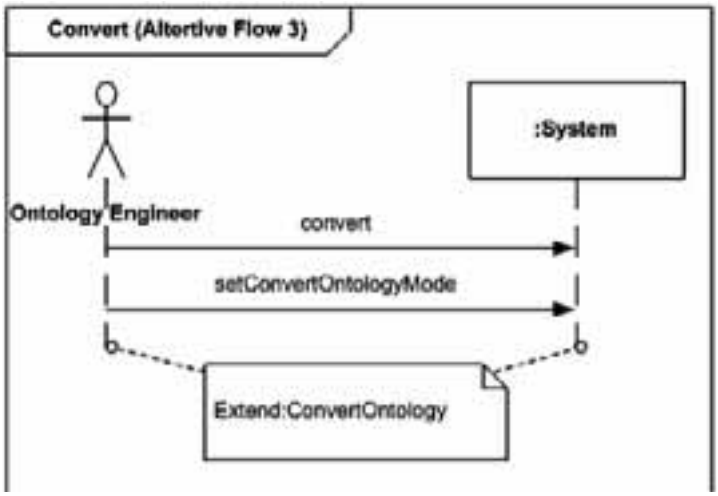


Figure 25. Convert Sequence Diagram (Convert Ontology Mode)

UC-5.4 Define Resource Converter

The following figures show the system sequence diagram for this use case.

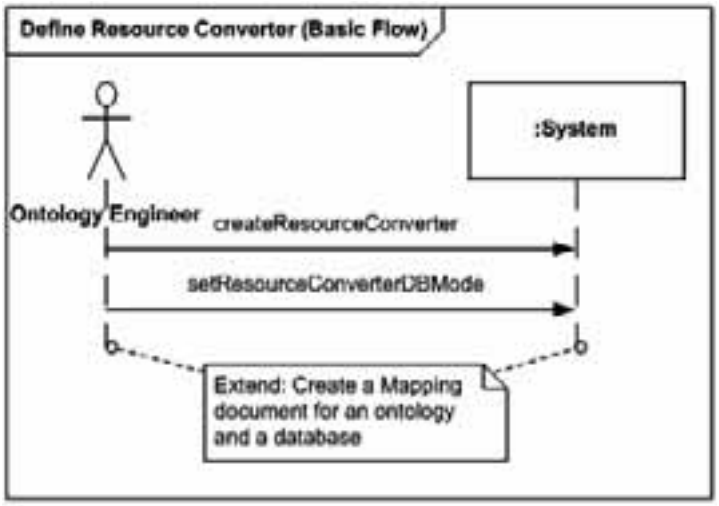


Figure 26. Define Resource Converter Sequence Diagram (for a DB)

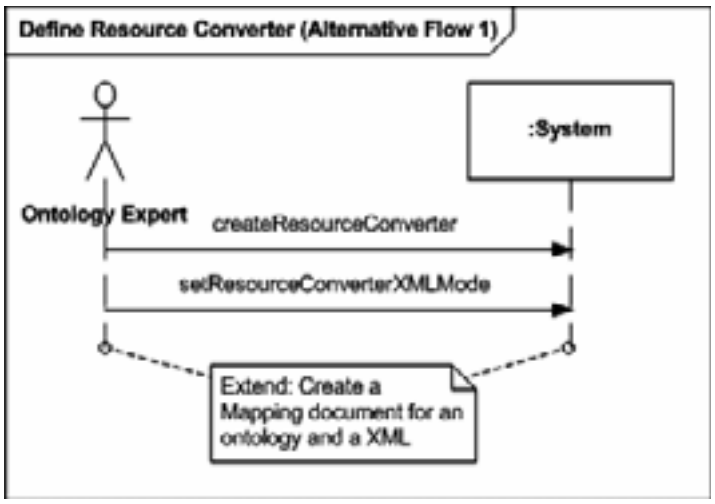


Figure 27. Define Resource Converter Sequence Diagram (for a XML)

UC-5.2 Populate from Database

The figure below depicts how this tool interacts with the user and the GUI prototype for export ontology instances to database.

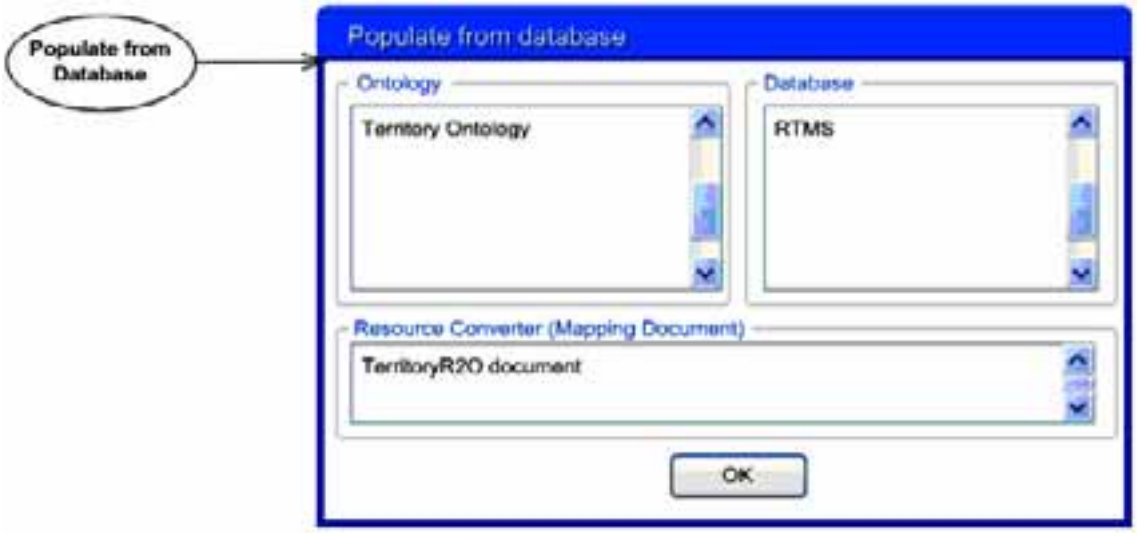


Figure 28. GUI prototype for Populate ontology from Database

The following figures show the system sequence diagram for this use case.

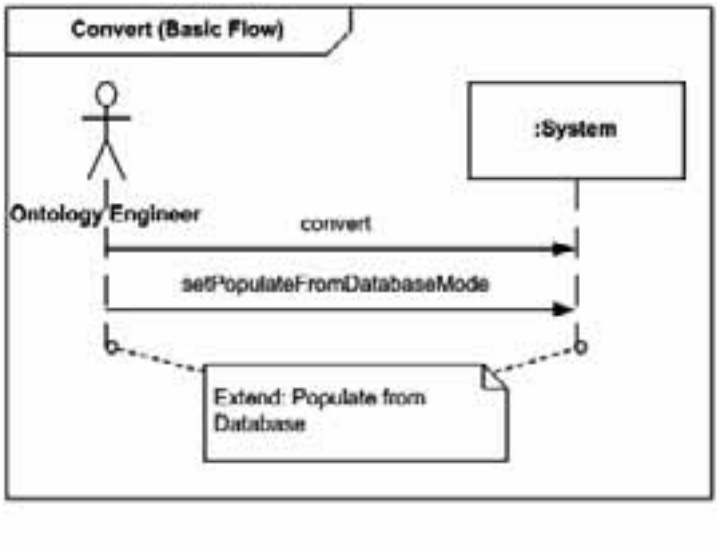


Figure 29. Convert Sequence Diagram (Populate from Database Mode)

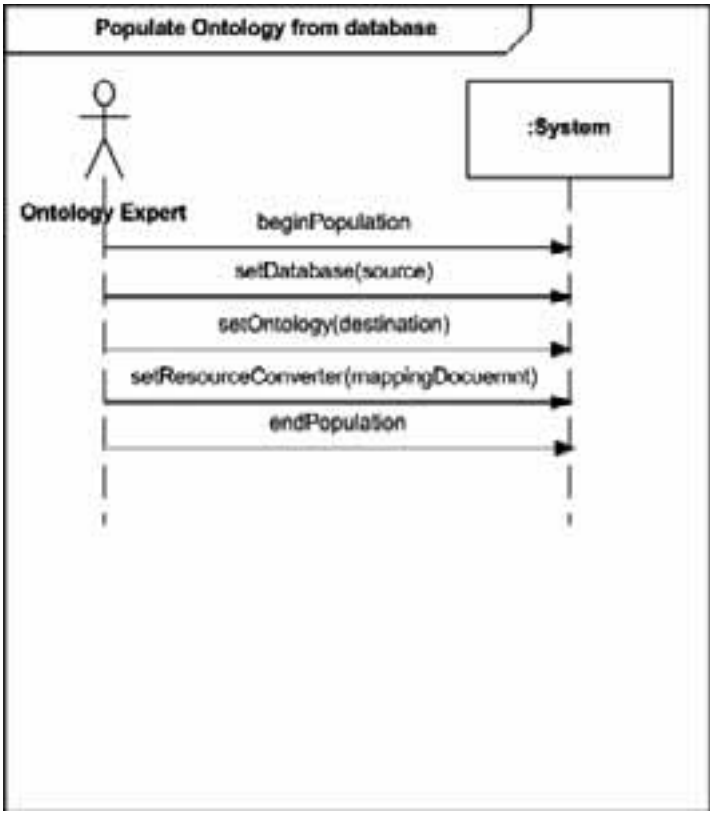


Figure 30. Populate Ontology from database Sequence Diagram

UC-5.4.1 Create a mapping document between ontologies and databases

The following mock-ups depict the GUI prototype for creating/editing database to ontology mapping.

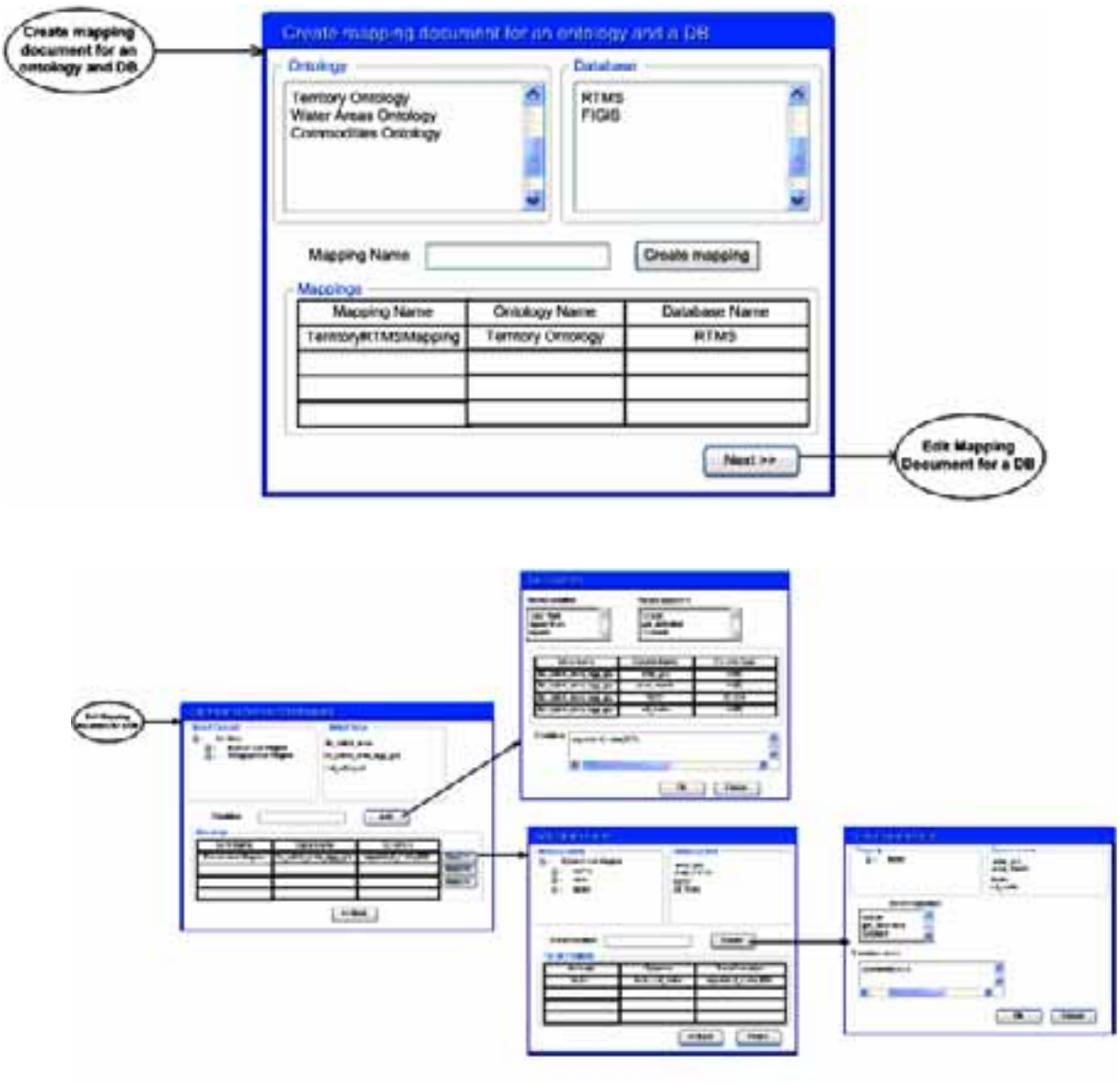


Figure 31. GUI prototype for creating/editing Database to ontology mapping

The following figures show the system sequence diagram for this use case.

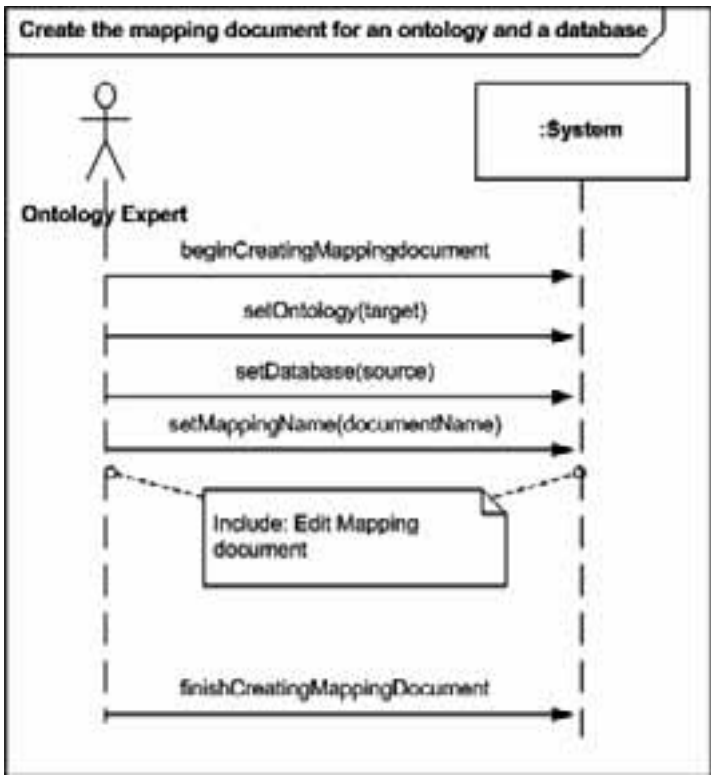


Figure 32. System sequence diagram for creating a mapping document between ontologies and databases.

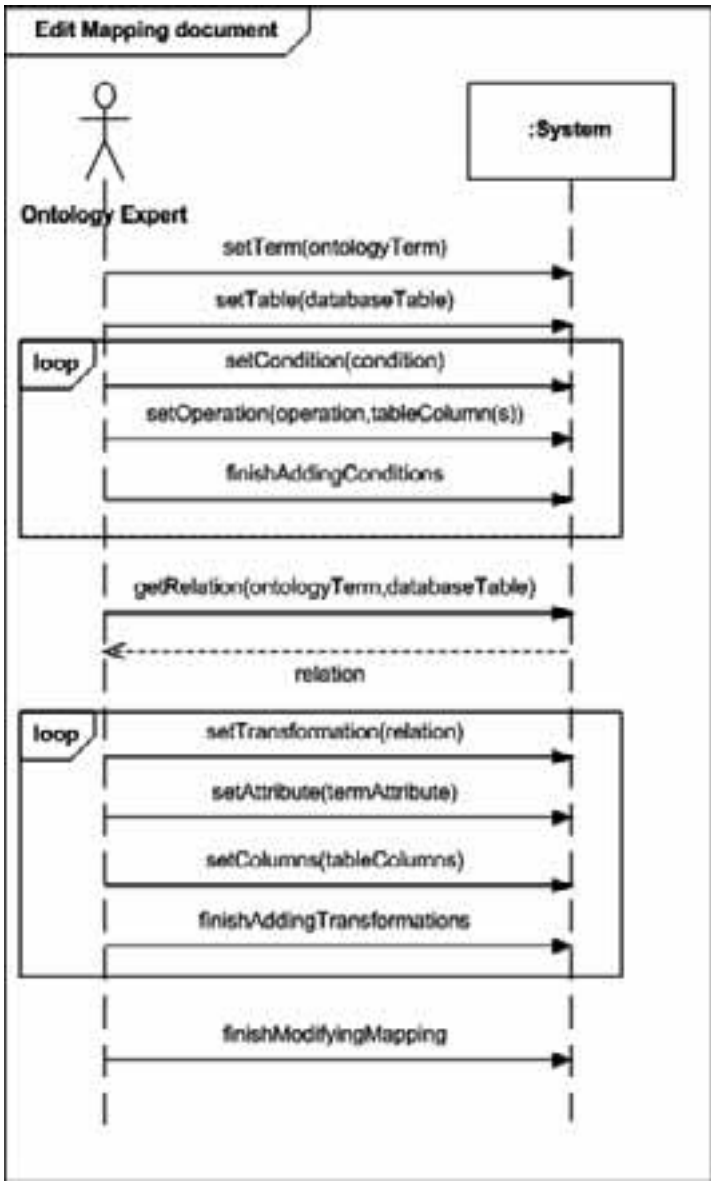


Figure 33. System sequence diagram for edit mapping document for a database use case.

UC-5.3 Populate from existing resources with implicit Schema (XML)

The following figure presents a mock-up showing how this tool interacts with the user, and in particular the GUI prototype for export ontology instances to XML.

