



NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D6.10.2 Realization of core engineering components for the NeOn Toolkit, v2

Deliverable Co-ordinator: Peter Haase

Deliverable Co-ordinating Institution: University of Karlsruhe (UKARL)

Other Authors: Mathieu d’Aquin, Mauricio Espinoza, Qiu Ji, Chan Leduc, Klaas Dellschaft, Raul Palma, Johanna Völker,

This deliverable describes the second set of plugins for the NeOn Toolkit that have been developed by the partners of the NeOn consortium. The plugins support a range of lifecycle activities of the NeOn ontology engineering methodology and significantly enrich the capabilities of the basic NeOn Toolkit.

Document Identifier:	NEON/2009/D6.10.2/v1.0	Date due:	November 30, 2008
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	November 30, 2008
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:

<p>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</p>	<p>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</p>
<p>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</p>	<p>Software AG (SAG) Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com</p>
<p>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</p>	<p>Institut 'Jožef Stefan' (JSI) Jamova 39 SI-1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si</p>
<p>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</p>	<p>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</p>
<p>Universität Koblenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de</p>	<p>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</p>
<p>Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de</p>	<p>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</p>
<p>Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.pariantelobo@atosorigin.com</p>	<p>Laboratorios KIN, S.A. (KIN) C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es</p>

Work package participants

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

UKARL

ONTO

UPM

OU

UKOB

INRIA

Change Log

Version	Date	Amended by	Changes
0.1	01-11-2008	Peter Haase	Creation
0.2	30-11-2008	Peter Haase	Input of partner contributions
0.3	06-12-2008	Peter Haase	Conclusions
1.0	01-08-2009	Peter Haase	Final Changes after QA review.

Executive Summary

The NeOn toolkit is an extensible Ontology Engineering Environment. It serves as the reference implementation of the NeOn architecture. It integrates functionalities common to today's ontology management tools and advances the state-of-the-art by addressing the requirements that must be met in order to support the lifecycle of ontologies in networked, distributed, and collaborative environments.

Basic ontology management and editing functionalities are provided by the core NeOn Toolkit. Plugins extend the core NeOn Toolkit with additional functionalities supporting specific lifecycle activities. The NeOn Toolkit relies on the architectural concepts of the Eclipse platform to enable the development of plugins: The Eclipse IDE (integrated development environment) provides both GUI level components as well as a plugin framework for providing extensions to the base platform.

In deliverable D6.10.1 we have described the first set of plugins that have been implemented by NeOn partners for the NeOn Toolkit.

This deliverable at hand (D6.10.2) provides an update of the available partner plugins. We describe plugins that have been developed either completely new or have been considerably extended.

The developments have been driven largely based on feedback from case study partners who used the first set of plugins within the case study prototypes. With the new set of plugins we now support an even wider range of ontology lifecycle activities according to the NeOn methodology.

Additional novel contributions of this deliverable include the availability of Eclipse help functionality for all plugins and an improved quality assurance process.

Table of Contents

NeOn Consortium	2
Work package participants	3
Change Log	3
Executive Summary	4
Table of Contents	5
List of figures	6
1. Introduction	9
1.1. The NeOn Toolkit: Architecture and Plugins	9
1.3. Relationship with other workpackages	11
2. Plugin Development Process	11
2.1. Source Code Management	11
2.2. Licensing	11
2.3. Plugin Wiki.....	12
2.4. Plugin Documentation	13
2.5. Quality Assurance and Bug Management.....	13
2.6. Download Site: Plugins as Eclipse Features.....	14
3. Plugin Descriptions.....	16
3.1. Overview of Plugins.....	16
3.2. Alignment Server Plugin.....	17
3.2.1. Functional Description	17
3.2.2. User Documentation.....	18
3.2.3. Integration into the NeOn Toolkit.....	22
3.2.4. Intended Usage in the Case Studies.....	23
3.3. Change Capturing	23
3.5. Cicero	30
3.6. Collaborative Workflow Support.....	35
3.7. DIG Interface	38
3.8. LabelTranslator	42
3.9. LeDA	48
3.10. Modules Plugin.....	51
3.11. ODEMapster.....	55
3.12. OntoModel.....	72
3.13. OWLDoc.....	76
3.14. Oyster Registry.....	78
3.15. RaDON Plugin.....	93
3.16. SPARQL Plugin.....	101
3.17. Text2Onto.....	103
3.18. Watson	105
4. Conclusions and Future Work	108
5. References	109

List of figures

Figure 1: NeOn Architecture	9
Figure 2: Plugin Concept of Eclipse	10
Figure 3: NeOn Plugin Wiki	12
Figure 4: NeOn Toolkit Help	13
Figure 5: NeOn Toolkit Bugzilla	14
Figure 6: NeOn Toolkit Update Site	15
Figure 7: Functions of the Alignment Server plug-in	18
Figure 8: Matching two ontologies	20
Figure 9: Browse OWL alignment in Ontology Navigator	21
Figure 10: Fetch alignments available from server	22
Figure 11: Starting Change Capturing	24
Figure 12: Visualizing Changes	25
Figure 13: Change Synchronization	26
Figure 14: Architecture of Change Capturing	27
Figure 15: Alignment Server: Fetching Alignments	29
Figure 16: Cicero Configuration	31
Figure 17: Cicero Context Menu	32
Figure 18: Annotating with an Issue in Cicero	33
Figure 19: Creating an Issue	33
Figure 20: Showing an Issue	34
Figure 21: Collaborative Development Preferences	36
Figure 22: Draft view	37
Figure 23: Setting the Reasoner Preferences	39
Figure 24: Invoking the coherency view	40
Figure 25: Getting back the results	40
Figure 26: A classification example	41
Figure 27: Linguistic Information Page	44
Figure 28: Label Translation Context Menu	44
Figure 29: Label Translation	45
Figure 30: Updating the Linguistic Information Repository	46
Figure 31: Lexicalizations associated with a lexical entry	46
Figure 32: Architecture of LabelTranslator	47
Figure 33: LeDA User Interface	49
Figure 34: LeDA Preferences	50
Figure 35: The result of union operator	52

Figure 36: The alignment between two modules	53
Figure 37: Module Specification	54
Figure 38: Open R2O Mapping Perspective	56
Figure 39: R2O Mapping perspective sections	57
Figure 40: Ontology visualization	58
Figure 41: New R2O Mapping item	58
Figure 42: Create a new R2O mapping window	59
Figure 43: Create a database window	59
Figure 44: R2O Mapping perspective	60
Figure 45: Constant mapping	61
Figure 46: R2O Code	61
Figure 47: Concat mapping	62
Figure 48: getDelimited mapping	63
Figure 49: Show and hide mappings	64
Figure 50: Filter database items	65
Figure 51: Filter ontology items	66
Figure 52: Expand and collapse the database tree.	67
Figure 53: Remove a mapping	68
Figure 54: Query mapping menu item	69
Figure 55: Query mapping window	70
Figure 56: ODEMapster component within NeOn Toolkit.	70
Figure 57: FAO Databases	71
Figure 58: Adding a view	72
Figure 59: Ontology diagram	73
Figure 60: Module Parameters	74
Figure 61: OntoModel Architecture	75
Figure 62: OWLDoc Export	77
Figure 63: Starting Oyster	80
Figure 64: Stopping Oyster	81
Figure 65: Oyster Preferences	82
Figure 66: Server selection combo	82
Figure 67: Server list shown by the server selection combo	83
Figure 68: Server list dialog	84
Figure 69: The search view	85
Figure 70: The configuration section	85
Figure 71: Properties selection dialog	86
Figure 72: The search results view	87

Figure 73: The import ontology entry in the context menu	88
Figure 74: The import ontology dialog	88
Figure 75: Toolbar detail with the submit metadata button on the top right corner	89
Figure 76: The submit dialog	89
Figure 77: The update entry in the context menu	90
Figure 78: Oyster architecture	91
Figure 79: Invoking Diagnosis and Repair	94
Figure 80: Handling incoherence	95
Figure 81: To repair an ontology automatically	96
Figure 82: To repair an ontology manually	97
Figure 83: The input for an ontology network	98
Figure 84: Compute the justifications for an ontology network	99
Figure 85: Repair an ontology network manually	100
Figure 86: SPARQL Query View	102
Figure 87: Text2Onto User Interface	103
Figure 88: Text2Onto Preferences	104
Figure 89: Watson Search	106

1. Introduction

1.1. The NeOn Toolkit: Architecture and Plugins

The NeOn toolkit is an extensible Ontology Engineering Environment. It serves as the reference implementation of the NeOn architecture. In this section we present an overview on the NeOn architecture, which is targeted to become the reference architecture for ontology management in large-scale semantic applications. The NeOn reference architecture (as introduced in [NeOnD621]) integrates functionalities common to today’s ontology management tools and advances the state-of-the-art by addressing the discussed requirements that must be met in order to support the lifecycle of ontologies in networked, distributed, and collaborative environments.

The general architecture of NeOn is structured into three layers (see Figure 3-8). The layering is done according to increasing abstraction together with the data- and process flow between the components. This results in the following layers:

- **Infrastructure services:** this layer contains the basic services required by most ontology applications.
- **Engineering components:** this middle layer contains the main ontology engineering functionality realized on the infrastructure services. They are differentiated between tightly coupled components and loosely coupled services. Additionally interfaces for core engineering components are defined, but it is also possible to realize engineering components with new specific ontology functionality.
- **GUI components:** user front-ends are possible for the engineering components but also directly for infrastructure services. There are also a predefined set of core GUI components.

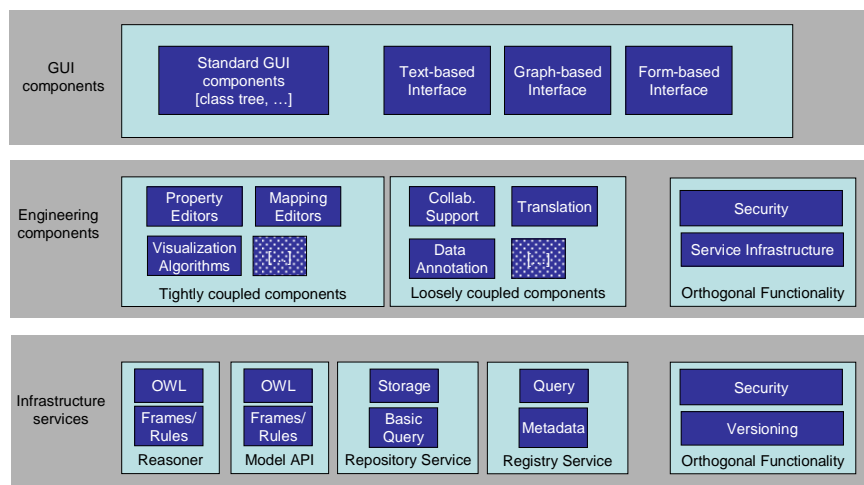


Figure 1: NeOn Architecture

1.2. Eclipse as integration platform

The NeOn architecture relies on the architectural concepts of the Eclipse platform¹. The Eclipse IDE (integrated development environment) provides both GUI level components as well a plugin framework for providing extensions to the base platform.

The Eclipse platform itself is highly modular. Very basic aspects are covered by the platform itself, such as the management of modularized applications (plugins), a workbench model and the base for graphical components. The real power in the Eclipse platform lies however in the very flexible plugin concept.

Plugins are not limited to certain aspects of the IDE but cover many different kinds of functionalities. For example the very popular Java-development support is not provided by the Eclipse platform but by a set of plugins. Even functionalities users would consider to be basic (such as the abstraction and management of resources like files or a help system) are realized through plugins. This stresses the modular character of Eclipse, which follows the philosophy that “everything is a plugin”.

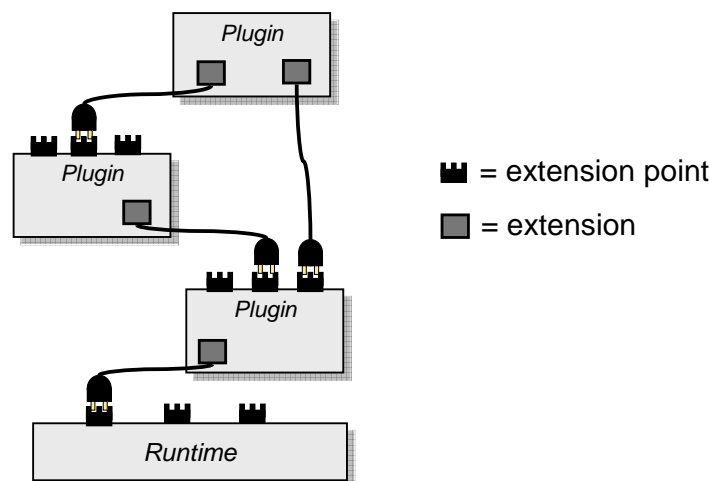


Figure 2: Plugin Concept of Eclipse

A plugin itself can be extended by other plugins in an organized manner. As shown in Figure 2: Plugin Concept of Eclipse, plugins define extension points that specify the functionality which can be implemented to extend the plugin in a certain way. An extending plugin implements a predefined interface and registers itself via a simple XML file. In the XML file the kind of extension as well as additional properties (such as menu entries) are declared.

¹ <http://www.eclipse.org/>

1.3. Relationship with other workpackages

The role of the WP6 task T6.10 “Realization of Core Engineering Components” is to coordinate the development of plugins, integrating novel methods and functionality developed in the individual technical workpackages (WP1-WP5). For example, the Oyster plugin makes available the registry functionality developed in WP1, the Cicero plugin interfaces with the argumentation support technology developed in WP2, etc.

At the same time, in WP5 we develop the NeOn methodology for engineering networked ontologies, for which the NeOn toolkit with its plugins provides the technical tool support. The methodology – as described in its initial version in [NeOnD531] – is organized along a set of ontology lifecycle activities. In Section 3.1 we provide an overview, which activities are supported by which plugin.

As part of WP10, we provide the training activities to disseminate a working knowledge of the toolkit according to the NeOn methodology.

Finally, in the case study workpackages, we deploy special configurations of the NeOn Toolkit within the case study prototypes. These configurations typically consist of the NeOn Toolkit along with a subset of the available plugins that are needed to support the ontology lifecycle activities in the case study. Additionally, the configuration may include plugins that are developed specifically for the case study prototypes. In the description of the plugins in Section 3, we explain how the individual plugins are used in the case studies.

2. Plugin Development Process

Generally, the plugin development process is a very open and decentralized one: Everybody is free to develop and publish his or her plugin, and to make it available under the conditions he or she prefers.

Yet to ensure a coherent development and coordination among developers *within* the NeOn project, we have set up a basic process for plugin development that includes guidelines addressing aspects such as source code management, licensing and quality assurance.

2.1. Source Code Management

The default assumption for plugins developed within the NeOn consortium is that the plugins are available in open source. Plugin developers are free to use the source code management system (SCM) of their choice for their plugin. However, as the default SCM, we provide the Ontoware system (<http://www.ontoware.org>). Ontoware is a SCM similar in functionality to sourceforge, yet it focuses on hosting project related to ontologies and semantic technologies. Ontoware does not impose any restrictions on licensing schemes. Currently, roughly 90 projects are hosted at Ontoware, around 15 of them being plugins for the NeOn toolkit.

2.2. Licensing

The NeOn Toolkit in its open source version is distributed under the Eclipse Public License (EPL). In order to avoid conflicts in bundling the plugin with the NeOn Toolkit, we therefore recommend also the EPL as the default license to plugin developers.

The EPL is an open source software license used by the Eclipse Foundation for its software. We deem the EPL as appropriate, as every plugin to the NeOn toolkit by its nature also is an Eclipse plugin and typically reuses various other Eclipse plugins.

The EPL is designed to be a business friendly free software license, and features weaker copy left provisions than contemporary licenses such as the GNU General Public License (GPL). The receiver of EPL-licensed programs can use, modify, copy and distribute the work and modified versions, in some cases being obligated to release their own changes.

The EPL is approved by the Open Source Initiative (OSI) and the Free Software Foundation (FSF).

2.3. Plugin Wiki

As a central entry point for information about available plugins, we maintain a plugin wiki. The purpose of the plugin wiki is to enable both the developers and users of plugins to create and find information about plugins. The plugin descriptions include metadata such as developer and developer's affiliation, availability, license, etc., along with a functional description and user documentation. The plugin wiki is integrated into the NeOn Toolkit portal (c.f. Figure 3: NeOn Plugin Wiki)

The screenshot shows the NeOn Plugin Wiki interface. At the top, there's a navigation bar with links like 'article', 'discussion', 'edit', etc. The main content area is divided into several sections:

- Welcome:** A yellow box containing a welcome message and a brief description of the wiki's purpose.
- Featured Plugin:** A section with an '[edit]' link.
- Topics:** A section with an '[edit]' link and a grid of topic links: Database, Editor, Export, Import, Inference, Project Management, Reasoning, Search & Navigation, Semantic Web, Software Engineering, Terminologies, Visualization, and Validation. Below this is an 'Add New Topic' button.
- News:** A section with an '[edit]' link and a table of recent updates.
- Plugin registration:** A green box with an '[edit]' link and a form to add a plugin to the directory.
- Organisation registration:** A green box with an '[edit]' link and a form to add an organization to the directory.
- Search:** A purple box with an '[edit]' link and a search form for plugins, users, and organizations.

Plugin	Updated
Cicero	22 January 2008
WikiFactoryDeployer	13 January 2008
SAFE-Plugin	19 December 2007
RaDON	9 December 2007
Text2Onto	11 November 2007
OntoModel	11 November 2007
Plugin for Ontology Schema Modeling	10 May 2007
Plugin for Query Answering	10 May 2007
Plugin for Rule Modeling	10 May 2007
Plugin for Rule Debugging	10 May 2007

Figure 3: NeOn Plugin Wiki

In addition, we also maintain an internal wiki (restricted to the NeOn consortium) for plugins that are not officially published yet. This internal wiki contains additional information about the current implementation status, integration issues etc. and is used to track and coordinate the progress of the plugin developments.

Once a plugin is completed, the relevant information is simply copied from the internal wiki to the public one.

2.4. Plugin Documentation

In addition to the plugin wiki, all plugins provide user documentation integrated into the Eclipse help infrastructure. This allows an easy documentation for all plugins within the NeOn Toolkit in a standardized way.

Figure 4: NeOn Toolkit Help shows the appearance of the NeOn Toolkit help in a sample configuration with a subset of the plugins installed.

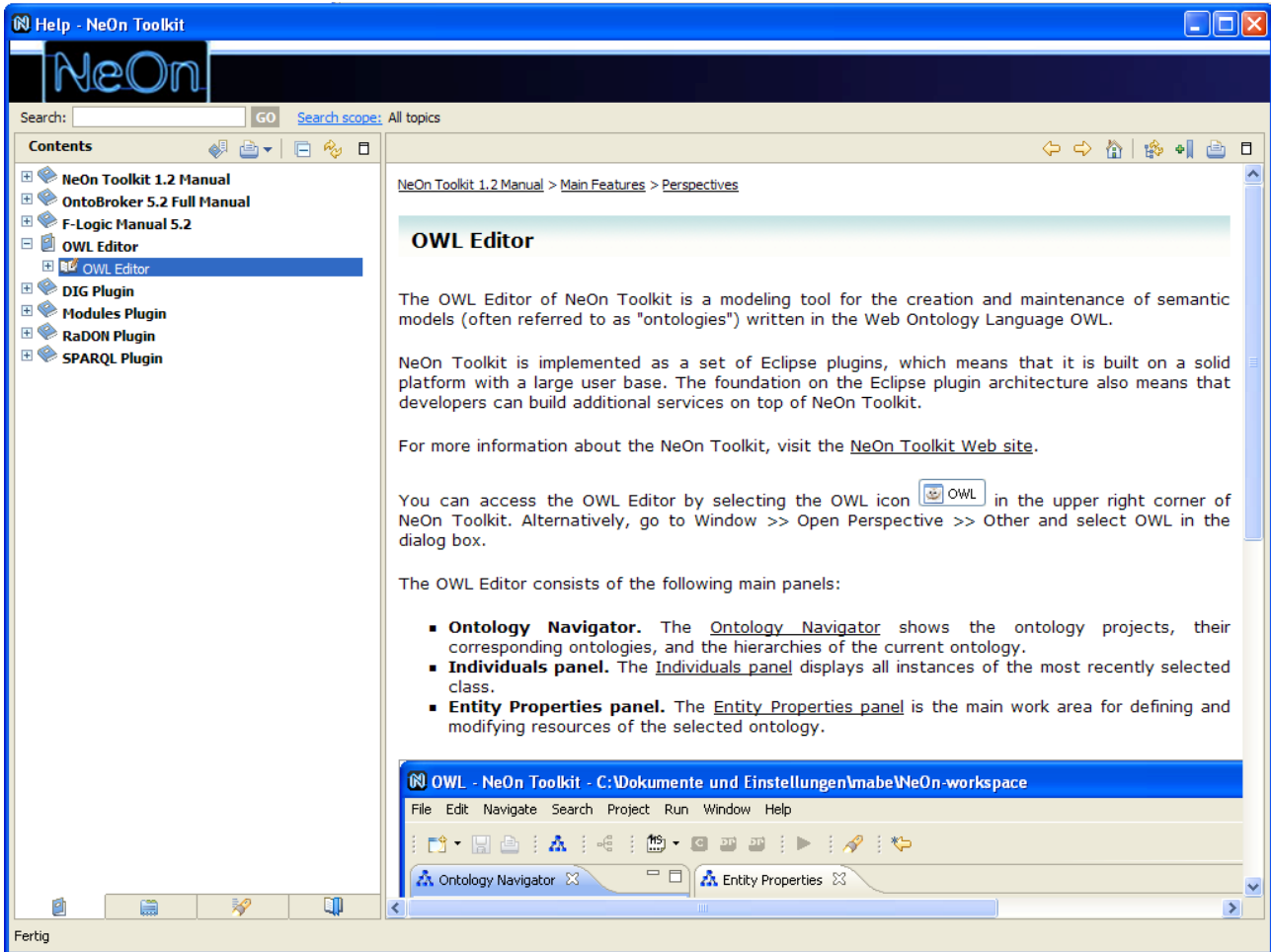


Figure 4: NeOn Toolkit Help

2.5. Quality Assurance and Bug Management

In order to ensure a professional appearance, we have established two basic measures that every plugin needs to meet before it is officially published. The first measure is an additional reviewing process for the plugins, the second measure being automated test cases.

Before a plugin is officially published and made available to the NeOn community, every plugin is formally reviewed by at least two persons from within the NeOn consortium. The review is done using a review form, which covers the following aspects:

- overall quality
- deployment procedure
- functionality and ease of use according to user documentation

- completeness and correctness of the description in the plugin wiki
- Eclipse help documentation

Further, every plugin we additionally require automated test cases to be defined. These test cases are realized in a standard way using JUnit.

In order for the users of the toolkit to have a single point to report bugs and get support in the case of problems, we maintain a bug management system within the NeOn Toolkit. The bug management system is based on Bugzilla portal (c.f. Figure 5: NeOn Toolkit Bugzilla).

The screenshot shows the Bugzilla 'Enter Bug' page for the NeOn Toolkit. The page has a dark header with the NeOn Toolkit logo and navigation links (Home, Community, Download, Resources, About Us). Below the header is a search bar and a navigation breadcrumb (Home > Community > Bugs). The main content area is titled 'Bugzilla - Enter Bug: NeOn Toolkit' and contains a form for reporting a bug. The form includes the following fields and values:

- Reporter:** uuasp@stud.uni-karlsruhe.de
- Version:** 1.0
- Severity:** enhancement
- Priority:** P5
- Initial State:** NEW
- Assign To:** Chan.Leduc@inrialpes.fr
- Cc:** (empty)
- Default Cc:** pha@aifb.uni-karlsruhe.de
- Estimated Hours:** 0.0
- Deadline:** (empty) (YYYY-MM-DD)
- URL:** http://
- Summary:** (empty)
- Description:** (empty)

Additional form elements include a 'Product' dropdown set to 'NeOn Toolkit', a 'Component' dropdown set to 'Alignment Server Plugin', a 'Platform' dropdown set to 'PC', and an 'OS' dropdown set to 'Windows'. There are also links for 'bug writing guidelines', 'most frequently reported bugs', and a 'search' button.

Figure 5: NeOn Toolkit Bugzilla

Depending on the experience and feedback to be collected, we may opt for a stricter quality assurance process (see Future Work).

Finally, as part of [NeOnD613], we will provide an evaluation of the plugins against the requirements defined for the NeOn Toolkit.

2.6. Download Site: Plugins as Eclipse Features

To make the deployment of new plugins as easy and smooth as possible for the end user, we make use of the Eclipse Update mechanism², a mechanism that allows deploying and updating new features.

Features are a concept of Eclipse to represent a unit of useful set of functionality. The role of features is to allow providers to make collections of plug-ins that logically go together. As such,

² <http://www.eclipse.org/articles/Article-Update/keeping-up-to-date.html>

when we talk about installing a plugin for the NeOn Toolkit, we actually mean a feature consisting of a set of plugins.

Features are made in such a way as to provide for easy transport over the network, have necessary legal and security mechanisms, and are modular to allow hierarchical product building. Features are designed to help in installing new functionality into Eclipse products and to update the plug-ins you already have to the newer versions.

For the NeOn Toolkit, we created a dedicated NeOn Update site at <http://www.neon-toolkit.org/plugins>. After a quality assurance procedure, newly available plugins (features, more precisely) are uploaded to the update site and thus immediately become accessible to all users of the toolkit.

Figure 6: NeOn Toolkit Update Site shows a screenshot of the NeOn toolkit update mechanism. The available updates are grouped by the functionality they provide. The user merely has to select the features he wants to install, while the update mechanism takes care of the deployment, dependency management, etc.

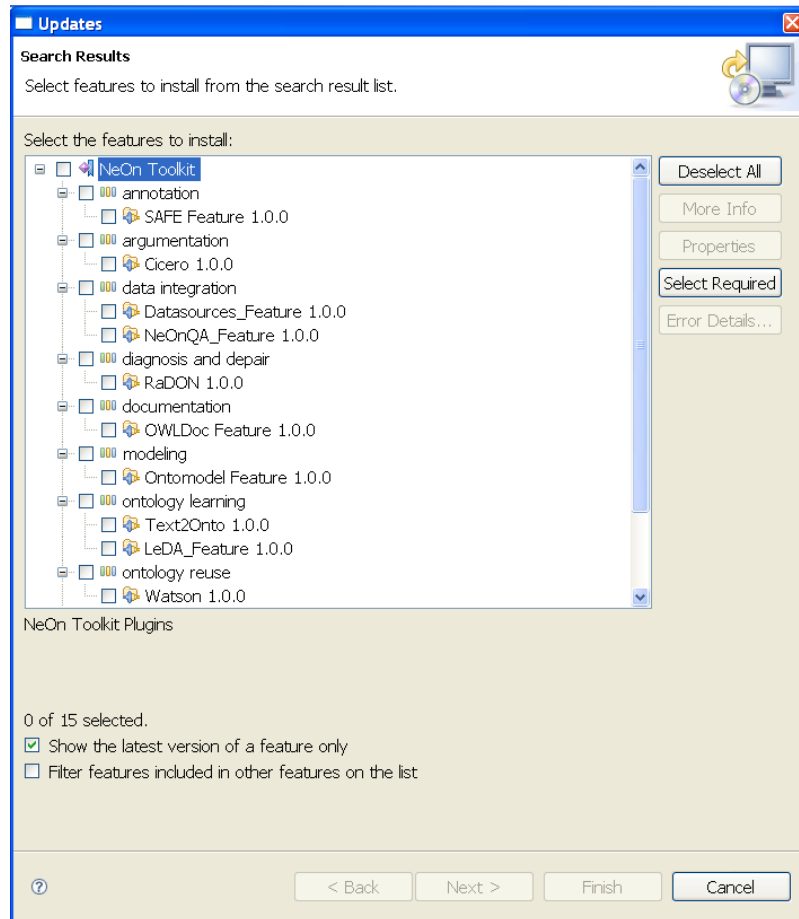


Figure 6: NeOn Toolkit Update Site

3. Plugin Descriptions

In this chapter, we provide descriptions of the individual plugins. We only include the plugins that either have been developed new or have been significantly extended since the publication of the first set of plugins in deliverable D6.10.2. After a general overview, we provide for each plugin a detailed functional description, user documentation, details about the integration into the NeOn Toolkit as well as information about the intended usage in the case studies.