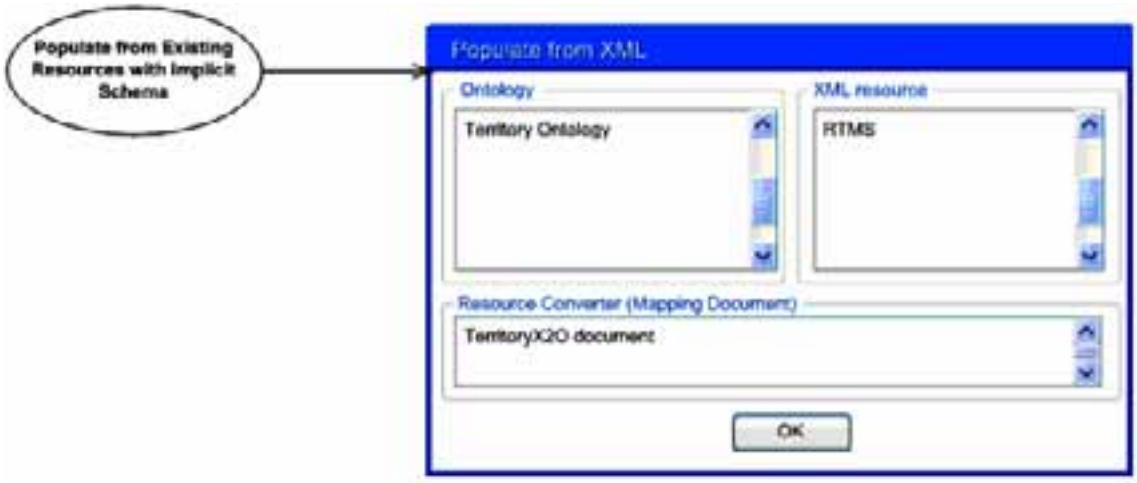


Figure 34. GUI prototype for Export Ontology to XML

The following figures show the system sequence diagram for this use case.

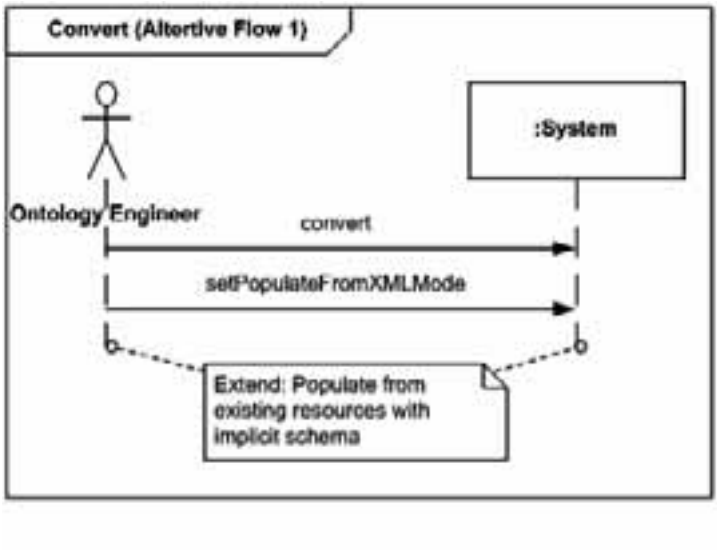


Figure 35. Convert Sequence Diagram (Populate from XML Mode)

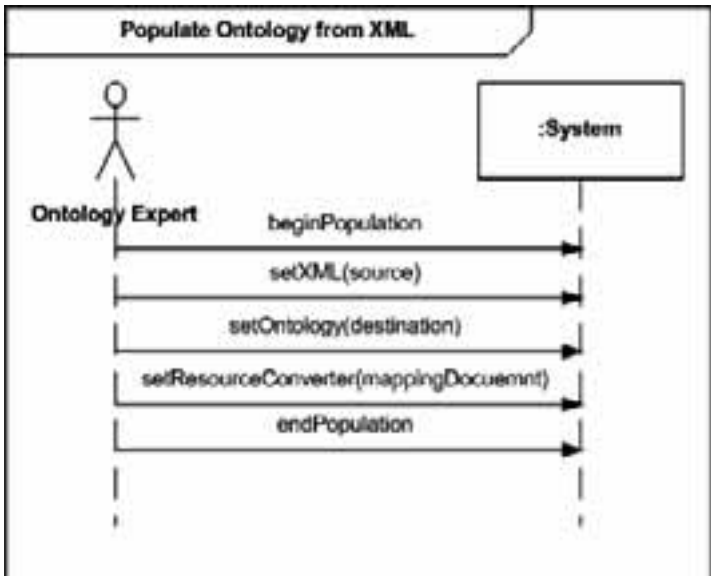


Figure 36. Populate Ontology from XML Sequence Diagram

UC-5.4.2 Create a mapping document between ontologies and XMLs

The following figure depicts the GUI prototype for creating/editing XML to ontology mapping.

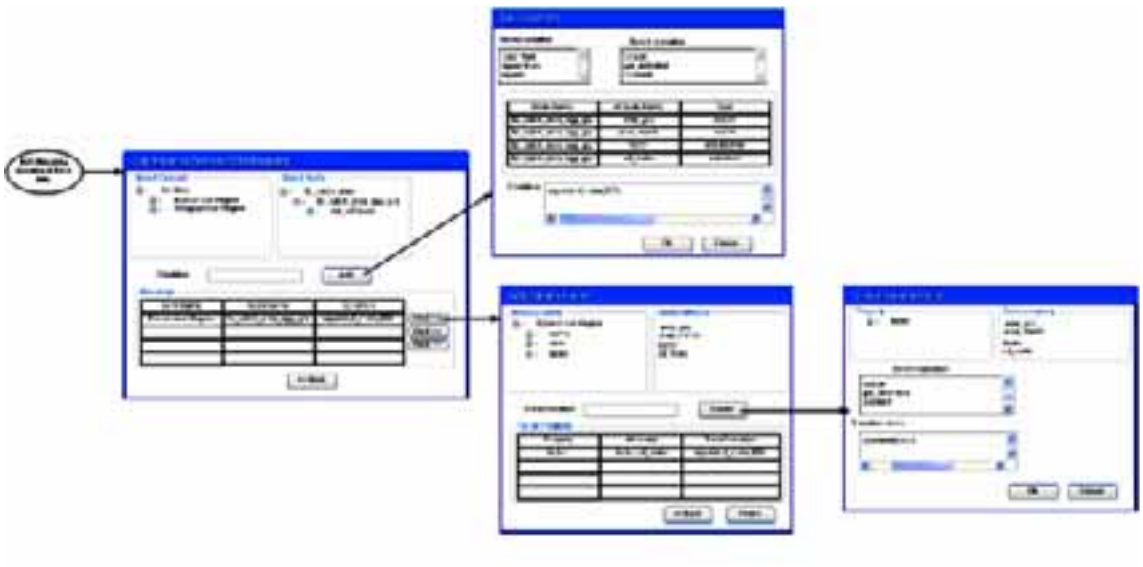


Figure 37. GUI prototype for creating/editing XML to ontology mapping

The following figures show the system sequence diagram for this use case.

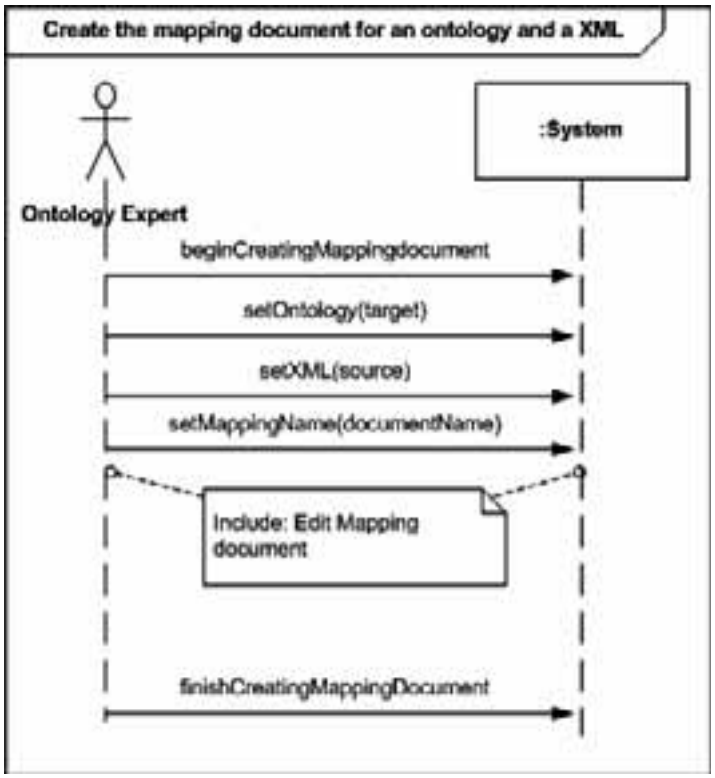


Figure 38. System sequence diagram for creating a mapping document between ontologies and XMLs.

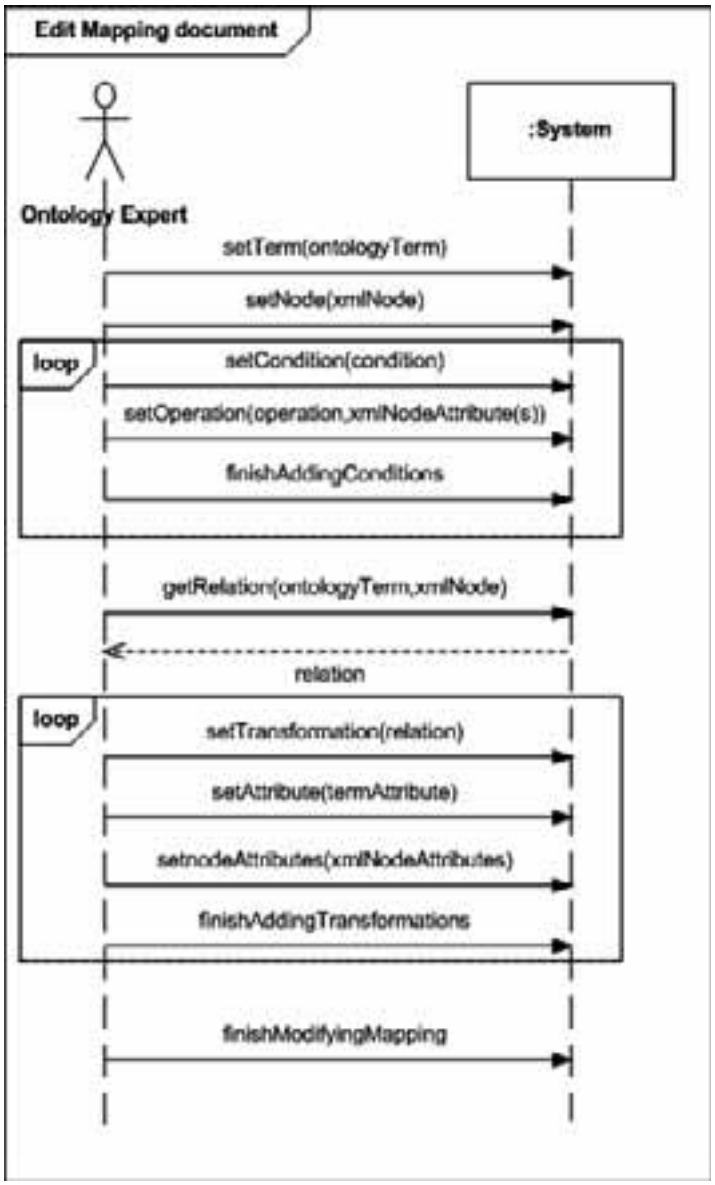


Figure 39. System sequence diagram for edit mapping document for a XML use case.

UC-6: Create mappings

Overview

The ontology editors and ontology engineers may create mappings between ontology elements, i.e., classes, instances and relations inside different ontologies. Any mapping can be either created in a totally manual way or in a semi-automatic way (i.e., system-supported). If a mapping between elements in two ontologies is established, then there is a mapping between the two ontologies. This use case refers to the ontology aligning activity.

We refer readers to section 3.2.6 for a general overview of use case of managing mappings.

Detailed Description

UC-6: Create mappings

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Primary Actor: Ontology Editor

Stakeholders and Interest:

The ontology editor wants to create a mapping between ontology elements in different ontologies.

Preconditions:

The ontology editor has been logged in and has the permissions for manage the mapping.

Success Guarantee:

Mapping is created, modified or deleted. Mapping assertions are attached correctly to the mapping

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions points:

- UC-6.1: Manage Manual Mappings
- UC-6.2: Create Automatic Mappings

Main Success Scenario (or Basic Flow): UC-6.1

Extensions (or Alternative Flows): UC-6.2

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-6.1 Create mappings manually

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Editor

Stakeholders and Interest:

The ontology editor wants to create a mapping between two ontologies manually.

Preconditions: The ontology editor has been logged into system.

Success Guarantee:

Mapping is created. Mapping assertions are attached correctly to the mapping.

Fail Guarantee: The system remains as it was before the execution of the use case.

Extensions points: N/A.

Main Success Scenario (or Basic Flow):

1. Add Ontologies to the “Mapping View”
2. Defining new dependencies
3. Deleting dependencies
4. Transformation
5. Enhancement of the mapping folder
6. Create different kinds of mappings manually
7. The system creates and stores the mapping

Frequency of Occurrence: Medium

UC-6.2. Create mappings semi-automatically

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Editor

Stakeholders and Interest:

The Ontology Editor wants to create a mapping between two ontologies in a semi-automatic way.

Preconditions: The ontology editor has been logged into system.

Success Guarantee:

Mapping is created semi-automatically. Mapping assertions are attached correctly to the mapping.

Fail Guarantee: The system remains as it was before the execution of the use case.

Extensions points: N/A

Main Success Scenario (or Basic Flow):

1. The Ontology Editor informs the system that he wants to semi-automatically create a mapping.
2. The Ontology Editor sends the source ontology to the system.
3. The Ontology Editor sends the destination ontology to the system.
4. The Ontology Editor sends initial mapping to the system.
5. The Ontology Editor sends the mapping technique to the system.
6. The Ontology Editor sends the auxiliary resources related with the mapping technique to the system.
7. The system creates a proposal of mapping and sends a log to the Ontology Editor.
8. The Ontology Editor revises the automatically created mapping, chooses the correct relations and confirms them.

9. The system creates and stores the mapping.

Extensions (or Alternative Flows):

(In 4) The Ontology Editor does not send initial mapping to the System. The use case continues on step 5.

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

Plugins/tools

This set of use cases should be provided by OntoMap and FOAM. OntoMap works as a perspective in Eclipse framework. The “Mapping” perspective is needed to build alignments and correlations between different ontologies or certain parts in a particular ontology.

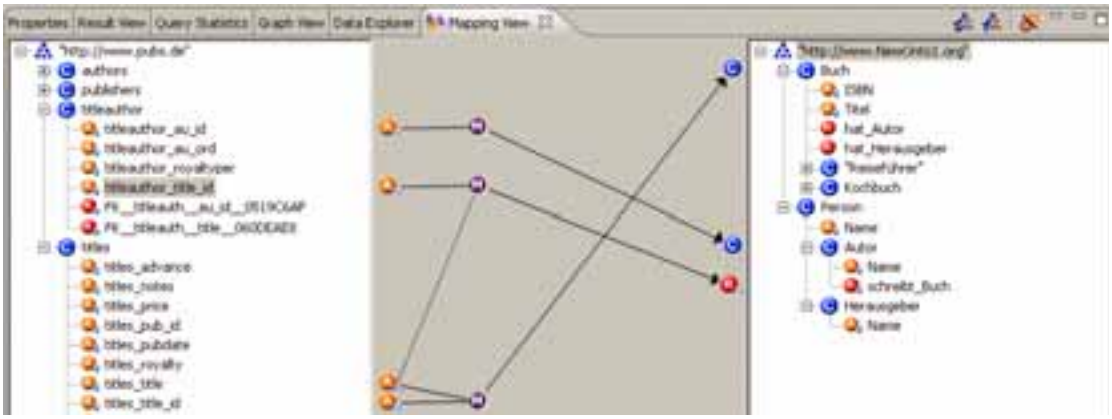


Figure 40. OntoMap Screenshot

FOAM (Framework for Ontology Alignment and Mapping) is a tool to fully or semi-automatically align two or more ontologies. It is based on heuristics (similarity) of the individual entities (concepts, relations and instances). The outputs are pairs of aligned entities with some confidence values.

UC-6.1 Create mappings manually

The “Mapping View” enables you to define mappings between different source (on the left side) and target-ontologies (on the right side). A released mapping between two elements is visualized by an arrow and can be activated by Drag &Drop from source to target elements.

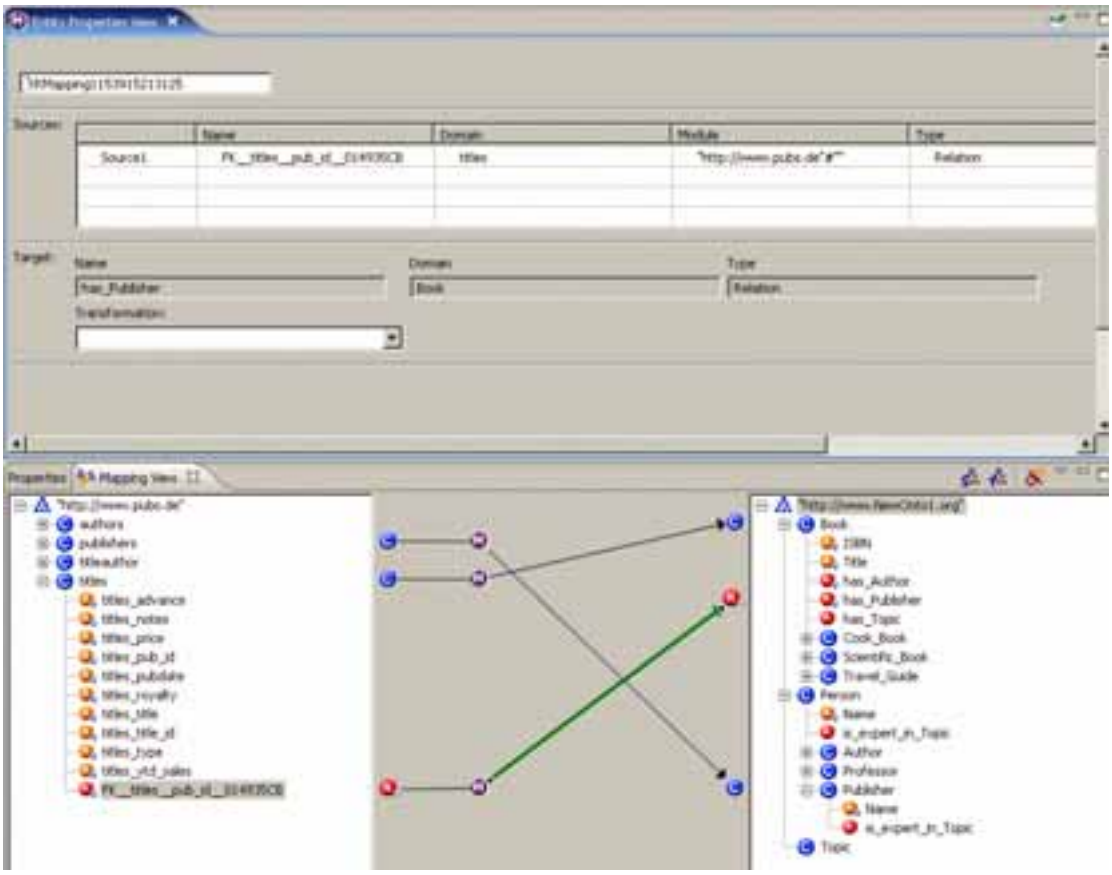


Figure 41. Mapping View

To activate the “Mapping View”, click on the “Mapping View” symbol right above the “Schema View” tab.