3.1. Overview of Plugins

In the following table we list the developed plugins along with their classification according to the NeOn lifecycle activities (c.f. [NeOnD531]) and developer. A complete analysis of the coverage of the lifecycle activities is provided as part of [NeOnD532].

Table 1. Mapping from Processes and Activities to Plug-ins that Support them

Plug-in	Process or Activity
<input type="checkbox"/> AlignmentServer	<input type="checkbox"/> Ontology Aligning
<input type="checkbox"/> Change Capturing	<input type="checkbox"/> Ontology Versioning <input type="checkbox"/> Ontology Evolution <input type="checkbox"/> Ontology Comparison
<input type="checkbox"/> Cicero	<input type="checkbox"/> Ontology Argumentation
<input type="checkbox"/> Collaborative Workflow Support	<input type="checkbox"/> Ontology Versioning <input type="checkbox"/> Ontology Evolution <input type="checkbox"/> Ontology Conceptualization <input type="checkbox"/> Ontology Formalization
<input type="checkbox"/> DIG Interface	<input type="checkbox"/> Ontology Reasoning
<input type="checkbox"/> LabelTranslator	<input type="checkbox"/> Ontology Localization
<input type="checkbox"/> Modules	<input type="checkbox"/> Ontology Modularization <input type="checkbox"/> Ontology Module Extraction <input type="checkbox"/> Ontology Partitioning <input type="checkbox"/> Ontology Matching <input type="checkbox"/> Ontology Merging
<input type="checkbox"/> LeDA	<input type="checkbox"/> Ontology Learning
<input type="checkbox"/> ODEMapster	<input type="checkbox"/> Non-ontological Resource Reuse <input type="checkbox"/> Ontology Population

<input type="checkbox"/> OntoModel	<input type="checkbox"/> Ontology Conceptualization <input type="checkbox"/> Ontology Formalization <input type="checkbox"/> Ontology Evolution
<input type="checkbox"/> OWLDoc	<input type="checkbox"/> Ontology Documentation
<input type="checkbox"/> Oyster	<input type="checkbox"/> Ontology Reuse
<input type="checkbox"/> RaDON	<input type="checkbox"/> Ontology Diagnosis <input type="checkbox"/> Ontology Repair
<input type="checkbox"/> Text2Onto	<input type="checkbox"/> Ontology Learning
<input type="checkbox"/> Watson	<input type="checkbox"/> Ontology Enrichment <input type="checkbox"/> Ontology Reuse

3.2. Alignment Server Plugin

3.2.1. Functional Description

The Alignment Server Plugin allows one to automatically compute, manipulate and manage ontology alignments. More precisely, the Alignment Server Plugin offers the following functionalities:

- **Find alignments between ontologies**
- **Match ontologies**
- **Trim alignments by applying thresholds to existing alignments**
- **Retrieve and render alignment in a particular format**
- **Store an alignment permanently on the server**

These functionalities support the alignment life-cycle which can be described as follows. *Alignments* are first created through a matching process (which may be manual). Then they can go through an iterative loop of evaluation and enhancement. Again, evaluation can be performed either manually or automatically, it consists of assessing properties of the obtained alignment. Enhancement can be obtained either through manual change of the alignment or application of refinement procedures, e.g., selecting some correspondences by applying thresholds. When an alignment is deemed worth publishing, then it can be stored and communicated to other parties interested in such an alignment. Finally, the alignment is transformed into another form or interpreted for performing actions like mediation or merging.

3.2.2. User Documentation

Use the plugin with the Alignment Server.

To activate the Alignment Server Plugin from the NeOnToolkit, click on the "Align" menu or the button on Toolbar, a view "Alignment" for the plug-in will be opened. Figure 7: Functions of the Alignment Server plug-in shows the available functions of the plug-in.

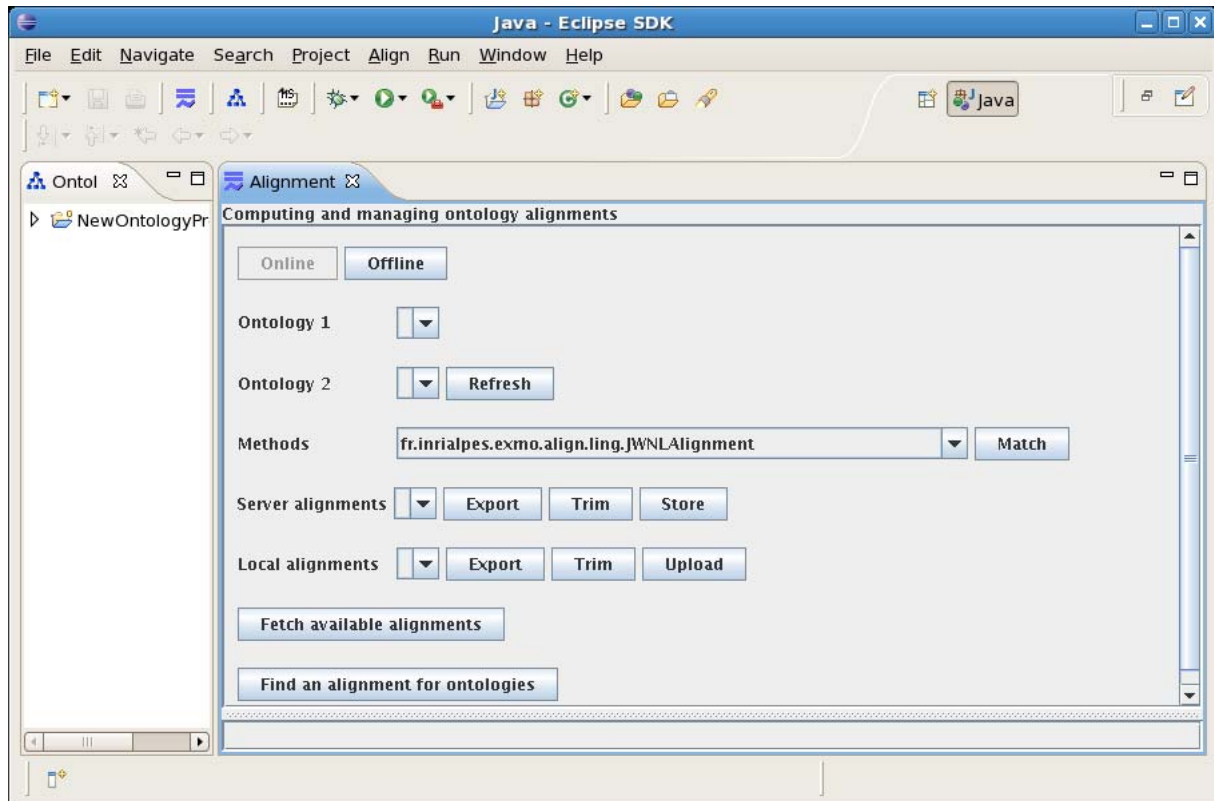


Figure 7: Functions of the Alignment Server plug-in

The Alignment Server Plugin can work in two modes: offline and online. Roughly speaking, the offline mode allows users to reach main functionalities of the Alignment Server without connection to server while the online mode offers additionally mechanisms to store and reuse alignments. In the offline mode, which is available by default or activated by clicking on button "Offline", the Alignment Server Plugin can access to NeOn Toolkit ontologies (i.e. opened ontologies in Ontology Navigator) and match any pair of them. Resulting alignments can be stored as local system files and exported to Ontology Navigator as OWL ontologies. In the online mode, which is activated by clicking on button "Online", the Alignment Server Plugin provides all functions from the Alignment Server. Resulting alignments are stored on the server and can be exported to Ontology Navigator as OWL ontologies. This allows NeOn Toolkit users, with help of an Ontology Editor, to use, share or edit alignments.

Stepwise example

Show the Ontology Navigator view

From the menu bar of the NeOnToolkit, create an OWL project, for example "OntologyProject", in this view.

Open working OWL ontologies

From the menu bar of the NeOnToolkit, open OWL ontologies in the project "OntologyProject" created above.

Activate the Alignment Server Plugin

From the menu bar or the button, activate the Alignment Server Plugin and a view for the plug-in will be opened.

Activate the on-line mode

From the view for the plug-in, activate the on-line mode by clicking on the button "Online".

Connecting

To connect to the INRIA's Alignment Server from the NeOn Alignment Plugin, you have to type "aserv.inrialpes.fr" for hostname and "80" for port.

If the connection is successful, a list of available alignment methods is visible at "methods".

Matching two ontologies

First we must fetch the opened ontologies from Ontology Navigator by clicking on the button "Refresh". All the working ontologies will be added to two lists "Ontology 1" and "Ontology 2". Choose two ontologies to match from the two lists. Next, choose an alignment method from the list "Methods". Click on "Match" to match these two ontologies with the method chosen. The resulting alignment will be added to the list "Server alignments" as an URI if the online mode is active.

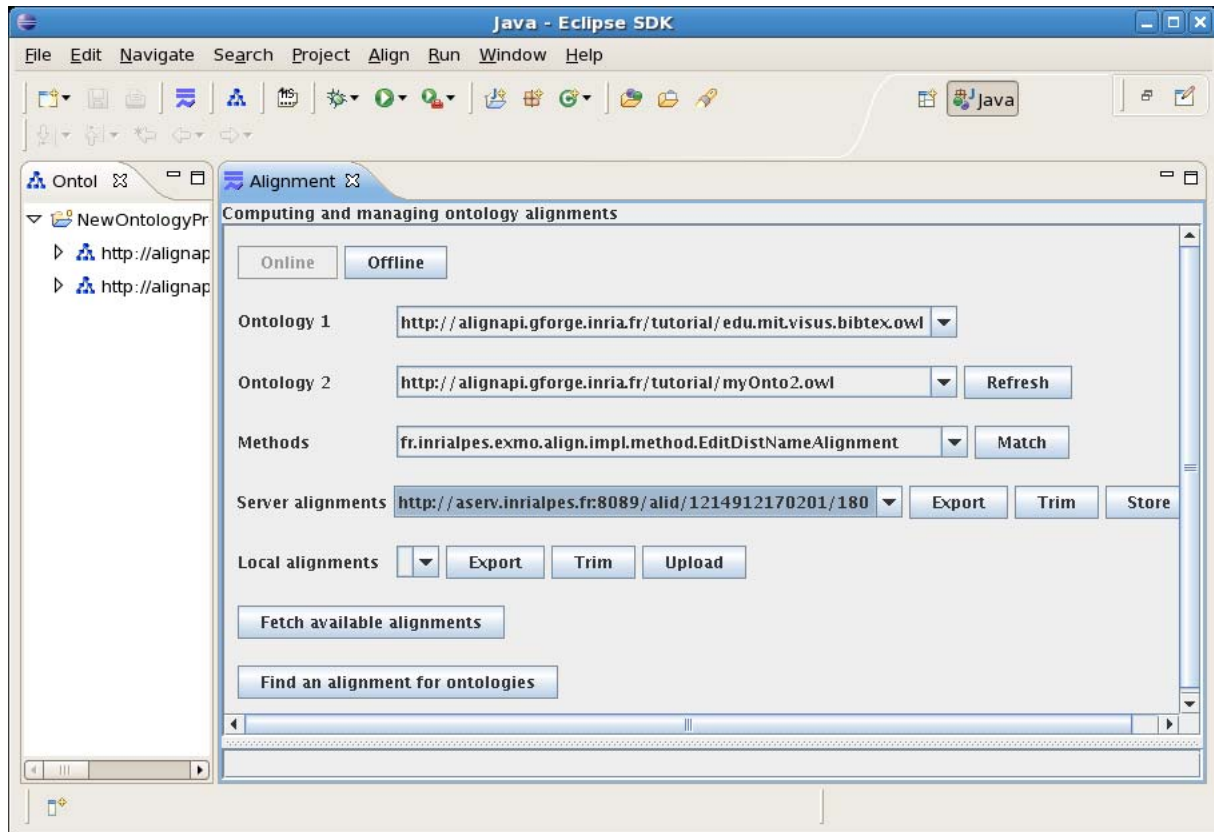


Figure 8: Matching two ontologies

Exporting an alignment

To browse and export the alignment computed, choose it from the list "Server Alignments" and click on "Export". The alignment will be visualized in a native format and exported as an OWL ontology to the alignment project which was created in Ontology Navigator. Users can navigate the OWL alignment imported in Ontology Navigator.