Or, if the “Mapping View” window has not been opened automatically, you can do it manually by Window -> Show Mapping View or Window -> Show View -> Other -> Mapping View.

Control Elements

In the mapping view there are three action-buttons and one dropdown list with the corresponding functions available.

Figure 42. The buttons of the “Mapping View”

If you drop down the list, the respective functions of the buttons are shown.

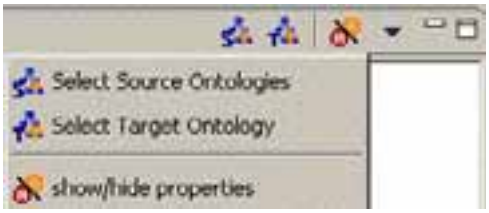


Figure 43. List field of the “Mapping View”

Source and Target Ontologies can be added with the aid of the context menu within the “Ontology Navigator” and the “Mapping View” or via buttons in the

- “Mapping View”
- “Select Source Ontologies” Button
- “Select Target Ontologies” Button
- “Show/Hide Properties”

Click on the buttons or list entries to open the “Select Ontologies” dialogue and select “Source” or “Target” ontologies.

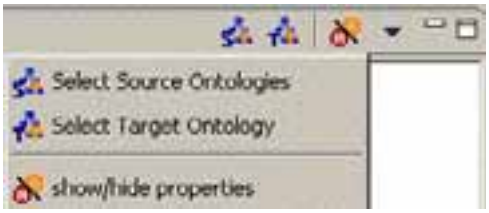


Figure 44. Dialogue “Select Ontologies” for “Mapping View”

Add Ontologies to the »Mapping View«

1. Select the element you want to add as Source or Target ontology by right clicking on the entry in the “Ontology Navigator”.
2. Choose “Add as Mapping Source” or “Add as Mapping Target” in the context menu.

The “Mapping View” opens in the group of Properties / Mapping View.

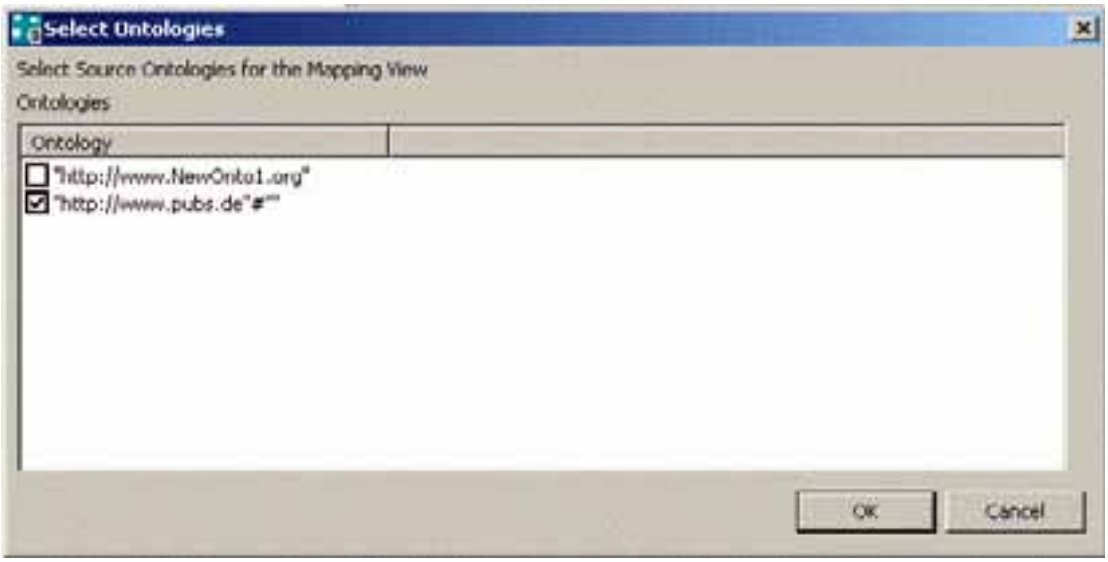


Figure 45. “Mapping View”

3. The Mapping View is divided into three parts. The left window shows the Source ontologies and the right window the Target ontologies. The arrows in the middle show the dependencies between the elements of the ontologies.

Defining new dependencies

1. Select the element of the Source ontology you want to depend on an element of the Target ontology.

2. Push and hold the mouse button onto the source-element and drag it on the requested element in the target ontology.

The dependency is indicated by an arrow in the middle of them. This new mapping is visible for you as arrow as well as mapping element in the mapping folder in the “Ontology Navigator”.

Deleting dependencies

1. Click on the arrow / dependency you want to remove.
2. Click <DEL>.
3. Click the right mouse button and select “Remove Mapping” in the context menu.
4. Alternatively you also can delete the elements in the “Ontology Navigator”.

The specifics of dependencies between two elements can be displayed with the help of a Tooltip. Therefore move the mouse over the requested arrow for a short time.

To create different mappings

OntoMap allows the creation of different “Mapping” connections:

- from concept to concept (CC)
- from attribute to concept (AC)
- from attribute to attribute (AA)
- from attribute to relation (AR)
- from relation to relation (RR)

Before you can map attributes and/or relations to the Target ontology, first there must exists an attribute-concept or a concept-concept “Mapping”, which specifies the identifier for the mapped instances within the Target ontology. Provided “Mappings” are visualized as arrows in the “Mapping View” and as mapping elements underneath the file “Mappings” within the target ontology in the “Ontology Navigator”. By selecting a mapping element in the “Ontology Navigator” or within the “Mapping View”, its properties in the “Entity Property View” are represented and can be edited there.

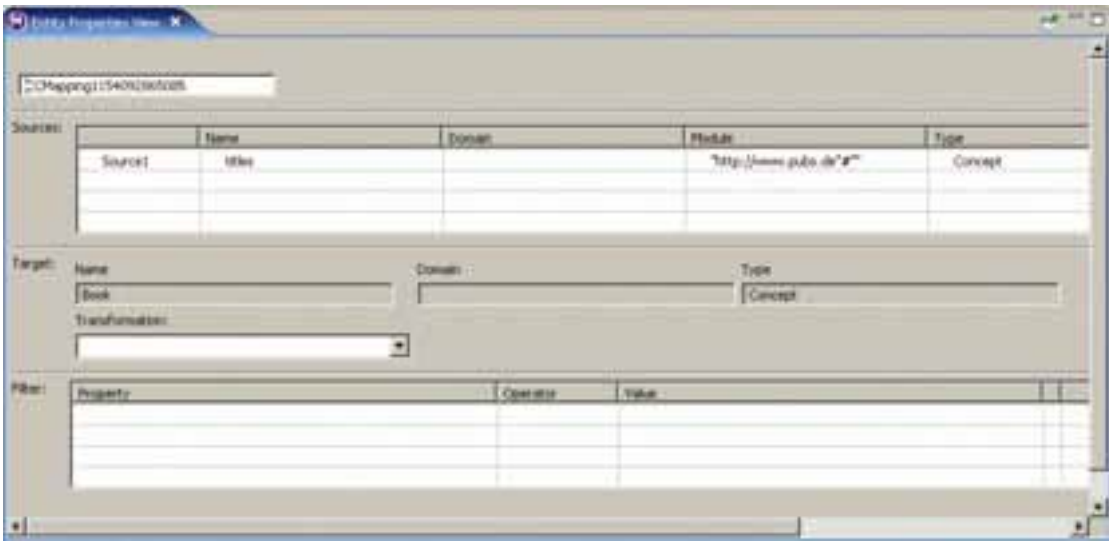


Figure 46. “Entity Properties View” of a Mapping

In the “Entity Properties View” you can see

- which kind of “Mapping” it concerns (CC, AC, AA, AR, RR),
- which element is the target element of the “Mapping” (Target)
- which element is the source element of the “Mapping” (Sources),
- whether type transformations for source elements are present,
- whether filters were defined (with attribute-concept “Mappings”) and
- whether a deviating identifier is to be used (by attribute-relation “Mappings”).

The following different “Mapping” types and their properties are described in the following on the basis of two example ontologies.

Transformation

It is possible to execute transformations for mappings in the “Entity Properties View”. These transformations can be defined from the Source to the Target ontology. You can choose a predicate in the “Entity Properties View”, whose arguments will be allocated with values or elements of the Source / Target ontology.

For example, you can use it to multiply a numerical value from the Source ontology to the Target ontology. Therefore you can use the “multiply” predicate: multiply ([Source1], [1.2], [Target])

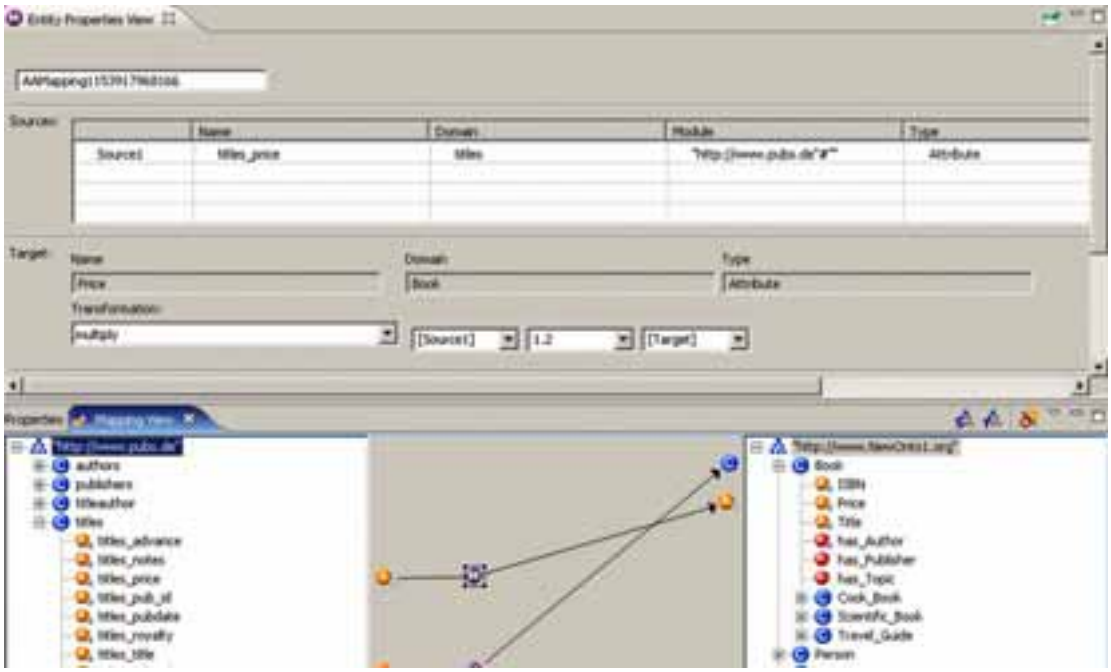


Figure 47. Transformation

Enhancement of the mapping folder

By clicking on a mapping element in the “Ontology Navigator”, the “Mapping View” opens. Now you have displayed the selected mapping in the “Mapping View”. The function “Show Mapping” in the context menu opens the view, even if it was closed before.

Figure 48. Context menu "Show Mapping" in the "Ontology Navigator"

Concept to concept mapping (CC)

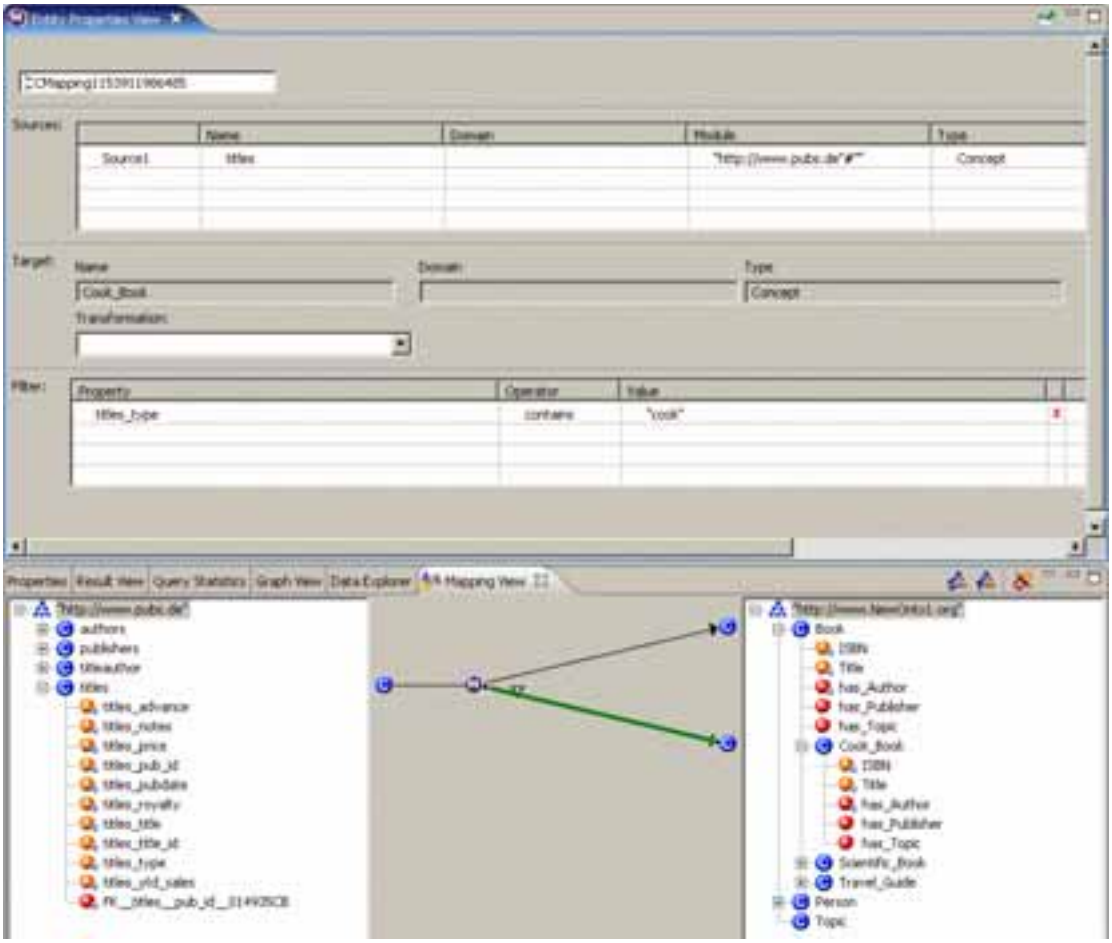


Figure 49. Concept-to-concept "Mapping"

By connecting a concept of the left side by "Drag & Drop" with a concept on the right side, a "concept-concept" mapping was produced.

In our example the concept of "titles" is illustrated on the both concepts "Book" and "Cook_Book". The mapping on "Cook_Book" has a filter, which considers only those instances of "titles", which contain the value "cook" as "Cook_Book".

Filters can be specified for "concept-to-concept" and "attribute-to-concept" mappings and gives the user the possibility to limit those mapped instances over their characteristic values. Filters are marked over an appropriate icon within the "Mapping View" and can be specified as follows:

1. Click on the arrow in the graphical diagram for which you want to define a filter.
2. Enter the desired values within the range filter.

   Property: The associated attributes and relations of the source element are provided.

   Operator: Different operators are offered.

   Value: Here you enter the value, with which the filter works.
3. For one mapping, several filters can be specified and linked by means of an “and” function.

   The filter was defined.

By selecting a “concept-to-concept” mapping, all subordinated mappings (“attribute/relation-to-attribute/relation”) are deleted automatically.

Attribute to concept mapping (AC)

Arrange a connection from the attribute on the left side to a concept on the right side and then you get an “attribute-to-concept” mapping.

To build explicit identifier you can integrate several attributes of a concept to one <id>-function. Therefore you can drop further attributes on the mapping icon in the left third of the mapping.

Attribute to attribute mapping (AA)

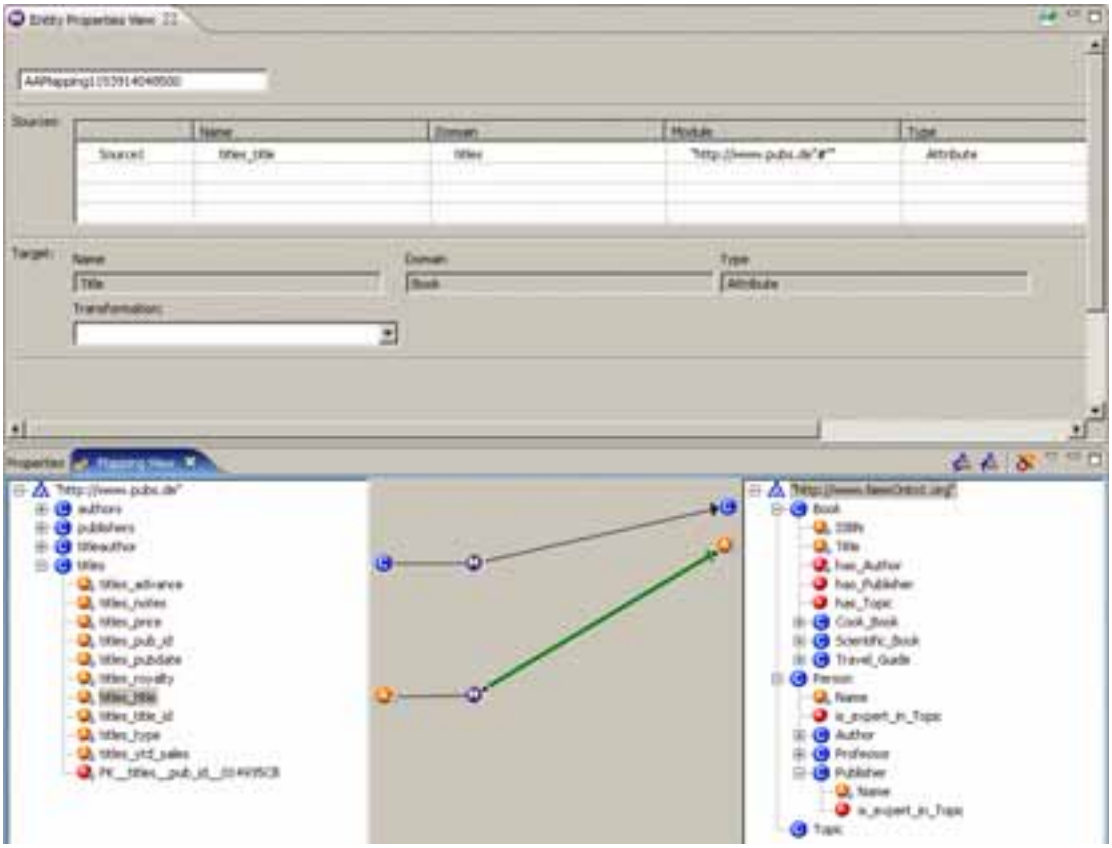


Figure 50. Mapping from attribute-to-attribute

An “attribute-to-attribute” mapping can be build by connecting two attributes. If no “concept-to-concept” or “attribute-to-concept” mapping exists so far, their concepts will be mapped to each other automatically.