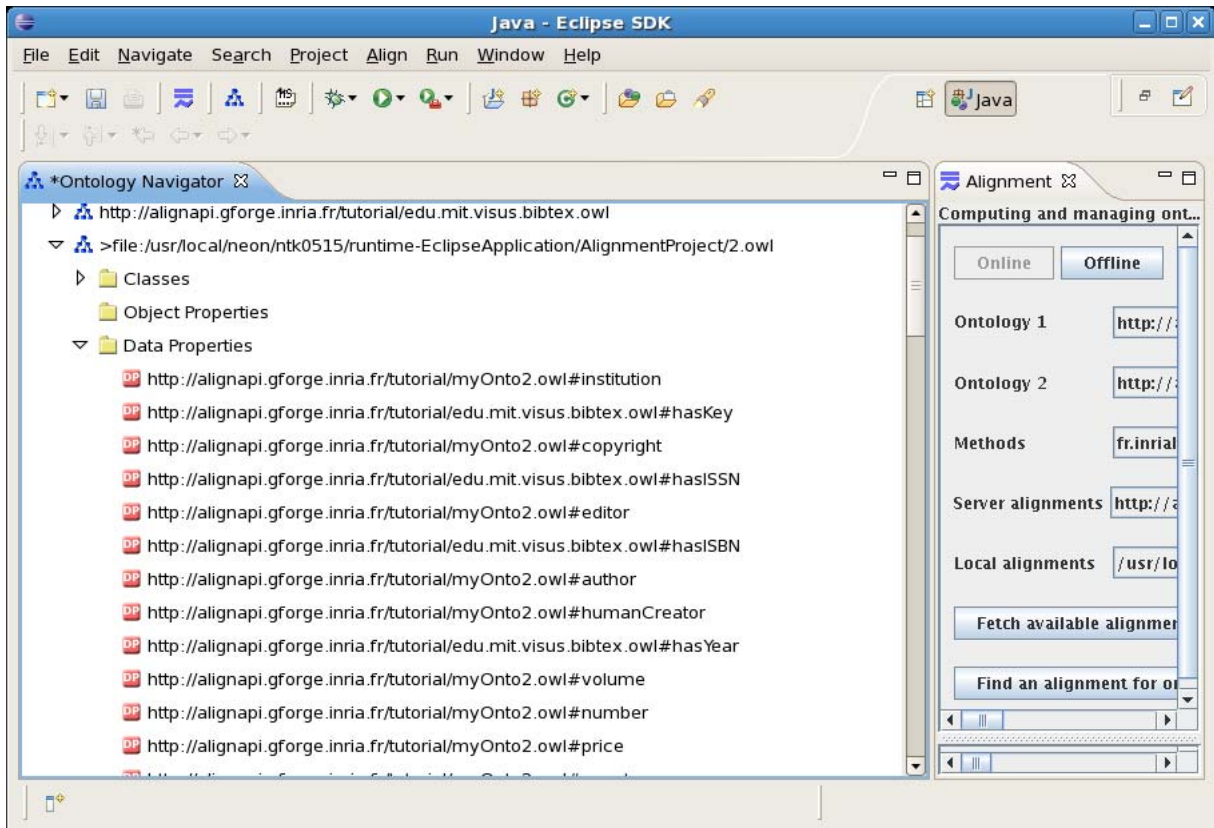


Figure 9: Browse OWL alignment in Ontology Navigator

Fetch available alignments

This function allows users to obtain a list of all alignments available on the Alignment Server. See Figure 10: Fetch alignments available from server.

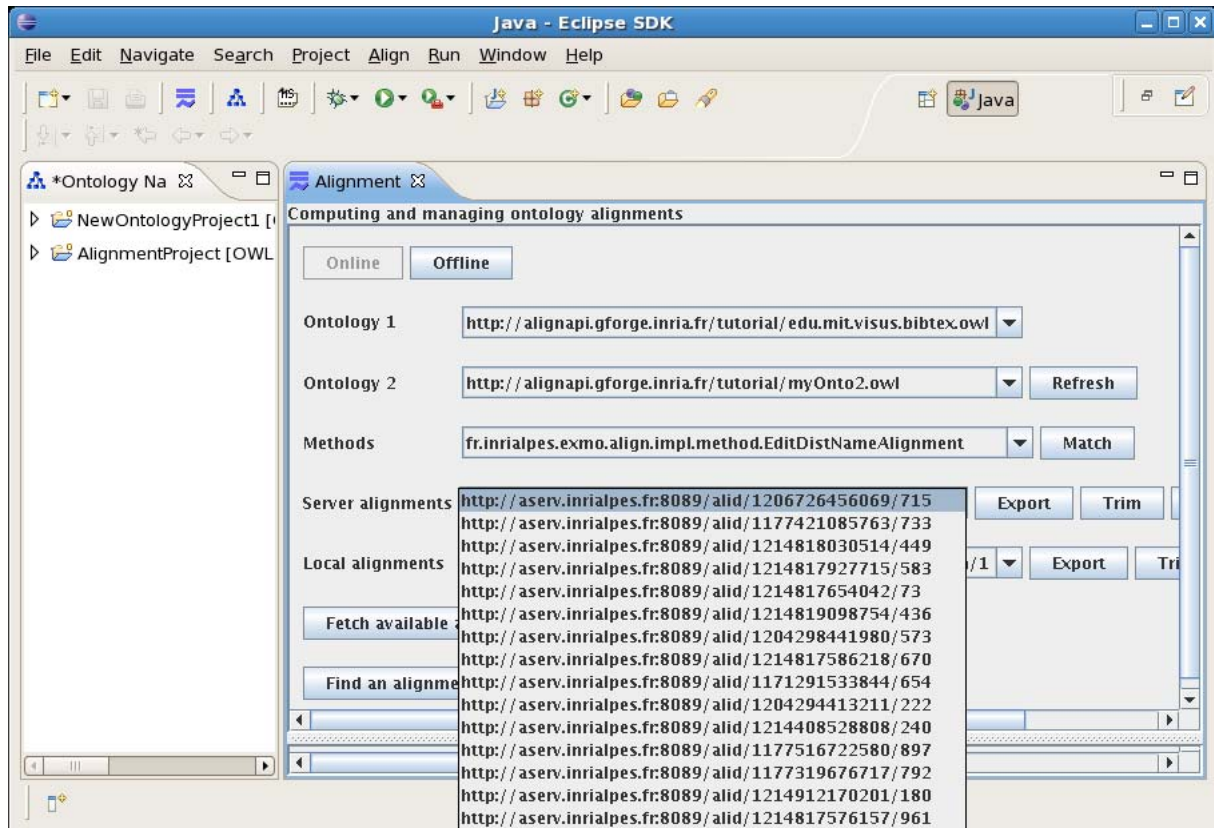


Figure 10: Fetch alignments available from server

Trim an alignment with a threshold

This function allows users to select from an alignment the correspondences whose confidence degree is greater or equal to a threshold. The selected correspondences will be store in a new alignment.

Upload and store an alignment

These functions allow users to upload an alignment, which can be manually created and edited, from local machine to Alignment Server. Alignments can be permanently stored on the server. This provides a possibility to share and reuse an alignment.

3.2.3. Integration into the NeOn Toolkit

In order to be activated in NeOnToolkit environment, the plug-in uses the following Eclipse extension points:

- *org.eclipse.ui.actionSets*
- *org.eclipse.ui.views*

The plug-in communicates with NeOnToolkit in the following way: (i) it gets working ontologies from projects in Ontology Navigator, (ii) and exports alignments in OWL to a project in Ontology Navigator. In order to implement this communication the plug-in uses the packages provided by NeOnToolkit, like

“com.ontoprise.ontostudio.datamodel”,

“com.ontoprise.ontostudio.gui”,

“com.ontoprise.ontostudio.io”,

“com.ontoprise.ontostudio.owl.gui”.

3.2.4. Intended Usage in the Case Studies

First, the Alignment Server plug-in is used in WP3 for finding alignments for jointly using two ontologies under the Neon Toolkit, and contextualizing an ontology with respect to another one.

In addition, the plug-in is a basic tool for finding and manipulating alignments between pharmaceutical ontologies in WP8. This work will be presented in the deliverable D.3.4.1.

3.3. Change Capturing

3.3.1. Functional Description

The purpose of this plug-in is to capture ontology changes from the NTK editor and log them into Oyster distributed registry. This plug-in also allows users to visualize the history of ontology changes. Additionally, this plug-in is in charge of applying changes received from other clients to the same ontology after Oyster synchronizes the changes in the distributed environment. Finally, this plug-in allows users to request Oyster to start the synchronization process. Specifically it provides the following functionalities:

- **Start/Stop Ontology Logging:** This functionality is available from the option "Log Changes" in the pop-up menu of the ontology objects in the NTK editor. When an ontology is being logged, the plug-in performs the following tasks on the background:
 - **Capture Ontology Changes:** Every change from the NTK editor is captured.
 - **Transform Ontology Changes:** Every change in the editor is transformed into instances of the change representation model (Change Ontology).
 - **Register Ontology Changes:** The instances of the change ontology are registered into Oyster distributed registry.
- **Visualize Ontology Changes:** This functionality is available from the "Change Log View" provided by the plug-in. From this view, users can visualize the ontologies that are being logged and for each of them, the history of changes sorted in chronological order. The list of changes is displayed in a table that shows the most relevant information (e.g. author of the change, time of the change, type of the change, etc.) and when a change is selected the complete information is displayed in the interface.
- **Start Synchronization:** This functionality is available from the option "Synchronize" in the pop-up menu of the ontology objects in the NTK editor. When this option is selected, the plug-in performs the following tasks on the background:
 - **Launch Synchronization Process:** This task request Oyster to start the Synchronization process.
 - **Apply Changes Received:** After the synchronization finishes, the plug-in applies locally (if necessary) changes received from other clients to the same ontology. When changes are applied locally, they are reflected in the NTK editor updating the local ontology with the changes received.

Synchronization: Oyster follows a synchronization approach that is a combination of a push and pull mechanism. During the synchronization, nodes contact other nodes in the network to exchange updated information (pull changes) and optionally they can push their changes to a specific node (called the push node) such that if a node goes offline before all other nodes pull the new changes, the node changes are not lost.

3.3.2. User Documentation

How to install it

- Download the compressed Plugin release from <http://ontoware.org/projects/oyster2>
- Extract and copy the Change Capturing Plugin JAR into the plugins directory inside the filesystem location where NeOn Toolkit is installed.

How to use it

The following figures show the Plug-in functionalities:

- Start/Stop ontology logging:

The logging can be started/stopped from the option "Log Changes" in the pop-up menu of the ontology objects in the NTK editor.

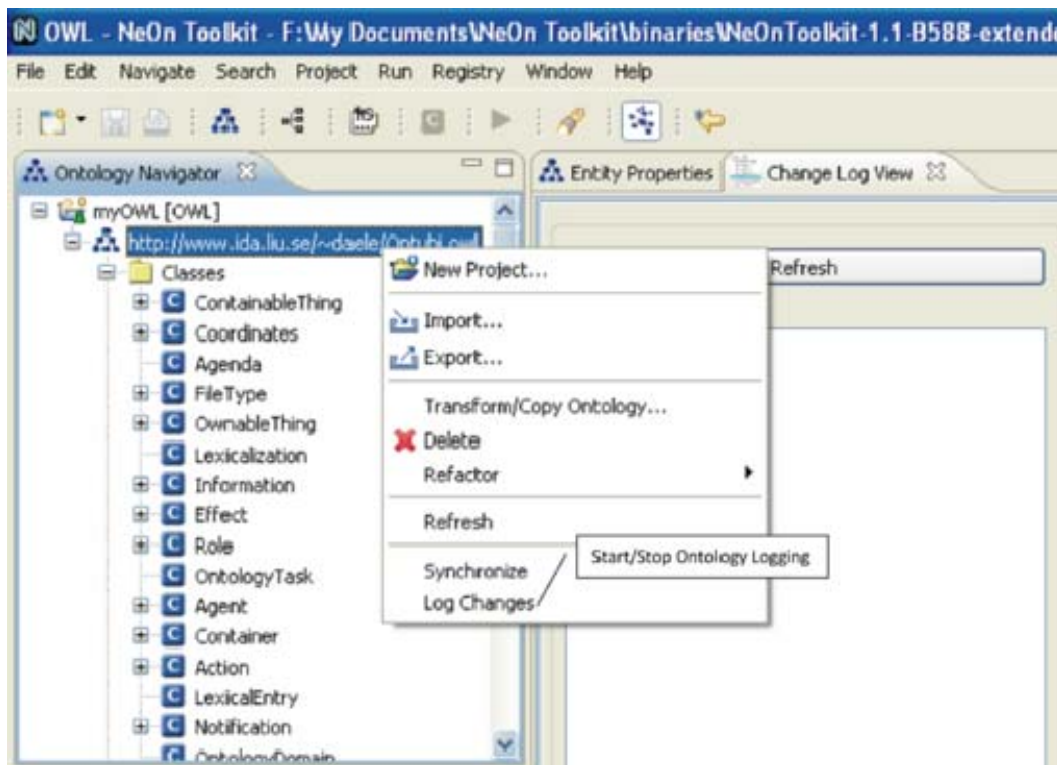


Figure 11: Starting Change Capturing

The status bar displays how many ontologies are being logged.

Visualize ontology changes:

Changes can be visualized from the "Change Log View" provided by the plug-in (Figure4).

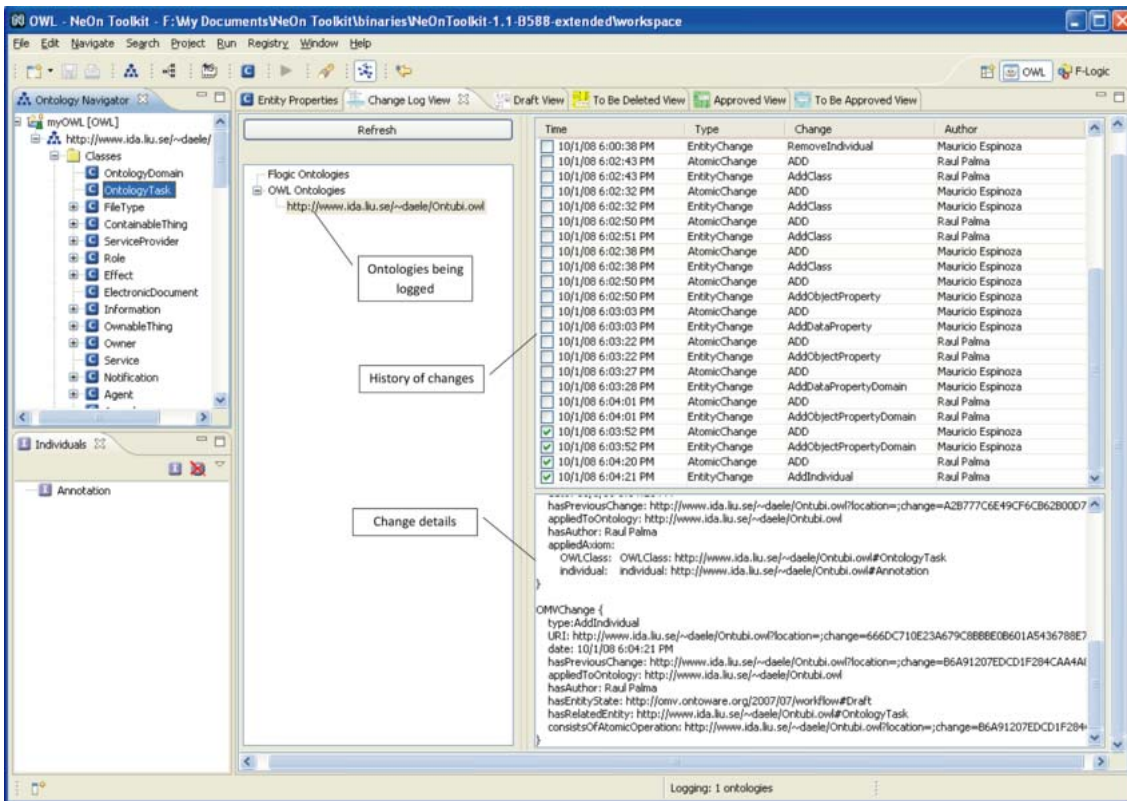


Figure 12: Visualizing Changes

- Start synchronization:

The synchronization can be started from the option "Synchronize" in the pop-up menu of the ontology objects in the NTK editor.