If you connect the attributes “titles_title” and “Title” with each other, the concepts “titles” and “Book” will be mapped automatically.

To build an “attribute/relation-to-attribute” mapping, the “attributeto-concept“ or “concept-to-concept“ mapping appending on it has to exist.

Attribute to relation mapping (AR)

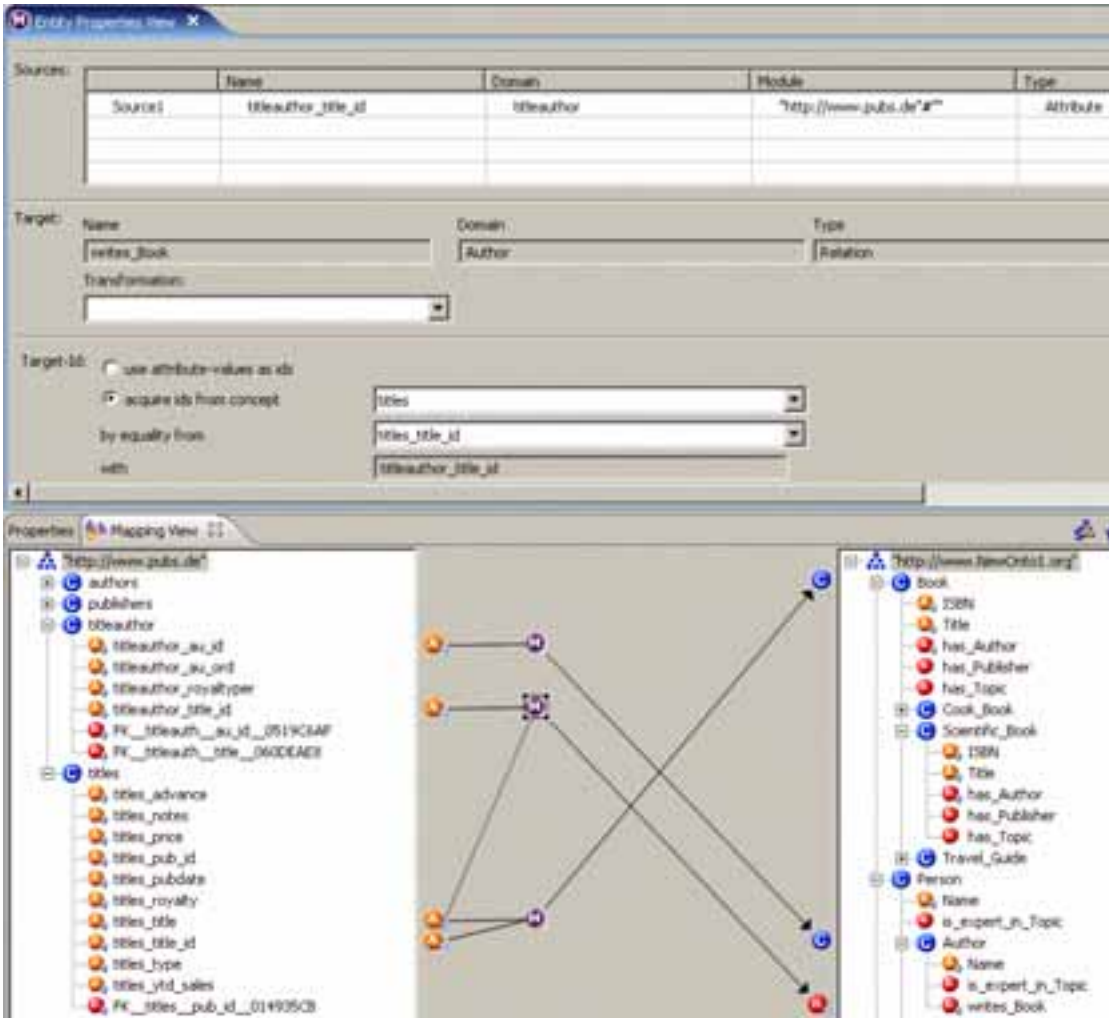


Figure 51. Mapping of Relation to Relation

For “attribute-to-relation” mappings the values of the source attribute will be assumed as instance-id for the target-relation.

This kind of mapping is necessary if two concepts in the Source ontology are only defined with attribute values and not with an explicit relation. In this example the attribute values “titleauthor_title_id” could be relation values of “writes_Book”.

In the case described here, the “titles_title_id” of the concept “titles” is not the instance-id for the target ontology. It is only the combination of “titles_title” and “titles_title_id”, therefore the correlation you want to express between “author.writes_Book” and “Book” gets lost in the Target ontology.

However, you can express the correlation in the Target ontology if you choose the concept whose instance-ids in the Target ontology - in case of equality of the selected attribute with the attribute you want to map - can be disposed for the target relation.

In case of equality of source attributes “titleauthor_title_id” and “titles_title_id” in the example above (displayed by thin dashed line), the combination of “titles_title” and “titles_title_id” is assumed as relation value for the target relation “author.writes_Book”.

Relation to relation mapping (RR)

To map relations to relations, the appropriate concepts of these relations have to be mapped as well. OntoMap suggests an automatically compilation of a “concept-to-concept” mapping, if this has not been build yet.

If the relation “FK_titles_pub_id_01…” has been mapped to “has_Publisher”, the necessary mapping from “titles” to “Book” will be suggested automatically, if there is no adequate mapping.

UC-6.2. Create mappings semi-automatically

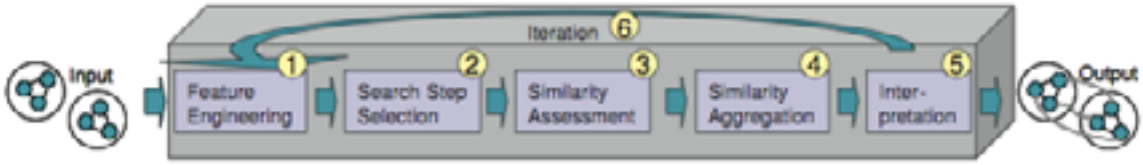


Figure 52. FOAM Mapping Process

The alignment process of FOAM is defined by the steps of the process model as follows:

1. *Feature Engineering.* First, the ontologies for the NOM approach have to be in a format equivalent to the RDFS format.
2. *Search Step Selection.* The selection of data for the comparison process can be straight forward to compare all entities of the first ontology with all entities of the second ontology. Or it can use the efficient way by ignoring some candidate concepts pairs.
3. *Similarity Computation.* The similarity computation determines the similarity values between two entities in two ontologies. Each similarity function is based on an ontology feature such as super concepts and the respective similarity measure.
4. *Similarity Aggregation.* Since there are always several ways to get the similarity values for a candidate pair of entities from two ontologies, they must be aggregated into a single aggregated similarity value.
5. *Interpretation.* Interpretation uses the individual or aggregated similarity values to derive mappings. Given similarity values above the threshold t, the two involved entities are mapped onto each another.
6. *Iteration.* Several algorithms perform iteration over the whole process to bootstrap the amount of structural knowledge. Iteration may stop when no new mappings are proposed, a fixed number of rounds is reached, or changes below a certain threshold, or dynamically depending on how much time and calculation power can be provided.

FOAM Package and Usage

Figure 53. The structure of FOAM package

Figure above shows the structure of FOAM package. The complex sub-package provides more sophisticate approaches to align ontologies, combine the results of multiple individual rules, and so on. In input sub-package, it gives the structure, which is supposed to be compared. In FOAM, only the structure of ontology is supported so far. However, the alignment algorithm also supports the non-ontology structure that is given in the nonOntologies sub-package. The followings give some description and the usage about main, agenda and combination sub-packages. For other sub-packages, we do not provide more details since they are obvious or they have already given in some publications.

Main sub-package First of all, the classes under subpackage main are the main entries to control all the algorithms. FOAM provides various main entries for different applications. For example, “Main” class is used to align one pair of ontologies. “MainMany” is to align more than one pair of ontologies automatically. Other main entries such as “MainOAEI” which is suitable for the alignment tasks organized by OAEI, are developed for specific applications. Besides, the parameters can be configured through the class “Parameter” in the main entries or by a separate local file. Anyway, for the parameters, the paths for the ontologies to be aligned and the file to store the alignment results are mandatory. Further, it is strongly recommended to use some existing alignments as external input. As for other parameters, they will be mentioned respectively in different sub-packages.

Usage: When a separate parameter file is needed, the align files such as “Align” or

“AlignMany” in main sub-package can be called in the following way:

java -cp kaon2.jar;align.jar edu.unika.aifb.foam.main.Align d:/parameterFile.txt

The minimal needs in the parameter file should include the following sentences:

ontology1 = D:/foam/ontologies/animalsA.owl;

ontology2 = D:/foam/ontologies/animalsB.owl;

cutoffFile = D:/foam/results/result.txt;

Also, one can configure all the parameters in the main classes like “Main” and “Main-Many” directly with the following command as an example:

java -cp kaon2.jar;align.jar edu.unika.aifb.foam.main.Main

Agenda sub-package The classes in this sub-package are used to determine which entities should be compared. The simplest one is the “CompleteAgenda”, resulting in n × m comparisons. Others like random agenda and close labels agenda, are more elaborated and therefore more efficient. The FOAM system has given an appropriate cooperation for these efficient agendas. What the users can do is to choose which kind of agenda they prefer, complete one or efficiency one.

Usage: For specific configuration of the agendas, one can input the following sentence into the separate local file of parameters.

efficientAgenda = EFFICIENT;

or efficientAgenda = COMPLETE

Or, the following value can be passed to the class “Parameter” in the main entries:

Parameter.COMPLETE or Parameter.EFFICIENT

Combination sub-package This package is used to aggregate the output of individual rules. This can be done with a manual approach or a machine learned approach.

Usage: The following two lines give some examples about how to configure the combination strategy in the parameter file and in the main class entries separately.

strategy = DECISIONTREE; //or EQUALLABELS, MANUALWEIGHTED, etc.

Parameter.DECISIONTREE //or Parameter.MANUALWEIGHTED, etc.

UC-7 Visualize

Overview

UC7 describes use cases that present the ontology or ontologies to users in an easily accessible way, i.e. graphically or in hierarchically. In contrast to viewing single entities with great detail, e.g. a concept with all relations and attributes and labels in different languages, the visualizations are meant to give an overview of the whole ontology and enable users to “get the big picture”. In order to facilitate the overview character, the visualizations use graphical means to display the dependencies of ontology elements. For the same reason some details of the model will not be displayed by these visualizations.

Visualizations for different user roles and different steps in the ontology lifecycle must be distinguished. For a general overview of this use case please refer to section 3.2.7.

Detailed Description

UC-7.1 Visualize Ontology for Ontology Engineers

Scope: Ontology conceptualization and population by Ontology Engineers (cf. Section 2.3)

Level: Sub function

Stakeholders and Interests:

Ontology Engineers need to visualize ontologies, when performing activities like (iterative) ontology conceptualization and population

Preconditions: One or more ontologies are opened in the toolkit

Success Guarantee: Ontology is visualized as a graphic view

Fail Guarantee:

System fails to display the visualization and should provide feedback for the reason

Extensions Points:

UC-7.2 refines this UC by filtering certain ontology elements based on user role and status information

Main Success Scenario:

1. User selects one or more ontologies (modules).
2. User selects “Visualize Ontology” from menu or icon toolbar.
3. System visualizes selected ontologies in a window.
4. User interacts with the graph to navigate to certain classes or properties.

Extensions:

Selected ontology is too big to visualize graphically:

1. System returns a warning indicating the problem.
2. User chooses a particular module, or root-class of the ontology or U chooses other view to visualize the ontologies such as a tree view.

Frequency of Occurrence: High

UC-7.2: Visualize Ontology for Ontology Editors

Scope: Ontology validation and update by Ontology Editors (cf. Section 2.3)

Level: Sub function

Stakeholders and Interests:

Ontology Editor visualizes ontologies in order to approve, publish, or delete ontologies or ontology elements.

Preconditions:

- The user is logged in with a well-defined user role.
- System shows ontologies that ontology editors (with defined role) have authority to view. These ontologies are stable versions provided by Ontology Engineers.

Success Guarantee:

- Ontology is visualized as a graphic view.
- Elements in the selected status of the editorial workflow in the selected ontologies are highlighted and elements in other statuses are decolorized.

Fail Guarantee:

System fails to display the visualization and should provide feedback for the reason

Extensions Points: N/A

Main Success Scenario:

1. User selects one or more ontologies (modules).
2. User selects “Visualize Ontology” from menu or icon toolbar.
3. User selects one out of five status: “Draft”, “To be Approved”, “Approved, “To be Deleted”, and “Published”.
4. System visualizes selected ontologies in a window, showing all elements from the selected status high-lighted, while all other elements are greyed out.
5. User interacts with the graph to navigate to certain classes or properties.

Extensions:

Selected ontology is too big to visualize graphically:

1. System returns a warning indicating the problem.
2. User chooses a particular module, or root-class of an ontology or User chooses other view to visualize the ontologies such as a tree view.

Frequency of Occurrence: High

Miscellaneous:

User rights, incl. a log-in mechanism, and ontology-status information which is mandatory for this use-case is not covered by the prototype of the Fisheries Ontologies Lifecycle Management System.

UC-7.3 Visualize Mappings between Ontologies

Scope: Fisheries Ontologies Lifecycle Management System.

Level: Sub function.

Stakeholders and Interests:

Ontology Engineers and Ontology Editors need to visualize mappings between ontologies using a graphical view.

Preconditions: At least two ontologies (modules) are selected and opened.

Success Guarantee:

The relationship and mappings between selected ontologies (modules) are visualized graphically.

Fail Guarantee:

System fails to display the mapping visualization and should provide feedback for the reason.

Extensions Points: N/A

Main Success Scenario:

1. User selects option “visualize mapping” from menu or icon toolbar.
2. System displays selected ontologies (modules) in a one window indicating the relationships and mappings between them. This is a graphical view in which ontologies/modules are displayed as nodes.
3. User selects some ontology nodes and chooses from context menu or toolbar to open the “mapping tool”.
4. System displays detailed visualizations of the mappings between concept and properties of selected ontologies.

Extensions:

- User knows which ontologies are mapped onto each other.
    1. User opens “mapping tool”.
    2. User drags and drops ontologies into the mapping tool.
    3. System displays detailed visualizations of the mappings between concept and properties of selected ontologies.
- The visualization of mappings between ontologies (modules) for Ontology Editors needs to highlight/decolorize elements based on the selected status in the editorial workflow (cf. UC7-2).

Frequency of Occurrence: High

Miscellaneous:

User rights, incl. a log-in mechanism, and ontology-status information which is mandatory for this use-case is not covered by the prototype of the Fisheries Ontologies Lifecycle Management System.

UC-7.4 Print Visualization

Scope: Fisheries Ontologies Lifecycle Management System.

Level: Sub function.

Stakeholders and Interests:

Ontology Engineers and Ontology Editors need to print selected ontologies (or modules), when they visualize them in the Fisheries ontology lifecycle management system.

Preconditions: At least one ontology (module) is displayed by a visualization component.

Success Guarantee:

Selected ontology is printed out on a printer or saved into a file in a format such as PDF or PNG.

Fail Guarantee:

System fails to print/save the ontology visualization and displays proper error messages.

Extensions Points: N/A

Main Success Scenario:

1. U selects option “Print” or “Export to PDF/ PNG” from menu or icon toolbar, while ontology is visualized.
2. S opens dialog to choose printer or file to save into.
3. S prints/exports the currently displayed visualization.

Extensions:

Printer is not ready:

- S displays a warning dialog.
- U fixes problem and initiates the print again.

The specified file exists and is already opened (i.e. locked):

- S returns a warning.
- U is requested to close the file or choose another file name.

Frequency of Occurrence: Medium

Plugins/Tools

The basic requirements formulated in the use-cases above are already provided by the current implementation of NeOn toolkit (cf. “D6.7.1 Documentation of NeOn Toolkit” for more details). In the screenshots below we refer to other NeOn deliverables to indicate which plug-in already implements the functionality and to what extent.

UC-7.1 Visualize Ontology for Ontology Engineers

Ontology engineers need to be able to visualize one or more ontologies (modules) in different ways, such as tree view, graphic view, or form-based view. Multiple different graphical visualizations could emphasize different aspects of the model.

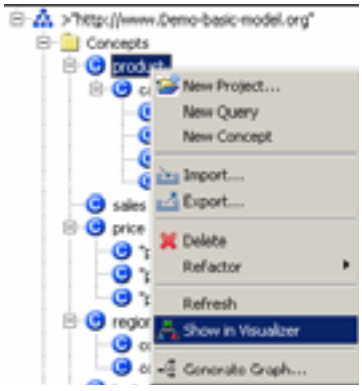


Figure 54. A tree view of an ontology (cf. D6.7.1 Documentation of NeOn Toolkit)

Figure 55. A graphical, interactive visualization of an ontology (cf. D6.7.1 Documentation of NeOn Toolkit)

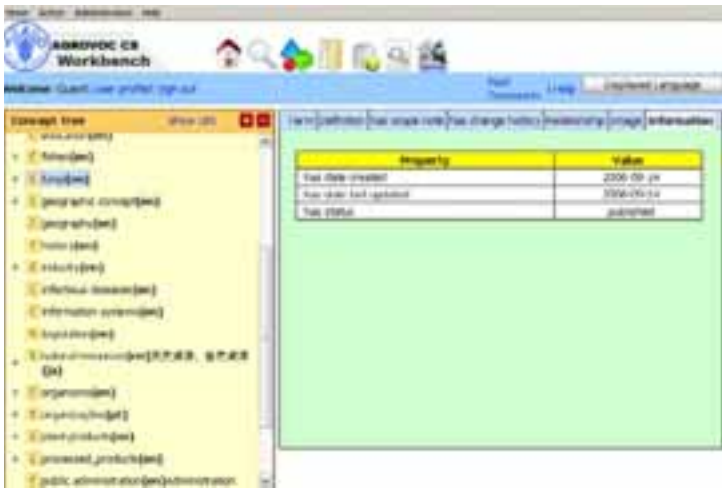


Figure 56. A form-based view of visualization (captured from AGROVOC concept server)

UC-7.2: Visualize Ontology for Ontology Editors

The main emphasis for visualization tools for Ontology Editors lies on respecting the role of the user and the status of elements. This is mandatory to support the editorial workflow. The screenshots below show possible visualizations that respect the status information of ontology elements.