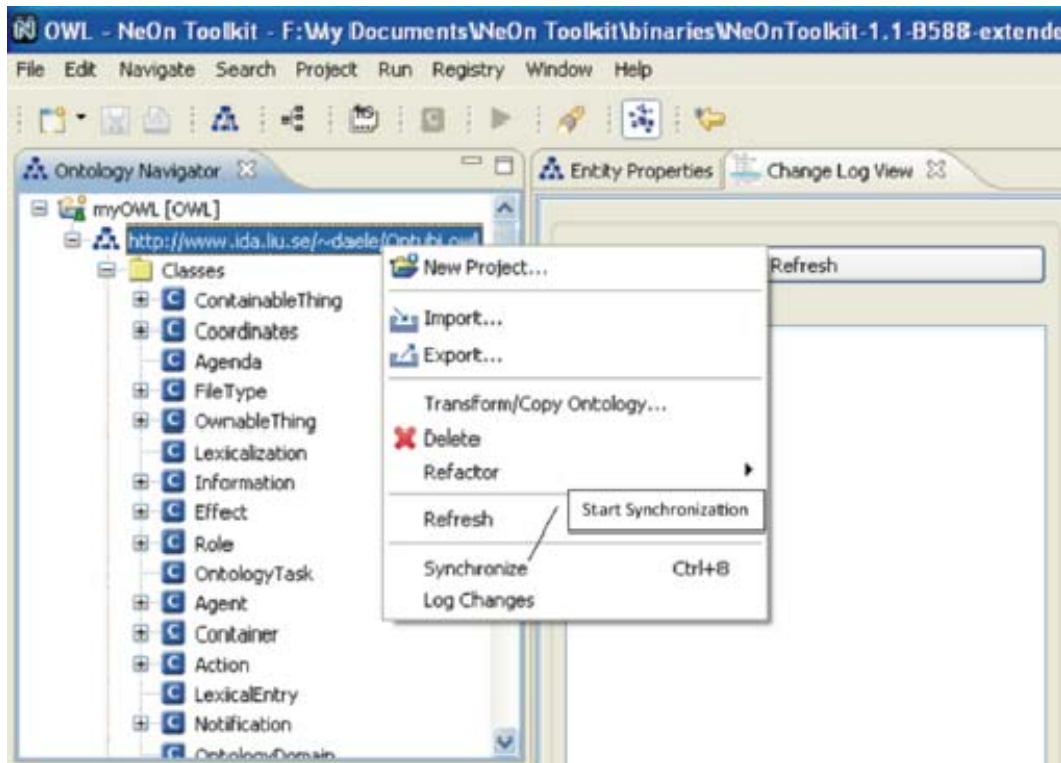


Figure 13: Change Synchronization

3.3.3. Architecture Description of integration with NeOn toolkit

A high level conceptual architectural diagram of the involved components is shown in Figure 6. The top layer represents the user related components for editing and visualizing ontologies (and related information) which consists of the NTK ontology editor and the "Change Log View" provided by this plug-in. The middle layer represents the change capturing component and finally the bottom layer consists of Oyster, our implementation of the registry services. The registry is one of the core services of the NeOn basic infrastructure layer (e.g. Repository services, Registry services, etc).

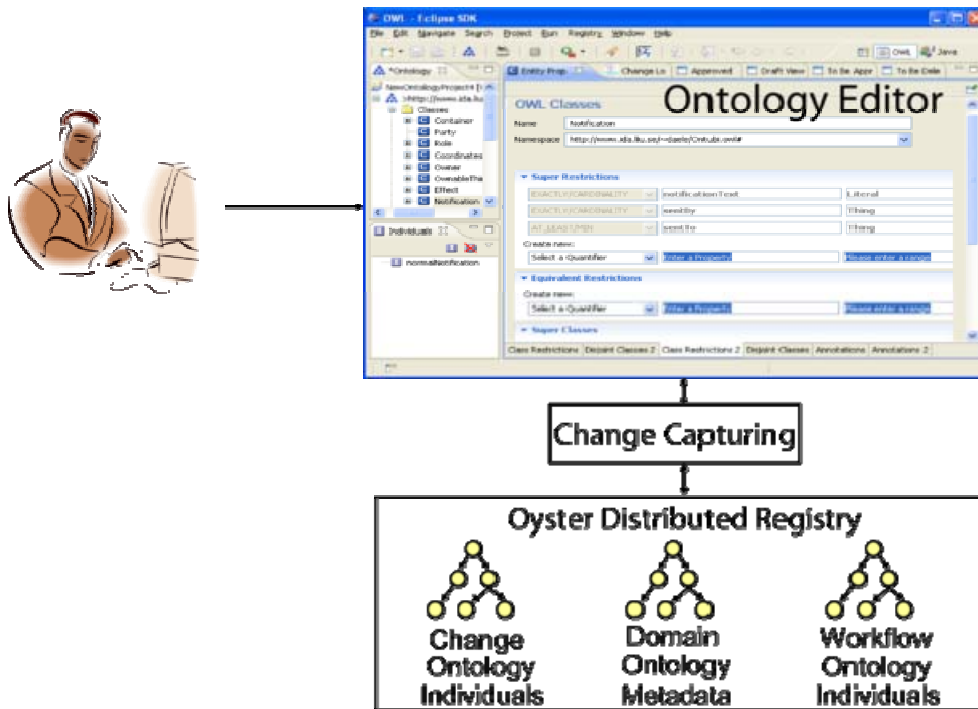


Figure 14: Architecture of Change Capturing

The following table shows the dependencies and compatibilities of the current version of the plug-in:

Version	Compatible with	Necessary plugins
Change Capturing 1.8.1	NeOn Toolkit 1.2	Org.neontoolkit.registry.api Org.neontoolkit.oyster.plugin.menu Com.ontoprise.ontostudio.io Com.ontoprise.ontostudio.gui Com.ontoprise.ontostudio.owl.gui Com.ontoprise.ontostudio.datamodel Datamodel DatamodelBase Util Org.junit4

3.3.4. Intended use in case study

Oyster will be used in WP7 to support tasks related to the management of provenance and statistics.

- **Motivation of using the change capturing plug-in**

One of the goals of the FAO use case partner is that fisheries ontologies produced within WP7 will underpin the Fisheries Stock Depletion Assessment System (FSDAS). However, for such a

dynamic domain like fisheries that is continuously evolving, we will need to provide the appropriate support for a successful implementation and service delivery of the FSDAS. In particular, for this task we need to support a collaborative editorial workflow that will allow Ontology editors to consult, validate and modify ontologies keeping track of all changes in a controlled and coherent manner. In this scenario, ontology editors are usually developing/maintaining ontologies collaboratively in a distributed environment. The change capturing plug-in is crucial component in the infrastructure to support the editorial workflow: first, changes have to be monitored and captured from the ontology editor. Those changes should be formally represented and stored in Oyster. Using the registry functionalities, those changes will be searched and retrieved by the change capturing plug-in to show the history of ontology changes. Finally, after the registry propagates those changes to the distributed copies of the same ontology, the change capturing plug-in updates the local copy of the ontology to reflect changes in the distributed copies.

- **Benefits/Advantages of using the change capturing plug-in**

- The change capturing plug-in automatically keeps track of the ontology changes
- The change capturing plug-in performs the required tasks on the background
- The change capturing plug-in allows users to check the history of ontology changes
- The change capturing plug-in applies locally changes received from the distributed copies of the same ontology after Oyster performs the synchronization process.

- **Data sets**

FSDAS ontologies

- **A usage example aligning to WP7 case study.**

For the use case UC-9.1 this plug-in is in charge of capturing ontology changes and for UC-9.2, this plug-in allows ontology editors to view the logs about changes to ontology elements. The information about each change is represented according to the change ontology (see Figure7) and includes among others the following minimum fields:

- change
- description
- operation executed
- timestamp
- URI
- user

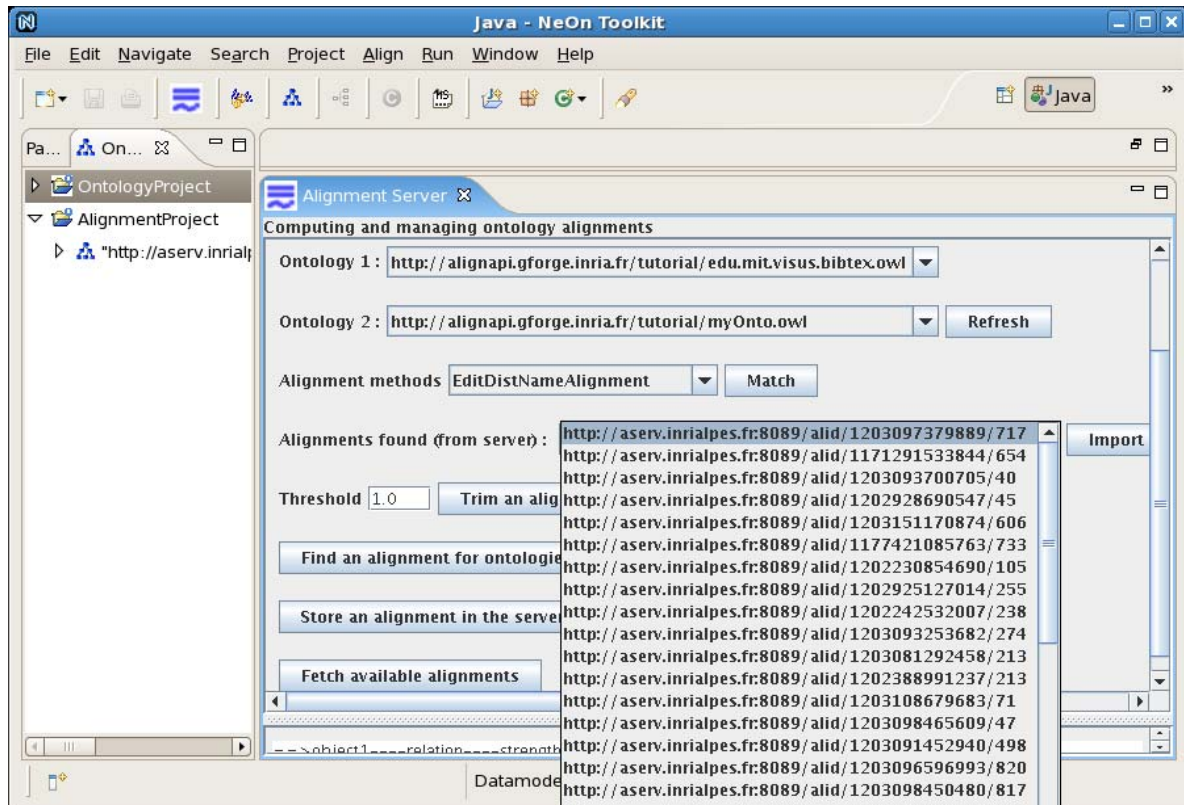


Figure 15: Alignment Server: Fetching Alignments

3.3.5. Integration into the NeOn Toolkit

The Alignment Server plugin creates a view in the NeOn Toolkit which can be invoked via the menu bar or the button "Alignment View". This view consists of two parts. The first one is designed such that users would enter easily input data and the second one visualizes results.

- Eclipse Extension Points
 - org.eclipse.ui.views
 - org.eclipse.ui.actionSets
- Ontostudio extension points
 - com.ontoprise.ontostudio.datamodel
 - com.ontoprise.ontostudio.io
 - com.ontoprise.ontostudio.gui
 - com.ontoprise.ontostudio.ontomap

3.3.6. Intended Usage in the Case Studies

The alignment server is intended to be used in the case studies to identify and manage alignments between heterogeneous ontologies.

3.4.

3.5. Cicero

3.5.1. Functional Description

The main purpose of the Cicero plugin for the NeOn toolkit is to keep track of discussions between the developers and users of an ontology. While the actual discussions are held in the Cicero-Wiki on a central server³, the toolkit plugin allows for establishing links between elements in an ontology (e.g. classes or properties) and discussions that influenced their design. These discussions are then used by the ontology developers for understanding the design rationale of specific ontology elements.

Altogether, the plugin supports the following functionality:

- Start discussions from within the NeOn toolkit.
- Establish provenance links between ontology elements and discussions that influenced their design.
- Search discussions that are relevant for understanding the design rationale of specific ontology elements.
- Show details of discussions in an integrated web browser.

3.5.2. User Documentation

How to Use in an Ontology Engineering Project

In general, one can distinguish two different cases in which argumentation plays an important role in enabling collaboration between the participants of an ontology engineering project (this includes the developers but also the future users of an ontology):

- First, there are activities during which argumentation data is actively created, e.g. by discussions between the participants. In this case, the argumentation framework has the role of structuring the discussion process, helping in systematically exploring possible solutions and capturing the pro and contra arguments. Argumentation support is then a means of having more efficient discussion and decision taking processes.
- Second, there are activities where previously recorded discussions are used for understanding the design rationale of elements in the ontology network. For example, in the DILIGENT methodology the argumentation data created during the local adaptation of an ontology is used by the control board during the analysis and revision activity. In this case, recorded discussions are part of the ontology documentation.

The most important activities of an ontology engineering project during which discussion data may be actively created are the [ontology specification](#), [ontology conceptualization](#), [ontology formalization](#) and [ontology implementation](#) phases. All four activities require reaching a consensus between the participants about the requirements of the ontology network and how they should be implemented. But the recorded discussions may also be used for understanding the decisions made during previous activities (e.g. during ontology formalization one has to understand the decisions from the ontology conceptualization activity). Reaching a consensus or explaining decisions to collaborators is not only needed during the previously mentioned activities of the ontology development process (i.e. during building an ontology from scratch) but it may also be needed during [reusing](#) or reengineering [ontological](#) and [non-ontological](#) resources, or during the [alignment](#) with other ontologies.

³ More details about the Cicero-Wiki are available at <http://cicero.uni-koblenz.de>

Discussions are an important part of the [ontology documentation](#), which should also refer to explanatory comments generated during the entire ontology building process. The recorded discussions help in keeping track of the design rationale all the way through the ontology engineering process, and keeping the design rationale up to date by amending it with additional arguments. It makes also sense that Cicero is used if only a single person is responsible for developing an ontology. In that case, the developer documents his/her design rationale either for future users of the ontology or for himself/herself if the corresponding part of the ontology has to be changed at a later point in time.

Recording discussions makes it easier to resume a previous activity in the ontology engineering process, if it turns out that a decision taken during that activity is underspecified or not appropriate. In this case, the discussion that led to the decision may be easily resumed because all stakeholders that participated in the decision making process are identified by the recorded discussion. Resuming a discussion may additionally be useful during the maintenance phase of an ontology, e.g. if there were changes to the requirements that affected the decision.

In general, supporting the argumentation process is important in each situation where either several users collaboratively decide an issue or where a user by himself creates an ontology element that should be later used as input for the activity to be developed by another user. In the latter case, the collaboration is facilitated by enhanced and more complete ontology documentation.

User Interface

The Cicero plugin is visible at two different locations in the NeOn toolkit. First, it provides an additional subpage for the entity properties of an ontology. The subpage is used for activating the Cicero support for that specific ontology. Second, the plugin extends the context menu for all elements in an ontology. The context menu can be used for creating new discussions, annotating ontology elements with already existing discussions and for showing a list of discussions annotated to the current ontology element. More details about the subpage and the context menu are available below.

Property Page of Ontologies

The Cicero-plugin provides a new (sub-)property page called **Argumentation Settings** for an (F-Logic or OWL) ontology element. You get this (sub-)property page by clicking on the label of the ontology in the Ontology Navigator (see picture below).

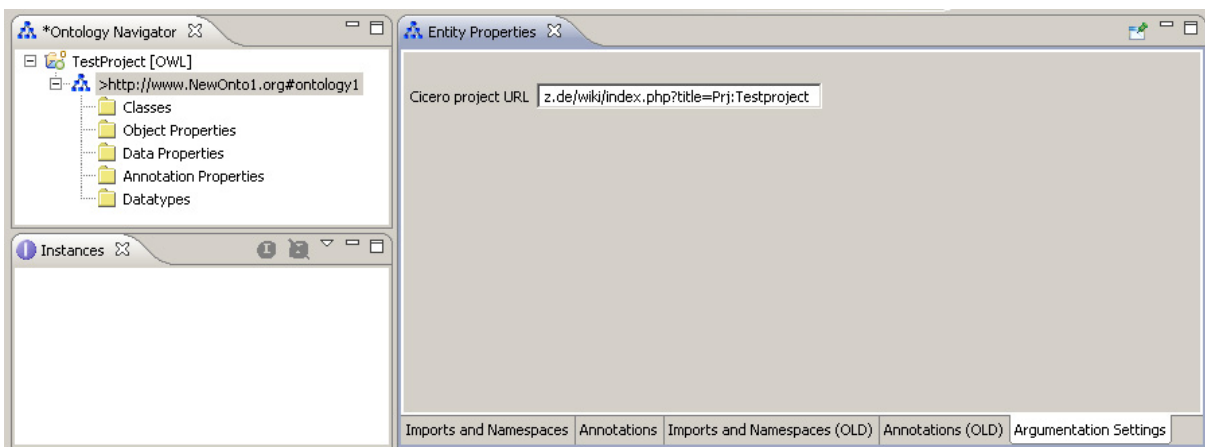


Figure 16: Cicero Configuration

The property page provides a text field **Cicero project URL**. Here the user can enter a (valid) URL of the project in a Cicero-Wiki-installation in which the currently selected ontology should be discussed. The value entered in the text field can then be persistently saved in the datamodel by clicking on the "Save"-button.

For testing you can enter <http://cicero.uni-koblenz.de/wiki/index.php?title=Prj:Testproject> as the project URL.

Context Menu for Ontology Elements

The Cicero plugin extends the context menu of elements in an OWL and/or F-Logic ontology. You get the context menu by right-clicking on an ontology element. The context menu allows for annotating ontology elements with corresponding discussions and for showing a list of previously annotated discussions. Currently, the plugin supports the context menu for classes, individuals, object-, data- and annotation-properties (for OWL ontologies) and for concepts, individuals, attributes, relations, rules and queries (for F-Logic ontologies). The context menu contains the following entries:

- **Annotate with Issue** with which the user is able to annotate the selected elements with an Issue-URL.
- **Create Issue** with which the user is able to create a new issue on the Cicero-Wiki from within the NeOn-Toolkit. The created issue is annotated for the selected ontology elements.
- **Show Issues** with which the user can access a list of all direct and related issues for the currently selected ontology element.

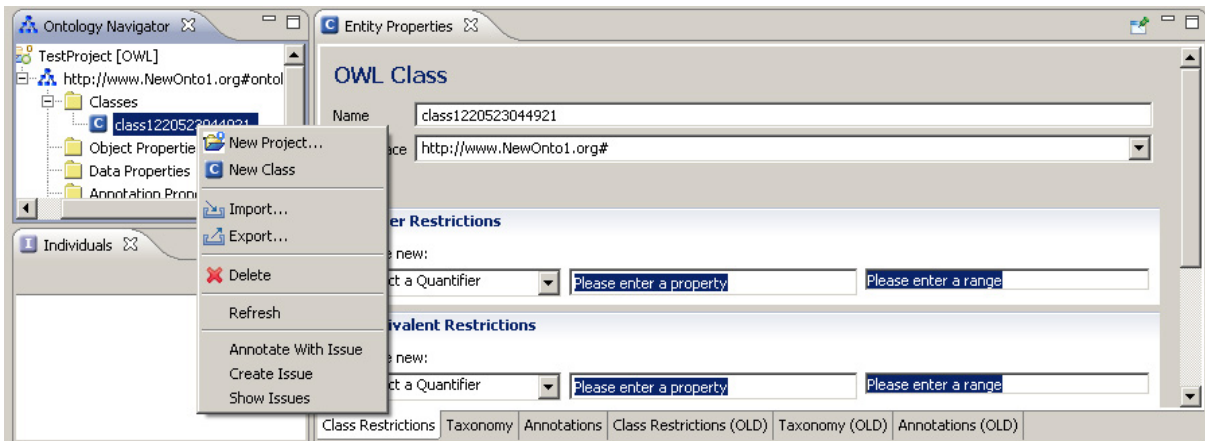


Figure 17: Cicero Context Menu

Annotate with Issue

After clicking on the **Annotate with Issue**-menu entry the window shown in the picture below appears. It contains a text field in which the user can enter an Issue-URL. For testing, any URL of an issue in the [Testproject4](http://cicero.uni-koblenz.de/wiki/index.php?title=Prj:Testproject) might be used.

4 <http://cicero.uni-koblenz.de/wiki/index.php?title=Prj:Testproject>

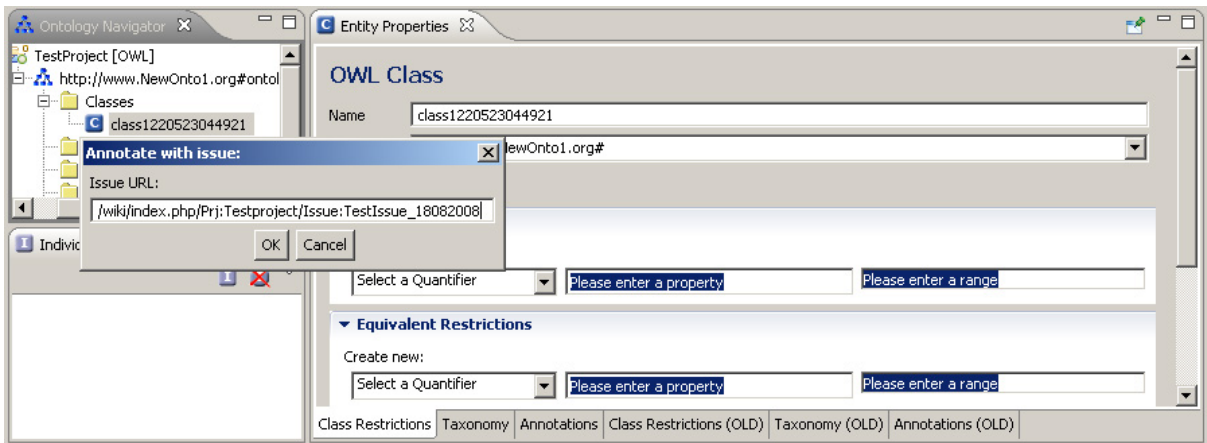


Figure 18: Annotating with an Issue in Cicero

By clicking the **OK**-button the entered URL is annotated for all currently selected ontology elements. A dialog appears with a message that the URL has been successfully added.

Create Issue

After selecting the **Create Issue**-entry, a new window appears in which the user needs to enter his **valid login data** (login name and password) to the Cicero-Wiki. By activating the checkbox **Remember me login** the entered login data is stored in a local text file with encryption for the current ontology project. With this the user doesn't need to enter the login data the next time for the same ontology project.

If you are using <http://cicero.uni-koblenz.de/wiki/index.php?title=Prj:Testproject> as cicero project for testing, please use the login data listed in http://cicero.uni-koblenz.de/wiki/index.php/Main_Page.

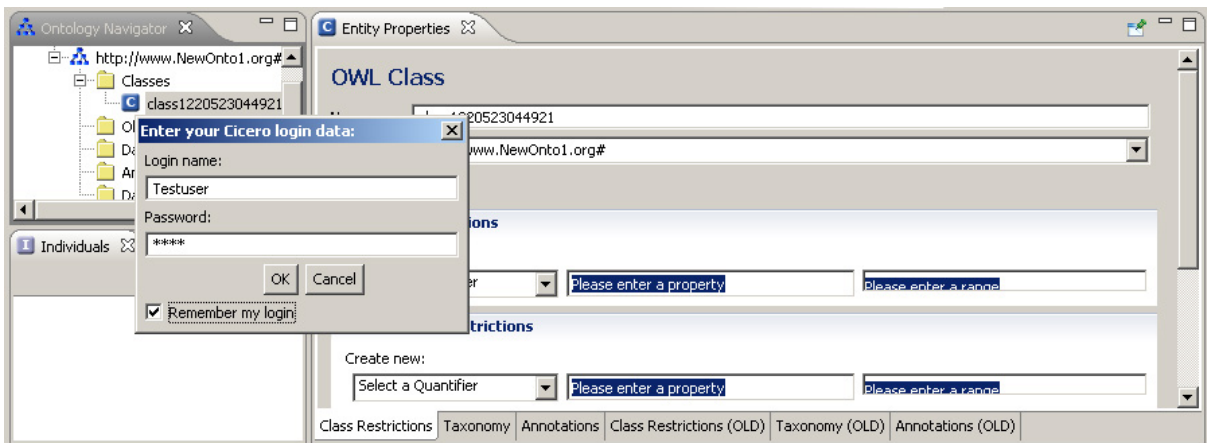


Figure 19: Creating an Issue

After pressing the **OK**-button a new window opens. It contains two text fields for creating a new issue. The user needs to enter an **issue title** and optionally can also provide an **issue description**. For sending the issue creation-request, the user needs to click on the **OK**-button.

Show Issues

If the user decides to select the **Show Issues**-entry, the window shown below appears.

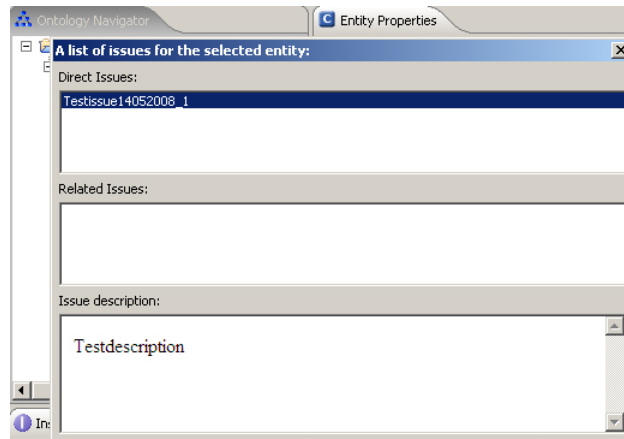


Figure 20: Showing an Issue

It consists of two lists:

- the first showing all issues directly annotated to the currently selected ontology element and
- the second showing all related issues which are annotated in ontology elements having a certain relation to the currently selected ontology element.

Whenever a user selects an issue title in one of the lists, the corresponding description is shown in the textarea at the bottom. By double-clicking on an issue title in one of the lists, an external browser will be opened with the respective URL. Note that if your login data to the Cicero-Wiki is not yet known for the current ontology project, a window for entering the login-data will appear before the overview window is shown.