In the following mock-up, the ontology navigator contains two elements (the concept “FileFolder” and the relation “hasSubfolder”) that are highlighted. They belong to the currently chosen status in the editorial workflow. All other elements were decolorized to denote that they are not in the selected status. Note, this screenshot is a GUI mock-up based on the current version of the NeOn Toolkit

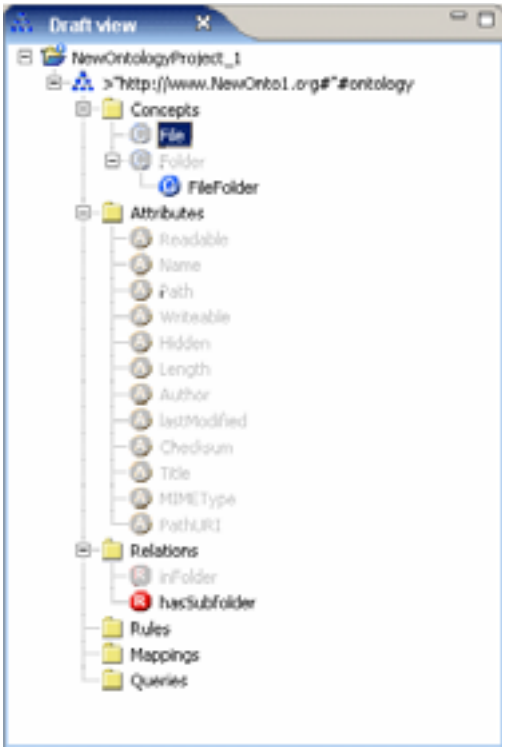


Figure 57. Ontology Navigator with highlighted and greyed out elements

UC-7.3 Visualize Mappings between Ontologies

The visualization of multiple ontologies and their relationships is not yet implemented in the NeOn Toolkit. A draft of a graphical view of such a visualization is presented in the following figure.

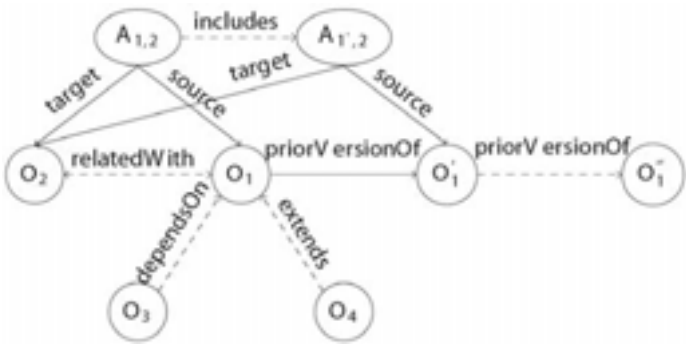


Figure 58. Possible visualization of the relationships between multiple (networked) ontologies (taken from "D1.1.1 Networked Ontology Model")

The visualisation of mappings on a concept and property level is already implemented by the Mapping Tool (cf. D6.3.1 First Implementation of critical Infrastructure Components).

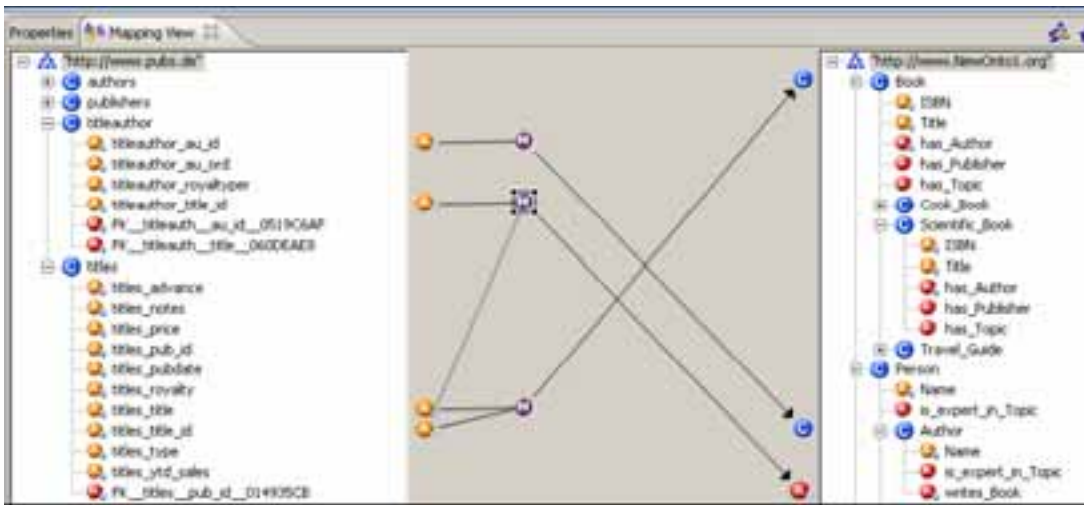


Figure 59. Combined tree and the graphic view to visualize mappings between ontologies

UC-8 Modularize

Overview

For a general overview please refer to section 3.2.6. UC-8.1 is about the creation of an empty module and its edition. UC-8.2 is about the module extraction and partitioning in a semi-automatic way. UC-8.3 is about combining modules.

Detailed description

UC-8: Modularize

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Primary Actor: Ontology Engineer

Stakeholders and Interest:

The ontology engineer wants to manage ontology modules for reusing issues and performance improvement.

Preconditions: The ontology engineer has been logged in and has the permissions modularizing.

Success Guarantee: Module is created or several modules are merged.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions points:

- UC-8.1: Create Ontology Module
- UC-8.2: Modularize Semi-Automatically
- UC-8.3: Merge Module

Main Success Scenario: UC-8.1

Extensions: UC-8.2 and UC-8.3

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-8.1: Create Ontology Module

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and Interest: The ontology engineer wants to create a module manually.

Preconditions: the Ontology Engineer has been logged in and has the permissions for creating the module.

Success Guarantee: Module is created.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions points: N/A

Main Success Scenario:

1. The user requests the creation of an empty module.
2. The module is edited by the user.

Extensions: N/A

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-8.2: Modularize Semi-Automatically

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and Interest:

The ontology engineer wants to create a module in a semi-automatic way.

Preconditions:

The ontology engineer has been logged in and has the permissions for creating the module.

Success Guarantee: Module is created.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions points: N/A

Main Success Scenario (or Basic Flow):

1. The user requests the automatic creation of modules.
2. The user is asked to choose between module extraction and partitioning.
3. The user is asked for the criterion for modularization.
4. The system provides an initial version of the module(s), together with their metadata.
5. The user edits the returned module(s), eventually going back to the third activity to refine the module(s) with further criteria.

Extensions (or Alternative Flows): N/A

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-8.3: Merge Module

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Ontology Engineer

Stakeholders and Interest: The Ontology Engineer wants to combine several modules.

Preconditions:

The ontology engineer has been logged in and has the permissions for merging the modules involved.

Success Guarantee: Modules are combined.

Fail Guarantee: the System remains as it was before the execution of the use case.

Extensions points: N/A

Main Success Scenario (or Basic Flow):

1. The user requests the creation of an empty module.
2. The user specifies that this module is a combination of other modules.
3. The user is asked to choose an operator for the combination.
4. The user is asked to select the modules to combine.
5. The system completes the module content and metadata in accordance.
6. The user edits the module, and can come back to the third activity to refine the definition of the module.

Extensions (or Alternative Flows): N/A

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-9 Manage Provenance and Statistics

Overview

For a general overview of these use cases please refer to section 3.2.9. This set of use cases will be implemented by integrating various plug-ins: UC-9.1, UC-9.2 and UC-9.3 will be implemented with OSYTER; and UC-9.4 will be implemented with Watson.

Detailed description

UC-9: Manage Provenance and Statistics

Scope: Ontology evaluation and validation on the editorial workflow

Level: User goal

Primary Actor: Ontology Editor

Stakeholders and Interest:

The ontology editor wants to retrieve information about provenance or statistics about the ontology or use.

Preconditions:

The ontology editor has been logged in and has the permissions for accessing to the provenance or the statistics.

Success Guarantee: Provenance or statistics are shown.

Fail Guarantee: The System remains as it was before the execution of the use case.

Extensions points:

- UC-9.1: Capture Ontology Changes
- UC-9.2: View Changes History
- UC-9.3: View Use Statistics
- UC-9.3: View Ontology Statistics

Main Success Scenario: UC-9.1

Extensions: UC-09.2, UC-9.3 and UC-9.4

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: High

Miscellaneous: N/A

UC-9.1 Capture Ontology Changes

Scope: Ontology evaluation and validation on the editorial workflow

Level: Sub function

Primary Actor: Ontology editor (subject expert, validator, and viewer)

Stakeholders and Interest:

Subject expert submits proposed changes to an ontology based on interactions with subject experts. Validator accepts/rejects proposed changes and copies changes into the production environment for external availability.

Preconditions: User logs in and selected ontology is opened.

Success Guarantee:

Proposed changes are logged into the registry using appropriate format (i.e. OMV extension) including relevant information (e.g. Timestamp, URI, User, Operation Executed, Rationale, Previous OWL/F-Logic code, Actual OWL/F-Logic code). The status of a proposed change is kept also in the registry at all times. Accepted changes are reflected in (new version of) the ontology.

Fail Guarantee:

The ontology remains unmodified (as it was originally). System returns an error message

Extensions points: N/A

Main Success Scenario:

1. The subject expert makes changes to the ontology (*draft* status).
2. The subject expert sends the proposed changes to be approved (*To Be Approved* status).
3. The validator validates proposed changes in status *To Be Approved* and either: Approves changes (send to *Approved* status) or Reject changes (reject to *Draft* status)
4. The validator sends changes to be deleted (*To Be Deleted* status).
5. The validator validates changes in status *To Be Deleted* and either: Destroy Reject deletion (send To *Be Approved* status)
6. The validator copy changes into production environment (*Published* status)

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: High

Miscellaneous: More detail in the flow is described in section 2.5

UC-9.2 View Changes History

Scope: Ontology evaluation and validation in the editorial workflow

Level: Sub function

Primary Actor: Ontology editor including subject expert, validator and reviewer

Stakeholders and Interest:

Ontology editor receives from the System the following information: date of creation/editing of an ontology element, history notes (to account for changes applied to a concept)).

Preconditions: User logs in and selected ontology is opened.

Success Guarantee: Editor successfully discovers and reviews ontology change information.

Fail Guarantee: System returns a warning for user to specify at least one criteria.

Extensions points: N/A

Main Success Scenario:

1. User searches for ontology based on certain criteria (i.e. status or element name)

2. User retrieves ontology change information
3. User visualizes ontology change information in more detail (e.g. date of creation/editing of an ontology element, history notes (to account for changes applied to a concept)).

Extensions: User doesn't specify to any criteria to search.

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-9.3 View use statistics

Scope: Ontology evaluation and validation in the editorial workflow

Level: Sub function

Primary Actor: Ontology editor

Stakeholders and Interest:

Ontology editor receives salient information about the visualized ontologies (e.g. provenance, which system they are used by, by whom they are edited, frequency of changes, fragment/domain of the ontology changed at fastest rate).

Preconditions:

User has been logged in and has the permissions for searching ontologies and visualizes its information. Selected ontology has been opened.

Success Guarantee: User successfully discovers and reviews ontology information.

Fail Guarantee: The ontology remains unmodified (as it was originally). System returns a warning

Extensions points: N/A

Main Success Scenario:

1. User logs in
2. User selects one or more ontologies/modules.
3. User searches for ontology based on certain criteria (i.e. metadata).
4. User retrieves ontology information.
5. User visualizes use statistics.

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-9.4 View Ontology Statistics

Scope: Ontology evaluation and validation in the editorial workflow

Level: Sub function

Primary Actor: Ontology editor

Stakeholders and interests: Ontology editor: wants to consult ontology statistics.

Preconditions:

User logs in and has the permissions for viewing the ontology. Selected ontology is opened.

Success Guarantee: Ontology statistics are returned to the user.

Fail Guarantee: System returns an error message, when no statistics are returned to the user.

Extensions Points: N/A

Main Success Scenario:

1. User choose " view statistics" in the menu or icon bar
2. System returns a statistic item to the user.

Extensions: System returns an error message.

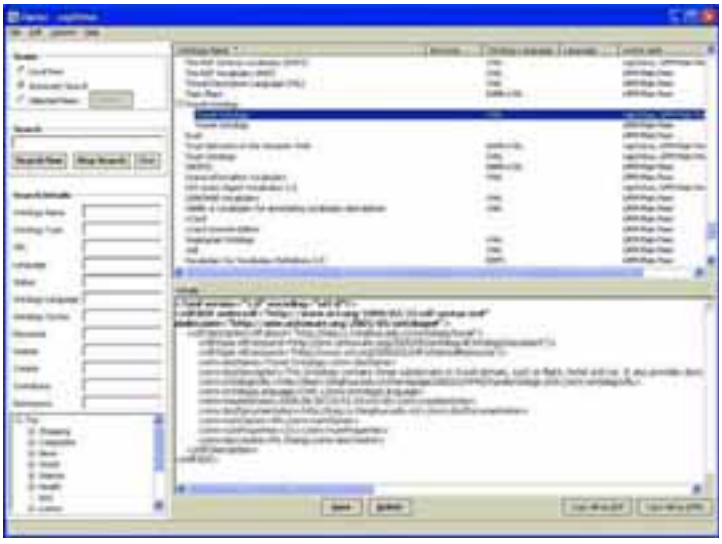
Special Requirements: N/A

Frequency of Occurrence: medium

Miscellaneous: N/A

Plugins/Tools

The following figure shows the GUI for the UC-9.1, UC-9.2 and UC-9.3 use cases.



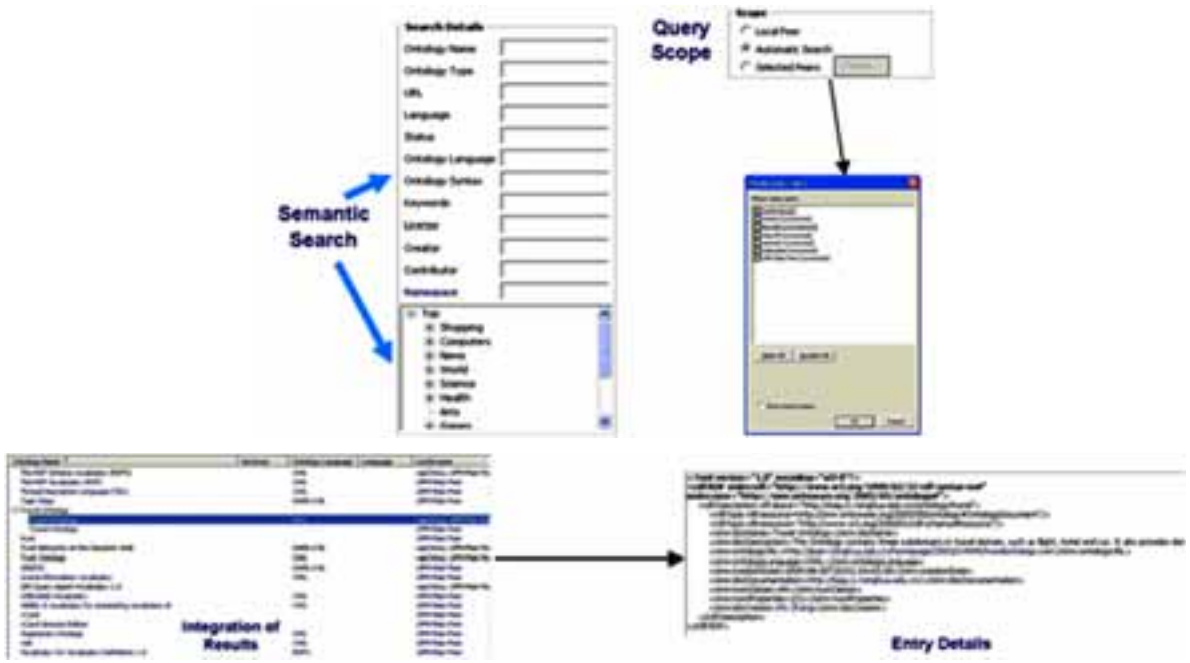


Figure 60. GUI prototype of UC-9.1, UC-9.2, and UC9-3

For UC-9.4, if the user wants to consult several statistics about the ontology then he can right click over the whole ontology and clicking an option inside a contextual menu the system will show:

- Depth.
- Number of child nodes.
- Number of relations.
- Number of properties.
- Number of concepts per “branch”.

OYSTER details

The Ontology Registry, Oyster, is a Peer-to-Peer system that provides services for storage, cataloguing, discovery, management, and retrieval of ontology (and related) metadata definitions. It exploits semantic web techniques in order to provide a solution for exchanging and re-using ontologies. To achieve this, Oyster implements the proposal for a metadata standard, so called Ontology Metadata Vocabulary (OMV) as the way to describe ontologies and their lifecycle (e.g. changes).

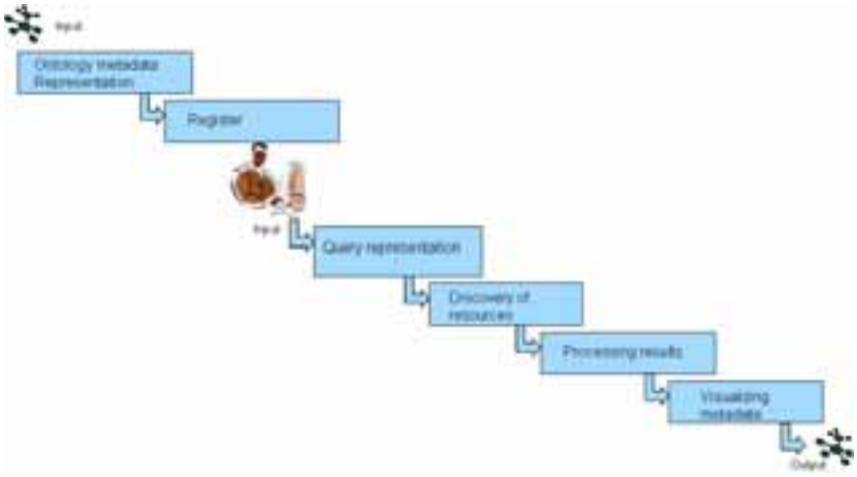


Figure 61. The process of management of provenance and statistics

The process for supporting the Management of Provenance and Statistics:

1. *Ontology metadata representation* The first step is to formally represent metadata about ontologies. For this step we use the ontology metadata vocabulary (OMV).
2. *Register* The metadata is stored into the register for later reuse.
3. *Query representation* The user sends a query to find ontologies that fulfil certain requirements.
4. *Discovery of resources* In order to answer on the query, all relevant distributed resources are searched.
5. *Processing of results:* Results received are integrated (e.g. duplicates are identified) and sent to for visualization.
6. *Visualizing metadata:* Information is displayed (e.g. provenance, which system they are used by, by whom they are edited, frequency of changes, fragment/domain of the ontology changed at fastest rate).

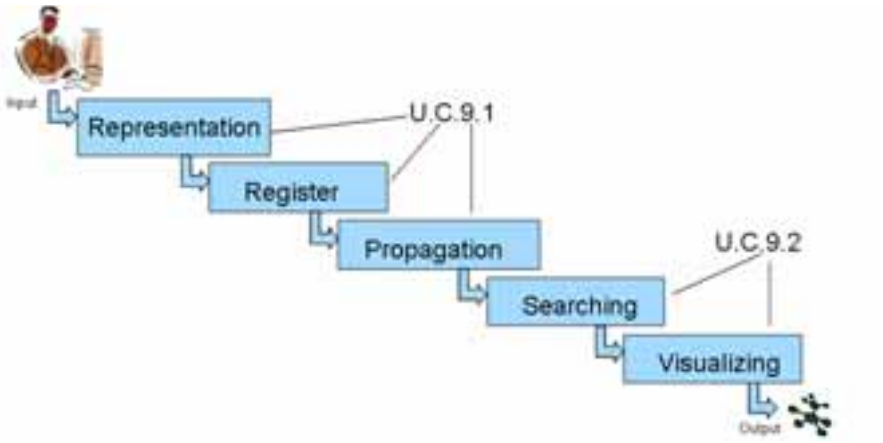


Figure 62. The process for management provenance and statistics in the editorial workflow

The process for supporting the Management Provenance and Statistics (and editorial workflow):

1. *Representation* The first step is to formally represent the changes proposed by Ontology editors. For this step we use the ontology metadata vocabulary (OMV) extension for ontology changes. All changes are logged with relevant information (e.g. Timestamp, URI, User, Operation Executed, Rationale, Previous OWL/F-Logic code, Actual OWL/F-Logic code).
2. *Register* The changes are stored into the register for later reuse.
3. *Propagation* One of the goals is to support the process in a distributed environment, therefore in this step, changes are propagated (i.e. synchronized) to every affected artefacts.
4. *Searching* This is a necessary part of the process in order to validate and query all the proposed changes.
5. *Visualizing* The editorial workflow needs to be visualized, regarding four data views based on user roles.
    a. *Approved view: approved information only.*
    b. *Draft view: approved information plus changes made by the current editor with difference between the two states clearly visible*

c. *To be approved view: approved information plus all unapproved changes by all editors with difference between the two states clearly visible*
d. *To be deleted view: approved information plus suggested deletion with difference between the two states clearly visible*

The set of components that will solve the use case according to the architecture proposed in WP6.

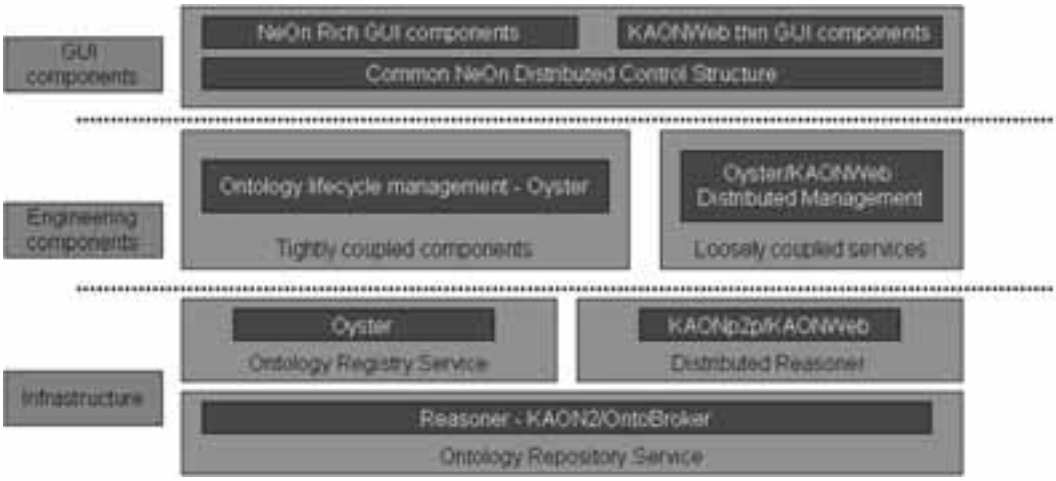


Figure 63. Proposed component architecture

Implementation Details

The Oyster system (Oyster system is freely available under http://ontoware.org/projects/oyster2/) was designed using a service-oriented approach, and it provides a well defined API. Accessing the registry functionalities can be done using directly the API within any application, invoking the web service provided or using the included java-based GUI as a client for the distributed registry.

In Oyster, ontologies are used extensively in order to provide its main functions (register metadata, formulating queries, routing queries and processing answers):

- *Creating and importing metadata*: Oyster2 enables users to create metadata about ontologies manually, as well as to import ontology files and to automatically extract the ontology metadata available, letting the user to fill in missing values. The ontology metadata entries are aligned and formally represented according to two ontologies: (1) the OMV ontology, (2) a topic hierarchy (i.e. the DMOZ topic hierarchy), which describes specific categories of subjects to define the domain of the ontology.
- *Formulating queries*: A user can search for ontologies using simple keyword searches, or using more advanced, semantic searches. Here, queries are formulated in terms of these two ontologies. This means queries can refer to fields like name, acronym, ontology language, etc. or queries may refer to specific topic terms.
- *Routing queries*: A user may query a single specific peer (e.g. their own computer, because they can have many ontologies stored locally and finding the right one for a specific task can be time consuming, or users may want to query another peer in particular because this peer is a known big provider of information), or a specific set of peers (e.g. all the member of a specific organization), or the entire network of peers (e.g. when the user has no idea where to search), in which case queries are routed automatically in the network.
- *Processing results*: Finally, results matching a query are presented in a result list. The answer of a query might be very large, and contain many duplicates due to the distributed nature and potentially large size of the P2P network. Such duplicates might not be exactly copies because the semi structured nature of the metadata, so the ontologies are used

again to measure the semantic similarity between different answers and to remove apparent duplicates.

The high-level design of the architecture of a single Oyster node in the Peer-to-Peer system is shown in the following figure.

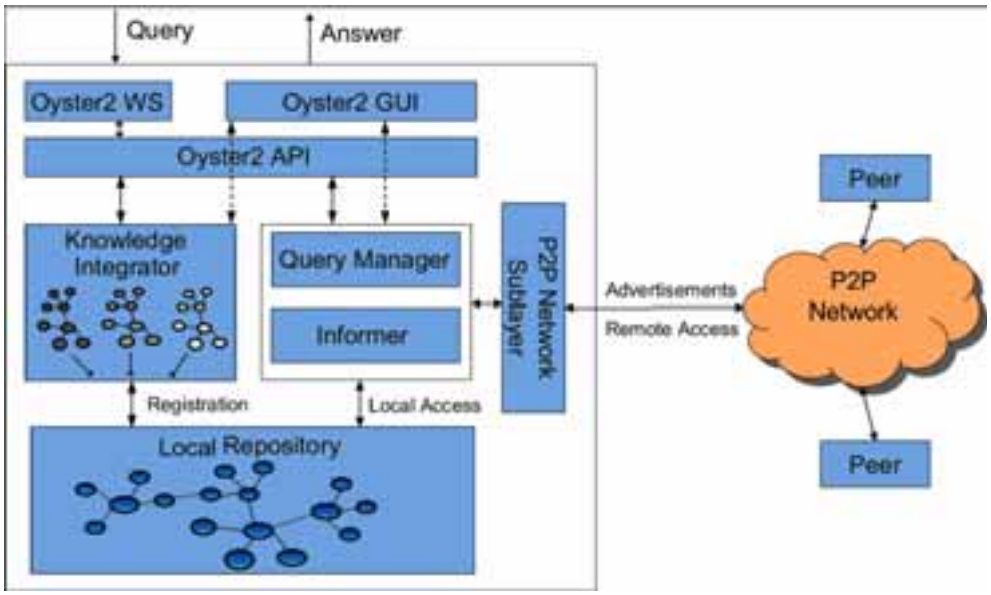


Figure 64. OYSTER architecture

- The *Local Repository* of a node contains the metadata about ontologies that it provides to the network. It supports query formulation and processing and provides the information for peer selection. In Oyster, the Local Repository is based on KAON2 and it supports SPARQL as its query language.
- The *Knowledge Integrator* component is responsible for the extraction and integration of knowledge sources (i.e. ontologies) into the Local Repository. Oyster supports automatic extraction of metadata for OWL, DAML+OIL , and RDF-S ontology languages. This component is also in charge of how duplicate query results are detected and merged.
- The *Query Manager* is the component responsible for the coordination of the process of distributing queries. It receives queries from the user interface, API or from other peers. Either way it tries to answer the query or distribute it further according to the content of the query. The decision to which peers a query should be sent is based on the scope of the query (i.e. a specific set of peers or entire network) and optionally on the knowledge about the expertise of other peers.
- The *Informer* component is in charge of proactively advertising the available knowledge of a Peer in the Peer-to-Peer network and to discover peers along with their expertise. This is realized by sending advertisements about the expertise of a peer. In Oyster, these expertise descriptions contain a set of topics (i.e. ontology domains) that the peer is an expert in. Peers may accept these advertisements, thus creating a semantic link to the other peer. These semantic links form a semantic topology, which is the basis for intelligent query routing.
- The *Peer-to-Peer network sub-layer* is the component responsible for the network communication between peers. It provides communication services for the data exchange with remote nodes, i.e. to propagate advertisement messages and to realize the access to remote repositories. In Oyster, we rely on an RMI-based implementation; however, other communication protocols would be possible as well.

The API, WS and GUI components provide the required interfaces to expose the registry services.

UC-10 Populate from Text

Overview

For a general overview of these use cases please refer to section 3.2.10. The implementation of this use case is based on the results of the experiments on ontology learning and population carried out within T7.3 and evaluated in D7.3.1 (M20). WP7 case study has already identified several tools for this use case. But selected tools will be integrated in the lifecycle management system, only if the tools and methods are found appropriate and efficient to assist on the networked ontologies maintenance.

Detailed description

UC-10 Populate from text

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Primary Actor: Ontology Editor

Stakeholders and Interest:

The ontology editor wants to generate a domain specific ontology from scratch.

Preconditions: the ontology editor has been logged in.

Success Guarantee:

A domain specific ontology is correctly generated and a list of domain entities/concepts is provided to the user. It includes domain specific concepts (e.g. researcher) as well as abstract concepts (e.g. person).

Fail Guarantee: The system remains as it was before the execution of the use case.

Extensions points: N/A

Main Success Scenario (or Basic Flow):

1. The ontology editor informs the system that he wants to create a new ontology.
2. The ontology editor type a query/submit a domain specific text corpus to the system.
3. The system creates a proposal of domain ontology and sends a log to the ontology editor.

Special requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

Plugins/Tools

In this section we will include the description of the tool DidOnE; but GATE, Text2Onto, LeXO, LeDa and Ontogen are also been tested in T7.3.

The aim of the DiDOnE component is to assist ontology engineers during the development of domain ontologies by suggesting her/him domain-specific taxonomies and instances. The input will be either domain-specific texts or a query in a standard Information Retrieval paradigm. DiDOnE

will be able to automatically build an OWL (DL) ontology. DiDOnE relies on the NeOn metamodel and implements a Latent Semantic Analysis (LSA) technique followed by a refinement based on WordNet and on NeOn ontology design patterns.

The system will provide users with a wizard-like interface. The wizard will allow users to select the domain of interest in two different ways:

- By simply typing a query (analogously to the common practice of querying search engines).
- By submitting a corpus of domain specific documents.

The output of DiDOnE will be an OWL (DL) domain ontology. The ontology will be expressed according to appropriate NeOn ontology design patterns and will represent:

- Taxonomies of domain specific concepts.
- Instances of interest in the domain.

This structure will be imported by the ontology engineering environment as well as standard ontologies.

Due to the fully automatic process, the expected quality is not high enough for the ontology engineering purposes. Therefore, the generated ontology will be manually checked (and possibly corrected or restructured) by the ontology engineer. Anyhow, we expect a sensible development time reduction.

At the moment we cannot provide snapshots of the GUI, since we are still in the experimental phase of the component development.

The component relies on the combination of two main knowledge sources:

1. WordNet (as an open domain , large coverage core taxonomy)
2. Latent Semantic Analysis (LSA)

The main idea is to first apply LSA to extract a domain terminology from a large open domain corpus, as an answer to the user query. This process is implemented by adopting similarity metrics in geometrical spaces induced from corpora.

Then, the algorithm leverages a Conceptual Density algorithm to project the inferred terms into WordNet to identify domain specific sub-regions in it that can be regarded as lexicalized core ontologies for the domain of interest. The overall approach allows achieving the goals of lexical ambiguity resolution and ontology pruning, and offers an online solution to the problem of domain adaptation of lexical resources.

Finally, the so generated taxonomy and terms are converted into an appropriate OWL-RDF format according to a predefined meta-model.

UC-11 Evaluate and Validate Ontology

Overview

For a general overview of these use cases please refer to section 3.2.11.

Detailed description

UC-11 Evaluate and Validate Ontology

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Primary Actor: Authorized User

Stakeholders and interests: The user wants to evaluate and validate the ontology.

Preconditions: The user has been logged in and has the permissions for viewing the ontology.

Success Guarantee: Validation/Evaluation is returned to the user.

Fail Guarantee: No validation/evaluation is returned to the user.

Extensions Points: N/A

Main Success Scenario: UC-11.1

Extensions: UC-11.2

Special Requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Low

Miscellaneous: N/A

UC-11.1 Check for duplicates within the ontology

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Authorized User

Stakeholders and interests: The user wants to check for duplicates within the ontology.

Preconditions: The user has been logged in and has the permissions for viewing the ontology.

Success Guarantee: Duplicates are returned to the user.

Fail Guarantee: No duplicates are returned to the user.

Extensions Points: N/A

Main Success Scenario:

1. The user informs the System that he wants to check for duplicates inside a given ontology.
2. The System returns to the User a group of duplicate concepts.
3. The System returns a duplicate concept to the User.

The System repeats step 3 until there are not more duplicates concepts in the same group of similarity.

The System repeats step 2-3 until there are not more groups of similarity.

Extensions: N/A

Special Requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Low

Miscellaneous: N/A

UC-11.2 Compare Ontologies Check for duplicates within the ontology

Scope: Fisheries Ontologies Lifecycle Management System

Level: Sub function

Primary Actor: Authorized User

Stakeholders and interests: The user wants to check for duplicates within the ontology.

Preconditions: The user has been logged in and has the permissions for viewing the ontology.

Success Guarantee: Duplicates are returned to the user.

Fail Guarantee: No duplicates are returned to the user.

Extensions Points: N/A

Main Success Scenario:

1. The User informs the System that he wants to check for duplicates inside a given ontology.
2. The System returns to the User a group of duplicate concepts.
3. The System returns a duplicate concept to the User.

The System repeats step 3 until there are not more duplicates concepts in the same group of similarity.

The System repeats step 2-3 until there are not more groups of similarity.

Extensions: N/A

Special Requirements: N/A

Technology and Data Variation List: N/A

Frequency of Occurrence: Low

Miscellaneous: N/A

UC-12 Undo

Overview

For a general overview of these use cases please refer to section 3.2.12. For certain actions, e.g. modelling rules, this use case will be implemented by NeOn tool kit (core).

Detailed description

UC-12 Undo

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Stakeholders and interests: The user wants to undo an operation.

Primary actor: Ontology engineer and ontology editor.

Preconditions: user logs in and creates an event.

Success Guarantee: System rollbacks the previous stage, before user creates the event.

Fail Guarantee: System returns a warning

Extensions Points: System returns a warning, when user tries to undo without doing any event.

Main Success Scenario:

1. User selects “undo” from menu or icon toolbar
2. System rollbacks the previous stage, before the event is executed.

Extensions: System returns an error.

Special Requirements: N/A

Frequency of Occurrence: High

Miscellaneous: N/A

UC-13 Obtain Documentation

Overview

For a general overview of these use cases please refer to section 3.2.13. This use case will be implemented taking OWLDoc as a reference.

Detailed description

UC-13 Obtain Documentation

Scope: Fisheries Ontologies Lifecycle Management System

Level: User goal

Stakeholders and interests:

The user wants to generate the documentation about an ontology or module.

Primary actor: Authorized user.

Preconditions: User logs in and selected ontology is opened.

Success Guarantee: All elements of selected ontology/module are documented.

Fail Guarantee:

System remains as it was. Previous documentation about the ontology or module (if any) remains as before the execution of the use case.

Extensions Points: N/A

Main Success Scenario:

1. The user request the generation of the documentation about the ontology/module in use.
2. User is requested to specify documents style and contents based on his/her interest
3. System generates the documentation.

Extensions: N/A.

Special Requirements: N/A

Technology and Data Variation List:

All documents are encoded using UTF-8 (All characters in each language - at least English, French, Spanish, Arabic, Chinese, and Russian - are shown correctly.

Frequency of Occurrence: Medium

Miscellaneous: N/A

Plugins/Tools

OWLDoc is a tool that generates JavaDoc style html page documentation for an OWL ontology. OWLDoc has been implemented so that it may be used with Protege-OWL in order to generate documentation for OWL ontologies. We will integrate this tool into the NeOn Toolkit. OWLDoc offers various configuration options. For example it is possible to include the Abstract Syntax representation of classes, properties and individuals. OWLDoc also supports the possibility of localizing the labels used in the pages it generates - different languages can be 'plugged' in.