3.5.3. Integration into the NeOn Toolkit

The following extension points are used within the plugin:

- Eclipse extension points:
 - **org.eclipse.ui.popupMenus**
 - **org.eclipse.ui.startup**
- NeOn toolkit extension points:
 - **org.neontoolkit.gui.entityProperties**

The Cicero Plugin for NeOn-Toolkit consists of three separate plugins:

- **org.neontoolkit.cicero** which contains the ontology-independent functionality,
- **org.neontoolkit.cicero.flogic** which contains the functionality needed for FLogic-ontologies and
- **org.neontoolkit.cicero.owl** which contains the functionality needed for OWL-ontologies.

The plugins **org.neontoolkit.cicero.flogic** and **org.neontoolkit.cicero.owl** depend both on the general plugin **org.neontoolkit.cicero**.

Since version **1.0.1** the plugins are now available as parts of so called **features**. The features are:

- **FLogic-feature** which contains the plugins **org.neontoolkit.cicero** and **org.neontoolkit.cicero.flogic**.
- **OWL-feature** which contains the plugins **org.neontoolkit.cicero** and **org.neontoolkit.cicero.owl**.

3.5.4. Intended Usage in the Case Studies

The Electronic Invoice Management of the Pharmalnova cluster offers a good domain to test and evaluate the Cicero wiki and the Cicero plugin in the context of a collaborative scenario. Pharmalnova is a cluster that consists of several Spanish pharmaceutical laboratories. In a collaboration process the participants of the Pharmalnova cluster are working to define a common invoice model which can be used for exchanging invoices between the different laboratories. Every time a new member is joining the cluster, its invoice model has to be integrated into the already existing model of the Pharmalnova cluster. A further reason for creating a new version of the invoicing ontology may e.g. be changes in legal requirements for invoices.

Agreeing on a new version of the ontology is a difficult task because the different stakeholders in the development process are distributed over multiple locations. There exist many problems like organizing and coordinating face-to-face meetings with all stakeholders. One objective of using the Cicero wiki and plugin may thus be to reduce the costs and personal effort that are required for travelling. An additional benefit of using Cicero may be the improved documentation of the resulting ontology.

3.6. Collaborative Workflow Support

3.6.1. Functional Description

The main purpose of the **Workflow Support** plugin is to track the ontology changes and then manipulate these changes according to the role of a user, where the user could log on / out or register in a preference page. Specifically,

For a **viewer**:

- Browse the changes in different status which are made by other subject experts or validators.

For a **subject expert**:

- Select the elements in Draft status and sent them to the validator for approval, by changing the status of the elements from "Draft" to "To be approved".

For a **validator**:

- Approve elements and thus to move them from the "To be Approved" status to the "Approved" status.
- 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.
- Move back to the "To be approved" status an element already approved, if they consider it necessary
- 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
- Accept the deletion and definitely destroy an element in the "To be deleted" status

3.6.2. User Documentation

In this section, we first introduce the preference page for a user to log on / out or do registration. Then the workflow support plugin will be demonstrated to show how it works according to different roles. Before you follow the steps below, please make sure you have started registry by following the steps in NeOn Toolkit: Registry --> Start registry, or you could simply click on the button with some blue dots in the tool bar of NeOn Toolkit.

Collaborative Development Preference:

1. You could find this preference page by following these steps in NeOn Toolkit: Window --> Preferences --> Collaborative Development Preference.
2. If you are a new user, you could do registration in the “Register” part by filling in your first name and last name and choosing a role for yourself. Then you can click the button of “Register”. If there is a person who has the same first name and last name as yours, you need to use a different first name or last name. Otherwise, you will be registered and log in automatically.
3. If you have registered already, you could log in the “Log in /out” area by inputting your first name and last name.

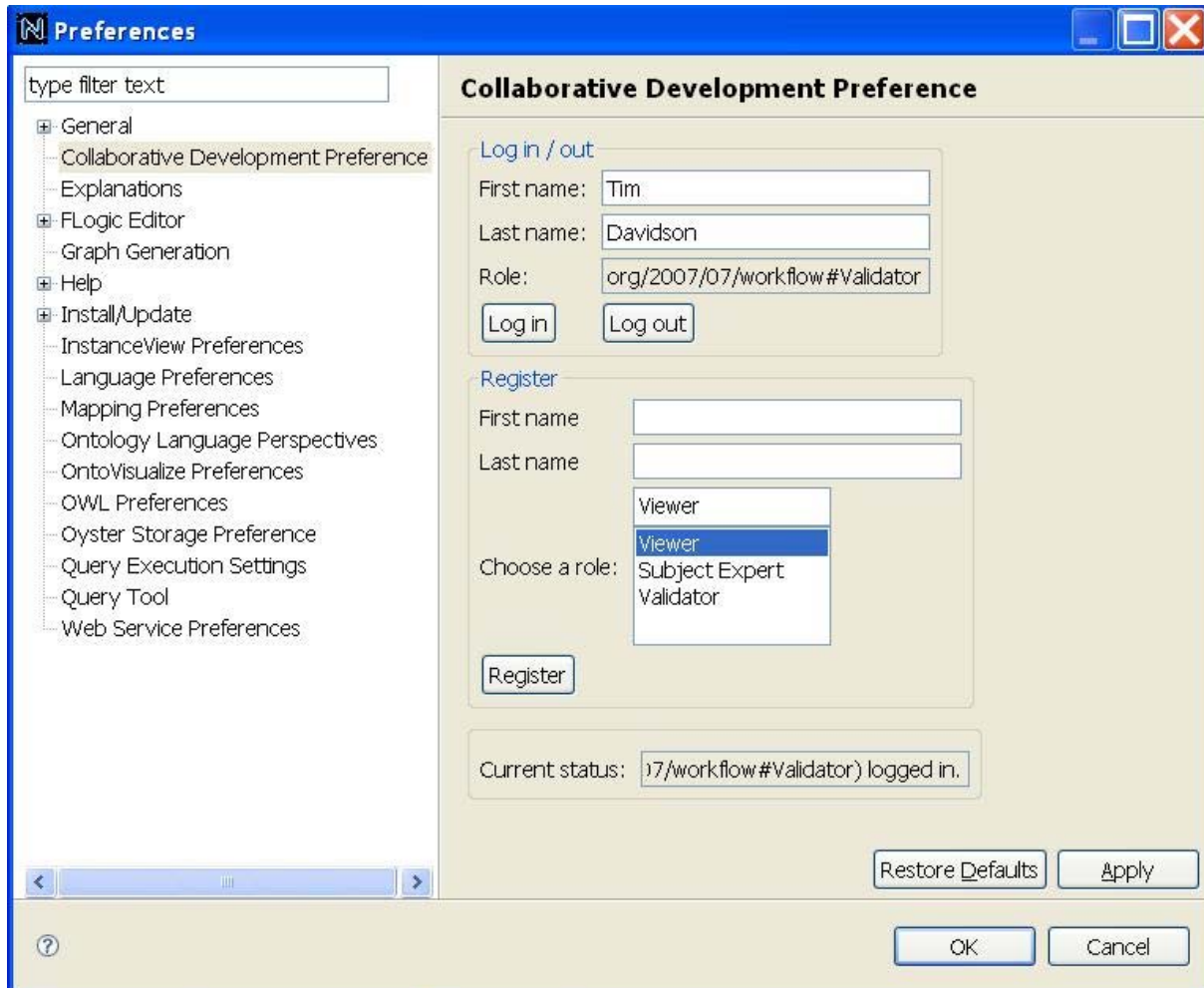


Figure 21: Collaborative Development Preferences

Workflow Support:

1. The four views in this plugin can be found by following these steps: Window --> Show View --> Other --> Workflow.
2. Please log on first in the preference page which have been introduced above and choose an OWL ontology to be tracked by right-clicking an OWL ontology in Ontology Navigator in NeOn Toolkit and then select "Log changes". We will introduce the four views according to the role you have registered.
3. If you are a viewer, you have no right to make some changes. But you could browse all the changes in different states (by clicking on the button of “Refresh Changes List”) which are made by the subject experts or validated by the validators. All the changes are shown in grey.

4. If you are a subject expert, you could make some changes for an OWL ontology in the Ontology Navigator in NeOn Toolkit. In the Draft view, you could manipulate these changes by submitting some changes to be approved or deleting the changes. See Figure 22: Draft views as an example. Please note that you cannot manage the changes made by other users which are shown in grey.

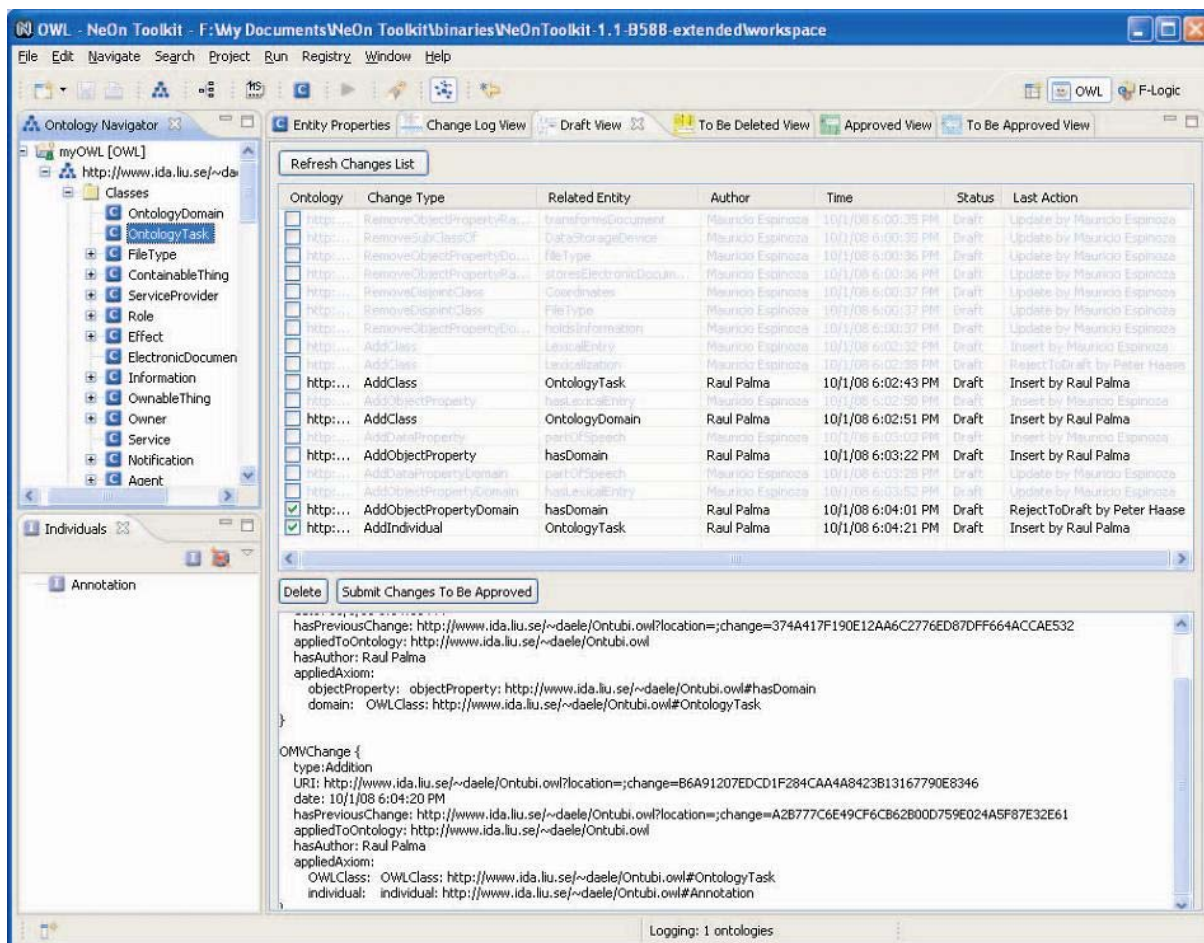


Figure 22: Draft view

5. If you are a validator, you are able to manipulate **To Be Approved View**, **To Be Deleted View** and **Approved View**. Specifically, **(1)** in To Be Approved View, a validator can decide to accept the changes by changing the change status from "to be approved" to "Approved" or reject them by changing the status to "draft". **(2)** In Approved View, the validator can reject the proposes by changing the status from "approved" to "to be approved" or propose to delete by changing the status from "approved" to "to be deleted". **(3)** Finally, in To Be Deleted View, the validator can delete the changes in "to be deleted" status permanently or reject the proposals by changing the status from "to be deleted" to "approved".

3.6.3. Integration into the NeOn Toolkit

Workflow support plugin has been developed as a view in NeOn toolkit and it can capture the changes in Ontology Navigator.

- Eclipse extension points
 - org.eclipse.ui
 - org.eclipse.core.runtime
 - org.eclipse.core.resources

- NeOn Toolkit extension points
 - [com.ontoprise.ontostudio.datamodel](#)
 - [com.ontoprise.ontostudio.owl.gui](#)
 - [com.ontoprise.ontostudio.owl.model](#)
 - [datamodel](#)
 - [datamodelBase](#)
 - [util](#)

- Oyster extension points
 - [org.neontoolkit.changelogging](#)
 - [org.neontoolkit.registry.api](#)
 - [org.neontoolkit.oyster.plugin.menu](#)

3.6.4. Intended Usage in the Case Studies

The Workflow support plugin can be applied in FAO case study to provide a mean for ontology engineers to develop ontologies collaboratively.

3.7. DIG Interface

3.7.1. Functional Description

The purpose of the DIG Plugin is the implementation of the DIG Interface version 1.1. The DIG Description Logics Interface Version 1.1 is a specification for defining a new interface for DL Systems. It is effectively an XML Schema for a DL concept language along with ask/tell functionality. In two words, the DIG interface provides a standardized way to access and query a reasoner. The user initially chooses the desired action she is interested in. Then the ontology is translated into the DIG interface and sent to the reasoner along with the queries that have been posed. After the query processing inside the reasoner has taken place, the reasoner sends back to the user the response encoded in the DIG Interface and the user can extract the answer to her query. The interested reader is referred to the [protocol specification](#) for further information.