Figure 65. GUI prototype of Obtaining documents. Note that this GUI is captured by OWLDoc

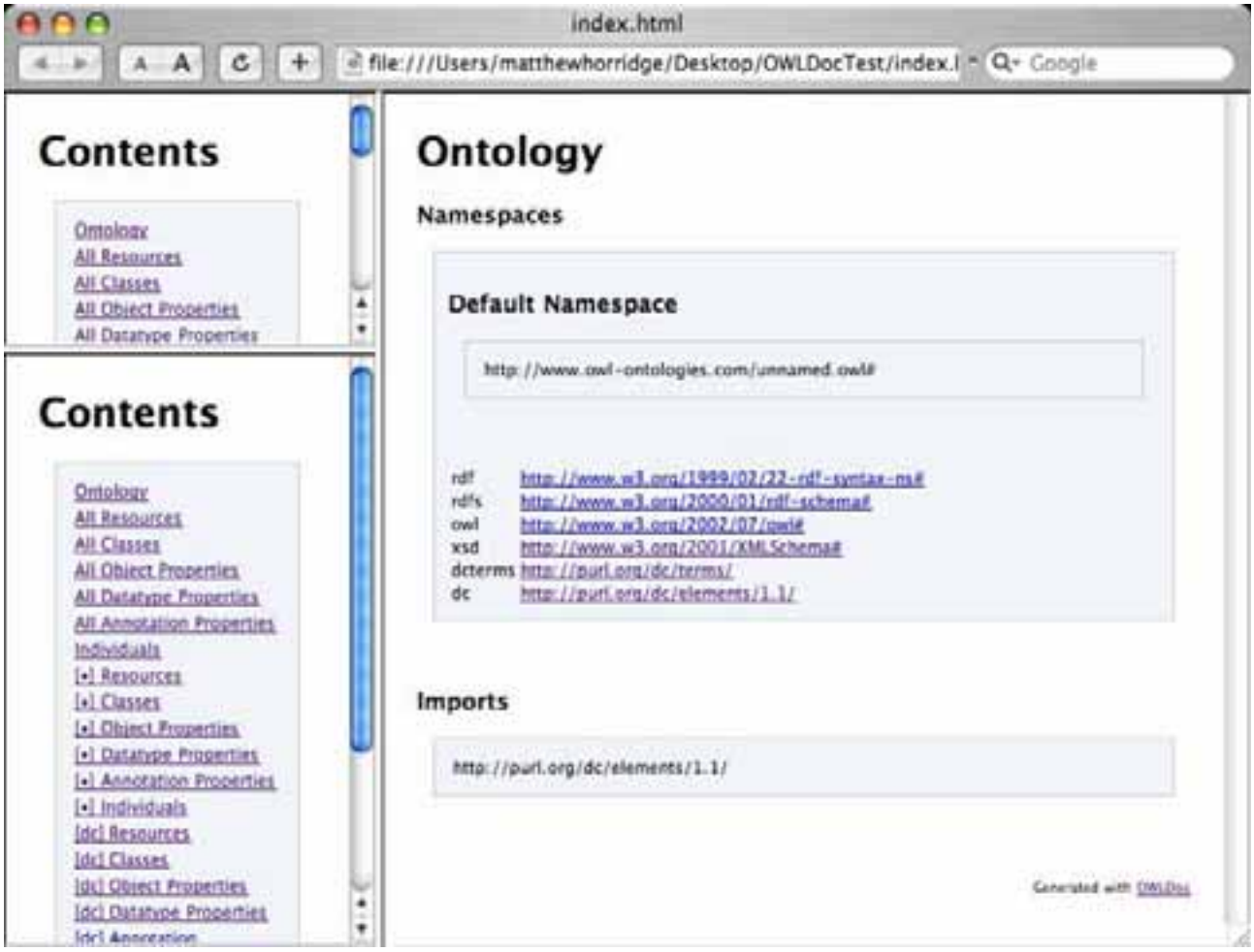


Figure 66. Example ontology documentation generated by OWLDoc

UC-14 User Rights

Overview

For a general overview of these use cases please refer to section 3.2.14.

Detailed description

UC-14.2: Manage Access Rights

Scope: Editorial Workflow

Level: Sub function

Primary Actor: Ontology Editor, Ontology Administrator

Stakeholders and Interest:

Key stakeholders are: the users accessing a particular ontology for the purpose of viewing and/or editing (further referred as Subjects); the users authoring or otherwise managing access to a particular ontology (further referred as Right Owners)

The primary interest is to satisfy the principle of minimal authority that states that every user of the system should operate using the least set of privileges necessary to complete the job.

Preconditions:

- The Ontology can be split into modules, and
- Each Ontology Module can be uniquely identified with the URI notation, and
- Subjects access the Ontology via an Ontology Repository that is capable of applying Rights

Success Guarantee:

Subjects only interact with the appropriate portions of the Ontology.

Fail Guarantee:

Subjects are allowed to store amendments to the Ontology, to which they have no appropriate Access Rights.

Main Success Scenario (actually Basic Sub-Flows):

- Assignment of Access Rights to the Ontology
- Access request for the Ontology with Access Rights

Extensions (or Alternative Flows):

- Delegation of Access Rights to third parties
- Revocation of Access Rights from third parties

Technology and Data Variation List: N/A

Frequency of Occurrence: medium

Miscellaneous: N/A

UC-14.2.1: Assign Access Rights for Ontology

Scope: Editorial and Administrative Workflows

Level: Sub function

Primary Actor: Ontology Editor, Ontology Administrator

Stakeholders and Interest:

Key stakeholders are: the users accessing a particular ontology for the purpose of viewing and/or editing (further referred as Subjects); the users authoring or otherwise managing access to a particular ontology (further referred as Right Owners)

The primary interest is to satisfy the principle of minimal authority that states that every user of the system should operate using the least set of privileges necessary to complete the job.

Preconditions:

- The Ontology can be split into modules, and
- Each Ontology Module can be uniquely identified with the URI notation, and
- The Ontology in question will be accessed via an Ontology Repository that is capable of applying and storing Access Rights

Success Guarantee:

Subjects' access is stored with respect to (the appropriate portions of) the Ontology.

Fail Guarantee:

Subjects are unable to express or store Access Rights to the Ontology.

Main Success Scenario (or Basic Flows):

1. The Right Owner specifies Ontology (or Ontology Module) URI with which s/he works.
2. The Right Owner selects one or more Subjects who would gain (some) Access Rights.
3. The Right Owner selects which of the Access Rights apply to each Subject.
4. The created triple [Subject , Access Right / Authority , Object / Ontology Module] is stored in the Ontology Repository

Extensions (or Alternative Flows):

1a. The Right Owner has the Ontology open and selects appropriate nodes, sub-sets, etc. within the Ontology, to which s/he wants to apply the Access Rights.

1b. A (new) Ontology Module is created to encompass the selected nodes in the Ontology in a consistent manner (incl. unique URI, definition, etc.)

2a. Subjects are specified by their unique identifiers directly.

2b. Subjects are obtained from the Repository based on User Group definitions, or may be defined based upon selecting a particular Workflow, and similarly

3a. See also the UC for access rights delegation.

Technology and Data Variation List: N/A

Frequency of Occurrence: medium

Miscellaneous: N/A

UC-14.2.2: Apply Access Rights to the Accessed Ontology

Scope: Editorial Workflow

Level: Sub function

Primary Actor: Ontology Editor, Ontology Viewer

Stakeholders and Interest:

Key stakeholders are: the users accessing a particular ontology for the purpose of viewing and/or editing (further referred as Subjects); the users authoring or otherwise managing access to a particular ontology (further referred as Right Owners)

The primary interest is to satisfy the principle of minimal authority that states that every user of the system should operate using the least set of privileges necessary to complete the job.

Preconditions:

- The Ontology can be split into modules, and
- Each Ontology Module can be uniquely identified with the URI notation, and
- Subjects access the Ontology via an Ontology Repository that is capable of applying Rights

Success Guarantee:

Subjects only interact with the appropriate portions of the Ontology.

Fail Guarantee:

Subjects are allowed to store amendments to the Ontology, to which they have no appropriate Access Rights.

Main Success Scenario (actually Basic Sub-Flows):

1. Subject identifies her- or himself to the Ontology Repository (a personal hash might be used to by pass the need to log in).
2. Subject makes a request for the Ontology using its URI (e.g. http://foo.org/data.owl)
3. The Repository checks the level of Access Rights and presents the Subject with so-called Ontology Access URI, which upon clicking opens up only those modules from the Ontology, to which the Subject has appropriate Access Rights.

Extensions (or Alternative Flows):

4. Subject makes modification to the Ontology and wants to store it in the Repository. To this extent the Subject must also possess a valid Ontology Access URI with the ‘Edit/Modify’ Access Right, and the communication with the Repository runs on the level of these ‘Access URI-s’

4a. Ideally, the Access URI-s may be processed and checked on background, using an idea of Keychain, which may contain all applicable access keys for a give Subject.

Technology and Data Variation List: N/A

Frequency of Occurrence: high to medium

Miscellaneous: N/A

UC-14.2.3: Delegate Access Rights for Ontology

Scope: Editorial and Administrative Workflows

Level: Sub function

Primary Actor: Ontology Editor, Ontology Administrator

Stakeholders and Interest:

Key stakeholders are: the users accessing a particular ontology for the purpose of viewing and/or editing (further referred as Subjects); the users authoring or otherwise managing access to a particular ontology (further referred as Right Owners)

The primary interest is to satisfy the principle of minimal authority that states that every user of the system should operate using the least set of privileges necessary to complete the job.

Preconditions:

- The Ontology can be split into modules, and
- Each Ontology Module can be uniquely identified with the URI notation, and
- The Ontology in question will be accessed via an Ontology Repository that is capable of applying and storing Access Rights, and
- The Right Owner possesses Access Rights (Authority) at least on the level s/he intends to delegate.

Success Guarantee:

Subjects' access is stored with respect to (the appropriate portions of) the Ontology.

Fail Guarantee:

Subjects are unable to express or store Access Rights to the Ontology.

Main Success Scenario (or Basic Flows):

1. The Right Owner specifies the Object (usually Ontology or Ontology Module, but also a Subject) URI to which s/he intends to delegate the Access Rights.
2. The Right Owner selects one or more Subjects who would be the Recipients of (some of his or her) existing Access Rights.
3. The Right Owner selects which of his or her existing Access Rights will be delegated to each Subject.
4. The created triple [Subject, Access Right / Authority, Object / Ontology Module] is stored in the Ontology Repository along with the information who delegated it in the first place.
5. An object-level representation of the Access Right Delegation is created in the Repository, which comprises the new Access URI that can be added to the Subject – Recipient's Keychain, and also the Right Owner's Delegation Access Pointer that is used in revocation

Extensions (or Alternative Flows):

1ai. The Right Owner has the Ontology open and selects appropriate nodes, sub-sets, etc. within the Ontology, to which s/he wants to apply the Access Rights.

1aii. A (new) Ontology Module is created to encompass the selected nodes in the Ontology in a consistent manner (incl. unique URI, definition, etc.)

1b. The Right Owner searches for the Objects (e.g. the subjects and/or ontologies) in an appropriate repository

2a. Subjects are specified by their unique identifiers directly.

2b. Subjects are obtained from the Repository based on User Group definitions, or may be defined based upon selecting a particular Workflow, and similarly

3a. See also the UC for access rights delegation.

Technology and Data Variation List: N/A

Frequency of Occurrence: medium to high

Miscellaneous: N/A

UC-14.2.4: Revoke Access Rights for Ontology

Scope: Editorial and Administrative Workflows

Level: Sub function

Primary Actor: Ontology Editor, Ontology Administrator

Stakeholders and Interest:

Key stakeholders are: the users accessing a particular ontology for the purpose of viewing and/or editing (further referred as Subjects); the users authoring or otherwise managing access to a particular ontology (further referred as Right Owners)

The primary interest is to satisfy the principle of minimal authority that states that every user of the system should operate using the least set of privileges necessary to complete the job.

Preconditions:

- The Ontology can be split into modules, and
- Each Ontology Module can be uniquely identified with the URI notation, and
- The Ontology in question will be accessed via an Ontology Repository that is capable of applying and storing Access Rights
- The Right Owner still possesses Access Rights (Authority) at least on the level s/he intends to revoke (i.e. there is no other Rights Owner who already revoked the Authority from the Actor).

Success Guarantee:

Subjects' access is stored with respect to (the appropriate portions of) the Ontology.

Fail Guarantee:

Subjects are unable to express or store Access Rights to the Ontology.

Main Success Scenario (or Basic Flows):

1. The Right Owner specifies Ontology (or Ontology Module) URI with which s/he works.
2. The Right Owner selects one or more Delegation Access Pointers known for that Ontology.
3. The Right Owner cancels one or more Pointers, thus revoking the Access Right / Authority from the Subject owning the actual Access URI
4. The operation is committed to the Ontology Repository

Extensions (or Alternative Flows):

1a. The Right Owner searches for the Objects (e.g. the subjects and/or ontologies) in an appropriate repository to obtain their Access URI-s

2ai. The Right Owner specifies or searches Subjects for whom s/he wants to revoke Authority

2aii. Appropriate Delegation Objects are retrieved and the respective Delegation Access Pointers are provided to the Right Owner in order to revoke them.

5. The Subject whose Authority (Access Right) to an Object was revoked is appropriately informed (e.g. by an email or message) that certain Access URI-s have been removed from his or her Keychain.

Technology and Data Variation List: N/A

Frequency of Occurrence: medium

Miscellaneous: N/A

UC-15 Editorial Workflow Related Use Cases

Overview

For a general overview of these use cases please refer to sections 2.52 and 3.2.15 of this deliverable. This set of use cases will be implemented by the NeOn Toolkit.

Detailed description

UC-15 Editorial Workflow Related Use Cases

Scope: Ontology validation and update

Level: User goal

Stakeholders and interests:

Ontology editors (Subject Experts, Validators, and Viewers) need to validate and, if necessary, update a stable version of an ontology previously developed by ontology engineers.

Preconditions:

User logs in and has user rights and an assigned role allowed for participating in the editorial workflow.

Success Guarantee: User completes his/her task within the editorial workflow.

Fail Guarantee:

System returns a warning message and remains as it was before the execution of this use case.

Main Success Scenario: The basic flow depends on the UC-15 sub use cases.

Extensions:

Special Requirements:

This use case and sub use cases are strongly related to UC-7.2 Visualize ontology for ontology editors.

The system would need to activate/inactive menus or icon bars, based on user's role in the editorial workflow. For example, Subject Expert is only a primary actor on UC-15.1.1, UC-15.1.2, UC-15.3.1, and UC-15.3.6. So, the system only activates "insert", "update", and "change status (send to *To be Approved* and send to *Delete*)" in menu bar or icon bar, when a Subject Expert logs in.

Frequency of Occurrence: High

UC-15.1: Edit Ontology Element

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Subject Experts and Validators need to edit elements in a selected ontology or module.

Preconditions:

User logs in and opens an ontology or module, which he/she has been assigned as Subject Expert or Validator role for the selected ontology (module).

Success Guarantee: Elements are edited.

Fail Guarantee: No element is edited.

Extensions Points: UC-15.1.1 and UC-15.1.2

Main Success Scenario:

1. User chooses “update an ontology element” in menu or icon bar in right location (UC-15.1.2)
2. User updates the selected elements

Extensions: UC-15.1.1

Special Requirements: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-15.1.1: Insert an ontology element

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests: Subject Experts need to insert elements in the selected ontology.

Preconditions:

User logs in and opens an ontology or module, which he/she has assigned a Subject Expert role for the selected ontology (module).

Success Guarantee:

Elements are inserted and the “Draft” status is automatically assigned to the elements by the system.

Fail Guarantee: No element is inserted,

Extensions Points: N/A

Main Success Scenario:

a. Add class (or concept)

1. User selects location in the ontology where to add a new class
2. User selects option “add class” from menu or icon
3. System adds class
4. User is requested to add the following pieces of information: class name. User may also add other information such as restriction, axiom, and/or comments.

Extensions:

b. Add property (or attribute/relation)

1. User selects option “add property” from menu or icon toolbar
2. User is requested to add the following pieces of information: property name, property type, domain and range.

c. Add instance

1. User selects a class
2. User selects option “add instance” from menu or icon toolbar

3. User is requested to add the following pieces of information: instance name, property values.

Special Requirements: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-15.1.2: Update an ontology element

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Subject Experts need to update elements in the “Draft”, or “Approved” status.

Validators need to update elements in the “To be Approved” and “Approved” status.

Preconditions:

User logs in and opens an ontology or module, which he/she has assigned either Subject Expert or Validator role for the selected ontology (module).

Success Guarantee:

Elements are updated.

If users are Subject experts, system needs to reset the status of the updated element to “Draft” status.

If users are Validators, system keeps the same status before the update.

Fail Guarantee: No element is updated

Main Success Scenario:

1. User selects a status of elements to update, from menu bar or icon bar.
2. System shows a list of elements in the selected status.
3. User clicks an element or chooses “update” in the menu or icon bar.
4. System shows element in detail.
5. User updates the element
6. System updates the element in the repository.

Extensions:

If updated elements already exist in the selected ontology, system should return an error message and system doesn’t update any element. User is requested to update it without duplication.

Special Requirements: Updating any element by Subject Experts and Validators triggers to start an editorial workflow.

Frequency of Occurrence: Medium

UC-15.2: Delete an ontology element

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Subject Experts and Validators need to propose deletion of elements.

Preconditions:

User logs in and opens an ontology or module, which he/she has assigned an Ontology Editor role for the selected ontology (module). The user chooses “Approved" view to see the elements to propose deletion.

Success Guarantee:

Status of Selected element is automatically assigned from “Approved” to “To be deleted”.

Fail Guarantee: No element is sent to “To be Deleted” status.

Extensions Points: N/A

Main Success Scenario:

1. User selects an element to propose deletion.
2. User chooses “Delete” in the menu or icon bar.
3. System changes status of the selected element from “Approved” status into “To Be Deleted” status.
4. The list of elements in the “Approved” status is updated.

Extensions: N/A

Special Requirements:

This is not a definitive action, and only Validators are able to definitely destroy an element in the “To be deleted” status (See more detail in 15.3.6).

Miscellaneous:

Any current activated window showing a list of “To be Deleted” might be refreshed by adding the new element, after this use case is executed.

Frequency of Occurrence: Low

UC-15.3: Change status of element

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Subject Experts and Validators need to change status of elements in the selected ontology.

Preconditions:

User logs in and opens an ontology or module, which he/she has assigned either Subject Expert or Validator role for the selected ontology (module).

System activates menu or icon bar, based on user’s role.

Success Guarantee:

Selected elements are sent to the next/previous assigned status - this assigned status is various in sub use cases (15.3.1 to 15.3.6). Any change of status can not trigger to start/terminate any editorial workflow.

Fail Guarantee: No status of elements is changed.

Extensions Points:

UC-15.3.1, UC-15.3.2, UC-15.3.3, UC-15.3.4, UC-15.3.5, and UC-15.3.6

Main Success Scenario:

1. User selects an element in menu bar or icon bar in order to change status of it.
2. System changes the status of selected element from current status into next assigned status.

Special Requirements:

Frequency of Occurrence: Medium

Miscellaneous:

Sections 2.52 (Figure 4 and 5) of this deliverable illustrates overview of change status of elements by user's roles.

UC-15.3.1: Send to "To be Approved" status

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Subject Experts need to change status of elements from "Draft" into "To be Approved" status.

Preconditions:

The Subject Expert has editing rights on the selected ontology (module) where the element to be sent for approval is. The user opened the ontology (module) and selected "Draft" view to see the elements waiting for his/her proposal.

Success Guarantee: Selected elements are sent to "To be Approved" status.

Fail Guarantee: No element is sent to "To be Approved" status.

Extensions Points: N/A

Main Success Scenario:

1. User selects an element in "Draft" status
2. User chooses an option "send to To be Approved" in menu bar or icon bar.
3. System changes the status of selected elements from "Draft" status into "To be Approved" status.
4. The element appears now in the "To be approved" view and not anymore in the "Draft" status view.

Extensions: N/A

Special Requirements: N/A

Frequency of Occurrence: Medium

Miscellaneous: N/A

UC-15.3.2: Send to the "Approved" status

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Validators need to change status of elements from "To be Approved" into "Approved" status.

Preconditions:

The Validator has editing rights on the selected ontology (module) where he/she approves to change status of the elements proposed by Subject Experts. The user opened the ontology (module) and selected “to be Approved” view to see the elements waiting for his/her revision and approval.

Success Guarantee: Selected elements are sent to “Approved” status.

Fail Guarantee: No element is sent to “Approved” status.

Extensions Points: N/A

Main Success Scenario:

1. User selects an element in the “To be approved” status
2. User chooses the option “send to Approved” in menu bar or icon bar.
3. System changes the status of selected elements from “To be Approved” status into “Approved” status.
4. The System refreshed the list of “to be approved” and “approved” elements.

Extensions: N/A

Special Requirements: N/A

Miscellaneous: N/A

Frequency of Occurrence: Medium

UC-15.3.3: Reject to “Draft” status.

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Validators need to reject elements proposed to them to review in the “To be approved” status and send them back to the “Draft” status for the Subject Expert to check, update or complete.

Preconditions:

The Validator has editing rights on the selected ontology (module) where he/she approves to change status of the elements proposed by Subject Experts. The user opened the ontology (module) and selected “To be Approved” view to see the elements waiting for his/her revision and approval or rejection.

Success Guarantee: Selected elements are returned back to “Draft” status.

Fail Guarantee: No element is retuned back to “Draft” status.

Extensions Points: N/A

Main Success Scenario:

1. User selects an element in the “To be approved” status
2. User chooses an option “reject to Draft” in menu bar or icon bar.
3. System changes the status of selected elements from “To be Approved” status into “Draft” status.
4. The System refreshed the list of “To be approved” and “Draft” elements.

Extensions: N/A

Special Requirements: N/A

Miscellaneous: N/A

Frequency of Occurrence: Medium

UC-15.3.4: Reject to "To be Approved" status

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Validators need to return an element which is already approved and located in "Approved" status back to the "To be approved" status.

Preconditions:

The Validator has editing rights on the selected ontology (module) where he/she rejects to change status of elements. The user opened the ontology (module) and selected "Approved" view to see the elements waiting for his/her revision and approval or rejection.

Success Guarantee: Selected elements are returned back to "To be Approved" status.

Fail Guarantee: No element is retuned back to "To be Approved" status.

Extensions Points: N/A

Main Success Scenario:

1. User selects an element in the "Approved" status
2. User chooses the option "reject to To be Approved" in menu bar or icon bar.
3. System changes the status of selected elements from "Approved" status into "To be Approved" status.
4. The System refreshed the list of "To be approved" and "Approved" elements.

Extensions: N/A

Special Requirements: N/A

Miscellaneous: N/A

Frequency of Occurrence: Medium

UC-15.3.5: Reject the "To be deleted" proposal for an element

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests: Validators need to reject the proposal of deleting elements.

Preconditions:

The Validator has editing rights on the selected ontology (module) where he/she rejects the "To be deleted" proposal for an element. The user opened the ontology (module) and selected "To be deleted" view to see the elements waiting for approval or rejection.

Success Guarantee:

Selected elements are returned back to "Approved" status. The system automatically assigns status of the elements as "Approved".

Fail Guarantee: No element is retuned back to "Approved" status.

Extensions Points: N/A

Main Success Scenario:

1. User selects an element in the “To be deleted” status
2. User chooses an option “reject the To be deleted l” in menu bar or icon bar.
3. System changes the status of selected elements from “To be Deleted” status into “Approved” status.
4. The list of elements in the “To be Deleted” status is updated.

Extensions: N/A

Special Requirements: N/A

Miscellaneous:

Any current activated window showing a list of “Approved” might be refreshed by adding the rejected element, after this use case is executed.

New arrived Elements in the “To be Deleted” status are highlighted.

Frequency of Occurrence: Low

UC-15.3.6: Accept the “To be deleted” proposal for an element

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Validators need to accept the “To be deleted” proposal for an element in the selected ontology and destroy it.

Preconditions:

The Validator has editing rights on the selected ontology (module) where he/she accepts the “To be deleted” proposal for an element. The user opened the ontology (module) and selected “To be deleted” view to see the elements waiting for approval or rejection

Success Guarantee:

Elements are destroyed and system automatically terminates the current editorial workflow.

Fail Guarantee: No element is destroyed.

Extensions Points: N/A

Main Success Scenario:

1. User selects an element in the “To be deleted” status
2. User chooses “Accept To be deleted” in the menu or icon bar.
3. System confirms to destroy it to user.
4. User confirms it.

Extensions: N/A

Special Requirements: N/A

Frequency of Occurrence: Low

UC-15.4: Publish Ontology

Scope: Ontology validation and update

Level: Sub function

Stakeholders and interests:

Validators need to copy an ontology where all its elements are in the “Approved” status, from the test and validated environment (editorial workflow in the intranet) to the production environment (Internet).

Preconditions:

The Validator has rights on the selected ontology which he/she copies from the test and validation environment to the production environment. The user opened the ontology.

Success Guarantee: Selected ontology is published.

Fail Guarantee: No ontology is published.

Extensions Points: N/A

Main Success Scenario:

1. User chooses an option “publish” in menu bar or icon bar.
2. System copies the new version of ontology into production environment.
3. System saves the selected ontology assigning it the right version.

Extensions: N/A

Special Requirements: N/A

Frequency of Occurrence: Medium

UC-15.5: Collaborative Editing of Ontology Elements

Scope: Ontology validation and update

Level: Sub function

Primary Actor: Domain expert

Stakeholders and interests:

The domain expert wants to navigate the ontology concepts by means of a web site-like interface, in order to evaluate, change, and maintain them by collaborating with ontology engineers.

The ontology engineer wants to collect arguments and feedbacks from domain experts and to have them formally associated with ontology elements.

Preconditions: the domain expert has been logged.

Success Guarantee:

The wiki site has been created and contains all the elements of the source ontology.

Fail Guarantee: No ontology is published.

Extensions Points: (In 2) Synchronization between the wiki site and the NeOn repository.

Main Success Scenario:

1. The ontology engineer or the domain expert deploys the wiki.
2. Domain experts put their comments on the wiki page corresponding to the ontology element he/she is interested in.

3. The ontology engineer export the OWL encoding of the comments and use them in order to fix the ontology
4. The ontology engineer deploys the new updated version of the ontology.

Extensions: N/A

Special Requirements: N/A

Frequency of Occurrence: Medium

Plugin/tools

The GUI for UC-15.1.1 inserting an element is shown in next figure. A Subject Expert inserts a new class (or concept) “Guidebook” as a subclass of “Book”. The black arrow (←) indicates a menu “New Concept” in the right-button-click menu bar, as illustrated in (A). System automatically assigns the “Draft” status in “Guidebook” element. After that, the Subject Expert can visualize the inserted element “Guidebook” by opening a window named “Draft”, which shows a list of element in the “Draft” status by decolorizing elements in other status, as shown in (B).

A. B.

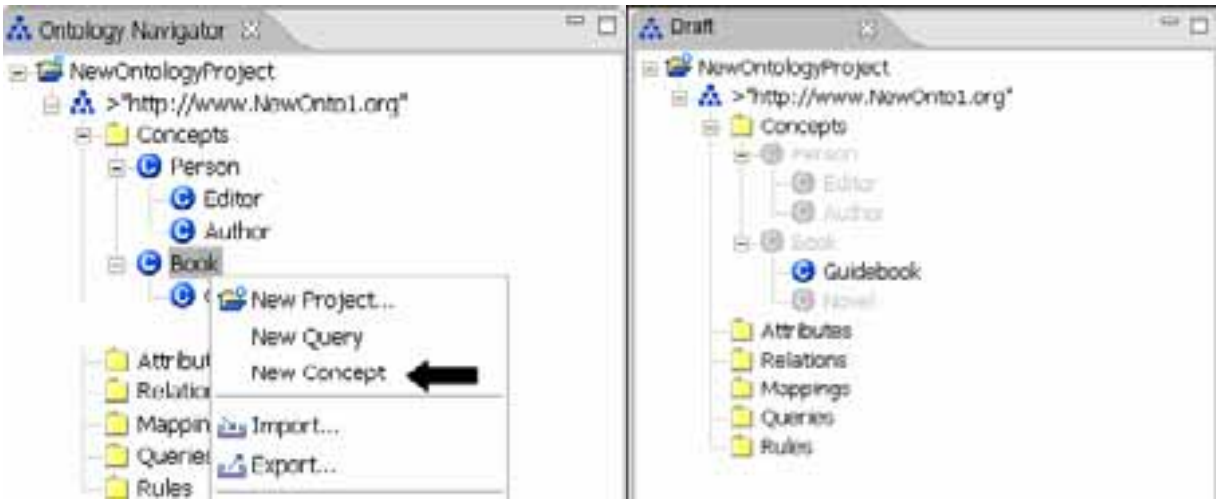


Figure 67. Inserting a class (or concept) in the editorial workflow.

The GUI for UC-15.1.2 Updating an element is illustrated in next figure. A Subject Expert opens a window showing a list of elements in the “Draft” status by decolorizing elements in other status, shown in (A). He/She double-clicks an element to be updated. Then, a new window pops up and visualize information of the element for the update, shown in (B).

A. B.

Figure 68. Updating elements.

Next figure shows the GUI prototype for UC-15.2 Deleting an element, which illustrates how Validator can delete the concept “Guidebook” in the “To be Deleted” status. The black arrow (←) indicates the menu “Delete” to destroy in the right-button-click menu bar.

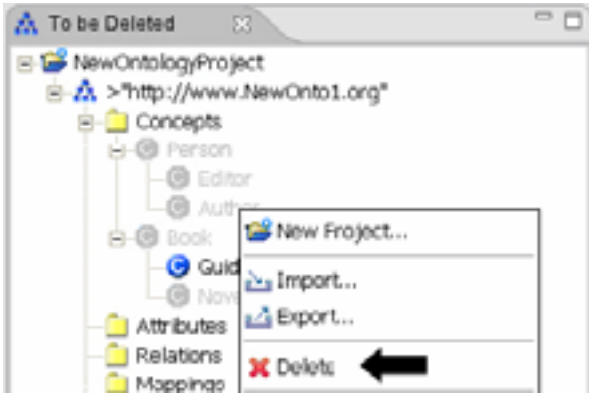


Figure 69. Deleting an element

Next figure shows all of available sub menus under a menu “changing status” in UC-15.3 Change status of element, when Ontology Editors click a right button to see available actions.

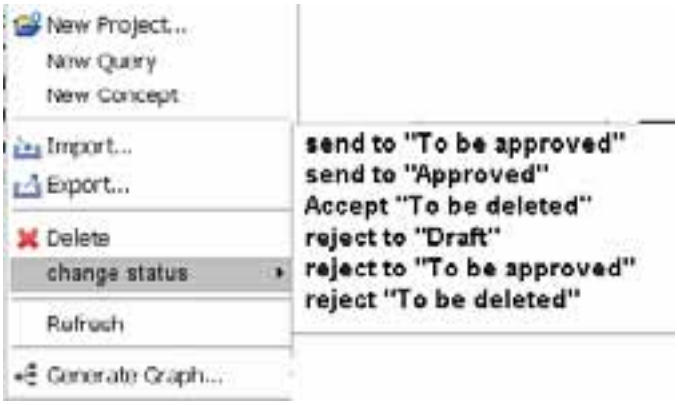


Figure 70. Menus for changing status of element.

Next figure illustrates how a Validator can approve the change of status on elements located in “To be Approved” status, as described in UC-15.3. He/She opens a window which displays a list of elements in the “To be Approved” status. Only elements in the “To be Approved” status are highlighted, comparing that other elements are decolorized. The Validator selects an element to change the status. In the right-button-click menu, the Validator can select either “send to *Approved*” or return the element back to “Draft” status. In both case, system automatically changes status of the element. Note that system only activates shows events which Subject Expert can select, based on user’s role.

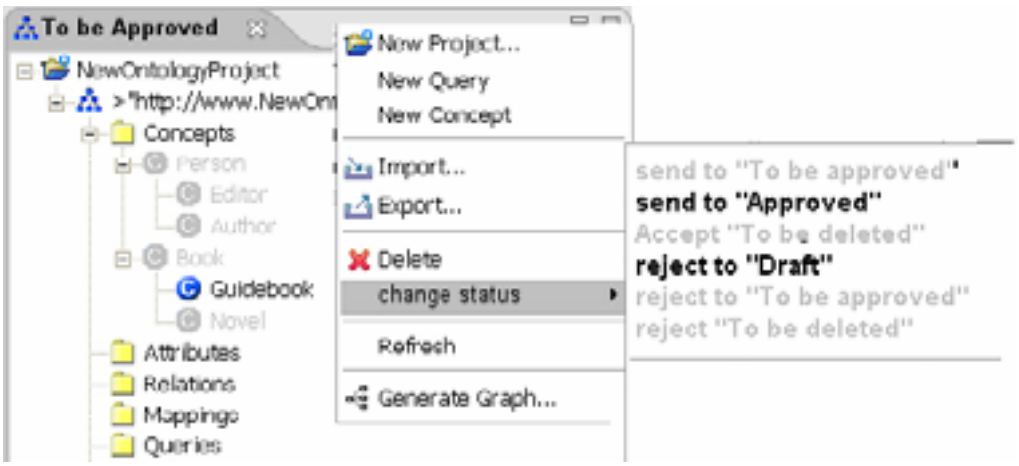


Figure 71. Changing status of elements.

Next figure shows a possible user interface for publishing ontologies, explained in UC-15.4 Publish ontology. A) shows a menu “publish ontology” in the top menu bar. 6. B illustrates that system copies the new version of ontology into production environment automatically. Note that system can visualize the ontology in production environment as well after publishing. These screenshots were captured from Microsoft Window Live Writer[2] and modified for publishing ontology.

[2] http://get.live.com/betas/writer_betas

A. B.

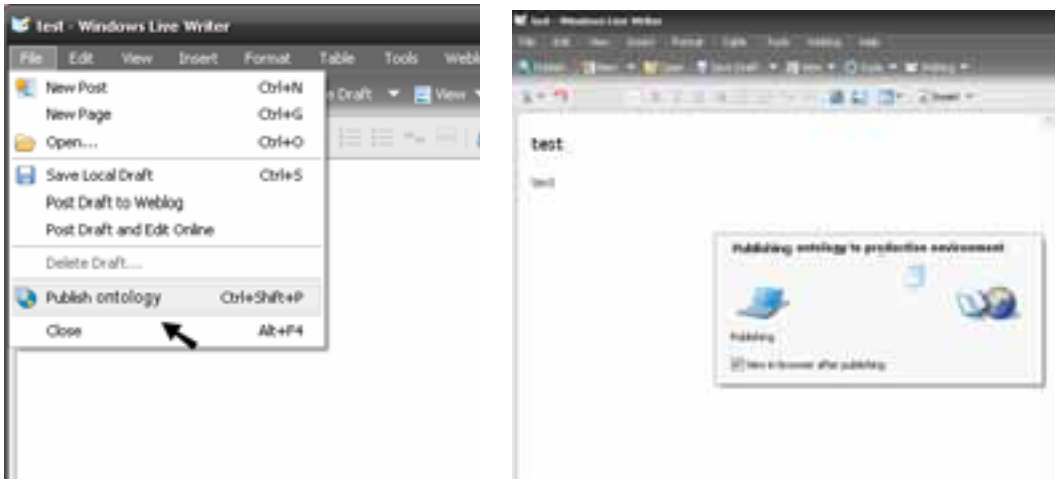


Figure 72. Publishing ontology.

Wikifactory details

WikiFactory is a tool that enables an automatic, ontology-driven deployment of a semantic wiki, where the ontology describes a specific domain of interest.

The resulting deployed semantic wiki can be used as a friendly interface to domain experts, for ontology browsing and editing of both ontology and content.

WikiFactory also provides run time synchronization between ontology and content: changes made over wiki content are reflected upon the underlying ontology, and vice versa.

WikiFactory is a server application, Java-based and can be integrated to any wiki platform by means of a specific plug-in. Currently, a plug-in for Semantic Media Wiki is available.

WikiFactory deployer will be released as a NeOn plug-in. Directly from the NeOn toolkit interface, it will be possible to generate a semantic wiki and deploy it either on a local or remote web server. The semantic wiki will provide domain expert users with a friendlier, interactive interface for collaborative ontology development. As additional functionality, the WikiFactory plug-in will allow users to generate the ontology-driven semantic wiki as an alternative interface perspective within the NeOn toolkit.

Next figure shows a screenshot of the WikiFactory deployer GUI used as a standalone application. GUI of the WikiFactory plug-in is not available yet.

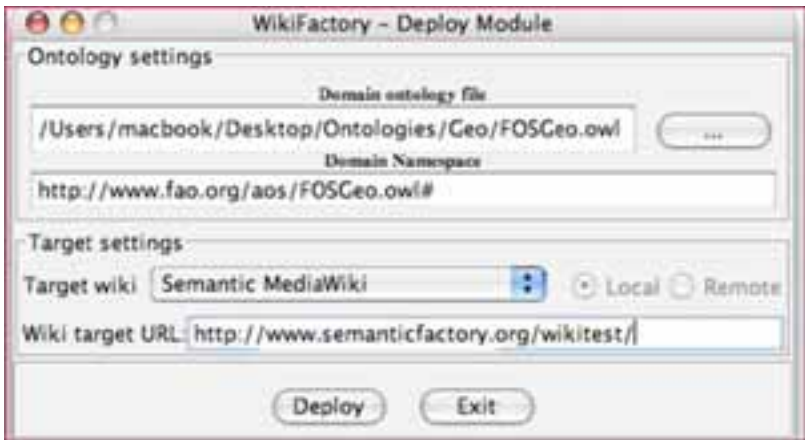


Figure 73. A GUI prototype for the automatic deployment of semantic wiki

WikiFactory features:

- Automatic deployment. Given an OWL ontology, WikiFactory is capable of automatically creating a corresponding semantic wiki.
- Synchronization. Any change applied to the wiki content is reflected upon the OWL ontology and vice versa.
- Help for users. WikiFactory helps users with a semantic assistant that exploits the capabilities of an OWL reasoner.

WikiFactory will be deployed as a Java server application. Furthermore, a NeOn plug-in will be release that will provide the automatic deployment functionality. The WikiFactory plug-in will provide support for argumentation. Specifically, all comments produced during argumentation sessions will be associated with the interested ontology elements by means of OWL object and datatype properties.

WikiFactory relies on the NeOn metamodel (developed in the context of NeOn WP1) and C-ODO (developed in the context of NeOn WP2).