The entire concept language and tell/ask functionality is enough to capture every functionality that is usually provided by a reasoner. In the context of the use cases in the NeOn-Project the reasoning tasks that are of utmost importance to us are ontology coherency and classification. More concretely:

- **Ontology Coherency:** The reasoner takes as input the (translated into the DIG protocol) ontology and for each concept it returns true or false, depending on whether the corresponding concept is satisfiable or unsatisfiable, respectively.
- **Classification:** The reasoner takes as input the (translated into the DIG protocol) ontology and returns the inferred classification of the various concepts, as opposed to the explicit one that the user initially sees. Moreover, for every concept in the inferred hierarchy we get whether it is satisfiable or not.

It must be highlighted at this point that the DIG Description Logics Interface that has been partially implemented in this plugin does not come bound to any reasoner. Contrary to that, it only provides an interface to query any reasoner that supports the DIG Protocol. The motivation behind this is that the current plugin is a general-purpose plugin that is intended to be used with different

reasoners, so we found it meaningful not to restrict it to a particular reasoner, but to rather provide the user with the flexibility to do that themselves, in full accordance with their needs.

3.7.2. User Documentation

In this section, we will show how to use the DIG plugin to achieve the aforementioned functionalities.

Specifying the Reasoner Preferences

Independent of the functionality that we are interested in we must first specify the reasoner IP address and port, since we do not a priori know where our reasoner is running. Indeed, any reasoner that implements the DIG interface 1.1 and additionally provides either of the functionalities under consideration can be used for the reasoning task. More specifically, we have:

1. The user interface for specifying the reasoner is the **DIG Reasoner Setting** preference page under the OWL Preferences in the **preference dialog**, which is accessible via the **Window->Preferences** menu group.
2. **Setting the preferences:** After invoking the view we specify the host name and he port of the reasoner and we then click the OK button, as depicted in Figure 23: Setting the Reasoner Preferences.

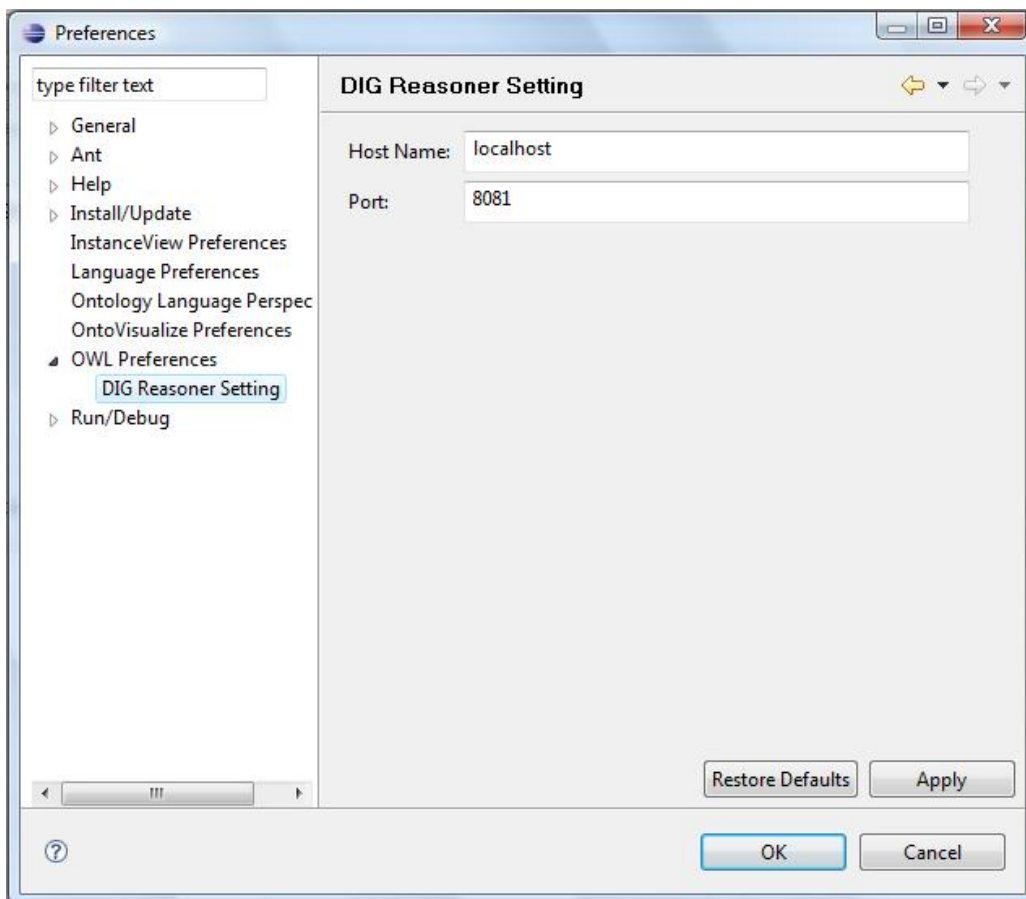


Figure 23: Setting the Reasoner Preferences

How to use Ontology Coherency

1. The user interface for checking an ontology for coherency is a view of **Coherency/Satisfiability**.

2. **How to invoke our view?** To check an ontology for coherency, right-click the ontology in the Ontology Navigator in NeOn Toolkit and then select the item of **Check ontology coherency**. The view of **Coherency/Satisfiability** will be activated. At the same time, the logical and physical URIs of this ontology will be shown. Besides, we also show some information about the number of concepts and how many among them are unsatisfiable.

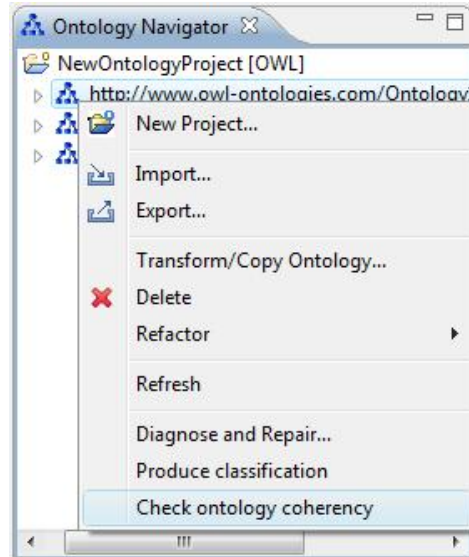


Figure 24: Invoking the coherency view

3. **Getting back the results:** In the end, we get back the results of the coherency check. In case the ontology is not coherent, the "Unsatisfiable Concepts" button will be activated and by clicking on it we will get all unsatisfiable concepts in a table.

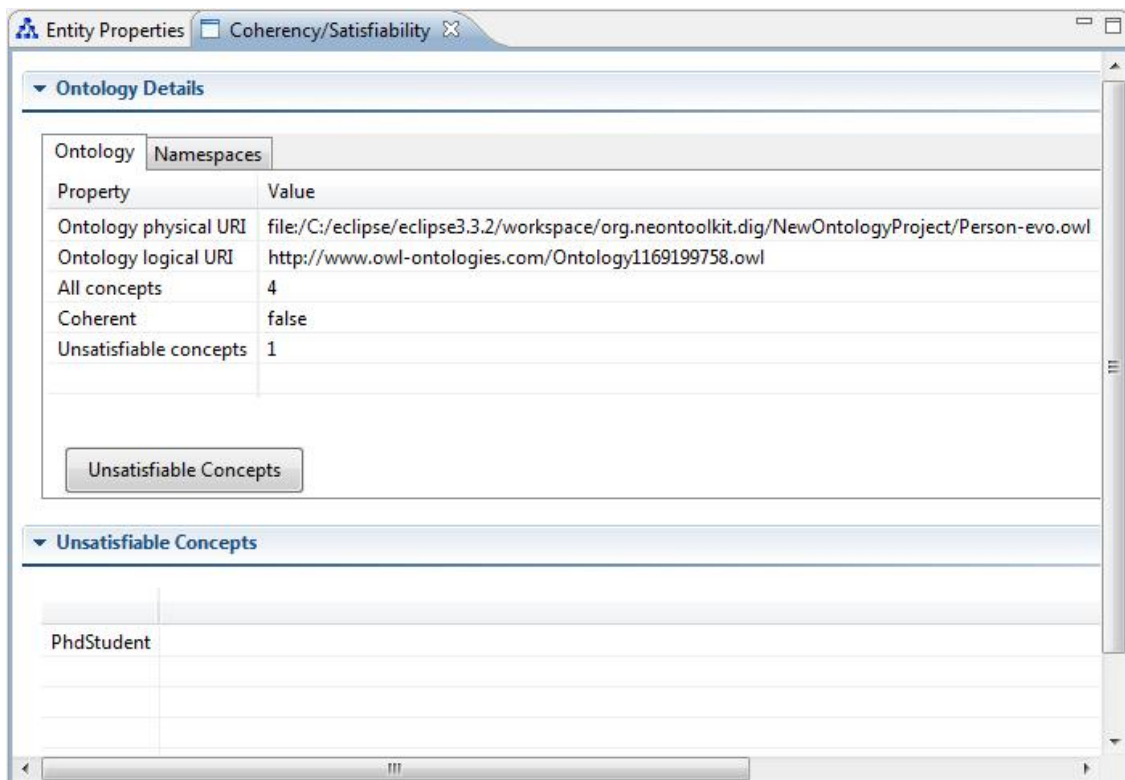


Figure 25: Getting back the results

How to use Ontology Classification

1. The user interface for ontology classification is a view of **Entailed Subsumption Hierarchy**.
2. **How to invoke our view?** To produce the classification, right-click the ontology in the Ontology Navigator in NeOn Toolkit and then select the item of **Produce Classification**. The view of **Entailed Subsumption Hierarchy** will be activated. At the same time, the name of the project where the ontology belongs to and the logical URI of this ontology will be shown.
3. **Getting back the results:** In the end, we get back the results of the classification. The inferred hierarchy contains the taxonomies and additionally for each concept in the classification extra information on whether the concept is satisfiable or not, according to the image next to it, as shown in Figure 26.

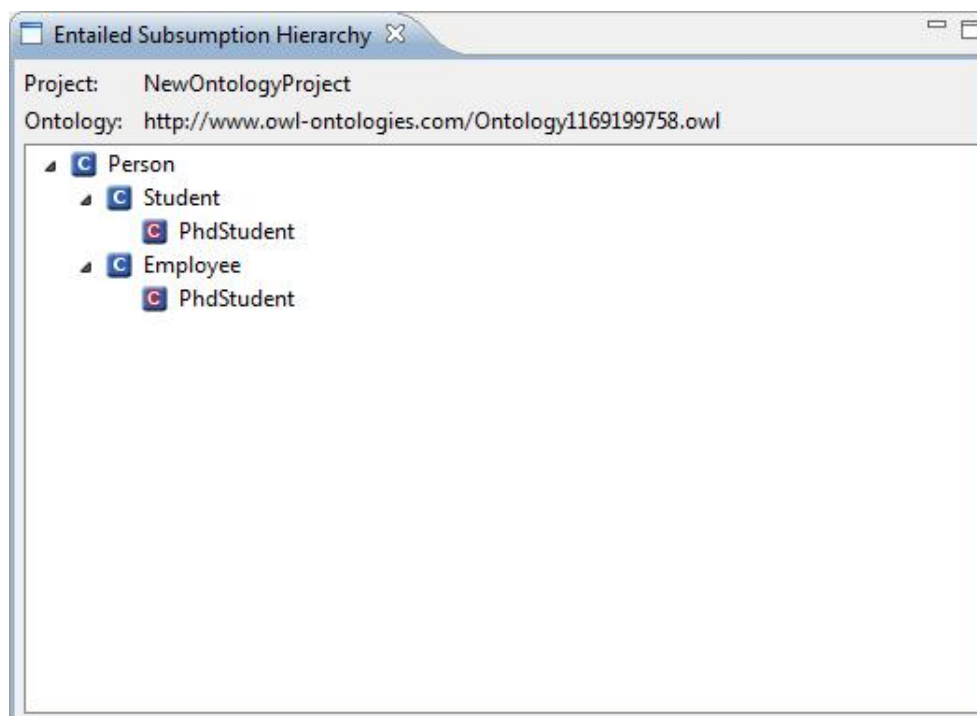


Figure 26: A classification example

3.7.3. Integration into the NeOn Toolkit

The DIG plugin provides reasoning services through the reasoning protocol and it could thus be characterized as a reasoning/engineering component. Of course, it comes along with a practical user interface, as depicted in the User Documentation Section.

The plugins (other than the eclipse plugins) upon which it is dependent are:

- com.ontoprise.ontostudio.datamodel
- com.ontoprise.ontostudio.gui
- com.ontoprise.ontostudio.owl.gui
- org.neontoolkit.gui.

Also, the extensions it uses are the following:

- `org.eclipse.ui.views`: this extension is used for defining the Ontology Coherency and Classification views for giving back to the user the results.
- `org.eclipse.ui.popupMenus`: this extension is used for defining the Ontology Coherency and Classification options in the pop-up menu, as explained in the previous section.
- `org.eclipse.ui.preferencePages`: this extension is used for defining the Reasoner Preference page in the Reasoner Preferences Section, as was also shown in the previous section.
- `org.eclipse.help.toc`: this extension is used for defining the online help. In the case of the DIG plugin, the help consists of the introduction section and the User Documentation section.

Last, it currently doesn't provide an extension point to other plugins and no other plugin is currently dependent upon it (though that could change, when the OBDA functionality is integrated in the NeOn toolkit as discussed in the next section).

3.7.4. Intended Usage in the Case Studies

The DIG plugin is concentrated on two prominent reasoning tasks, namely ontology coherency and classification, which means that it can be used in a wide variety of contexts, where these tasks are needed. The fact that the DIG interface is not attached to any particular reasoner but it rather presents a general protocol specification for a reasoning interface makes it very flexible.

As far as future intended cases are concerned, we have mainly developed the DIG plugin and extended it with the [OBDA](#) functionality, to make use of the [QuOnto](#) reasoner's reasoning services. We find that such functionality is at least promising and potentially very useful for the NeOn use cases, where ontology schemata are expected to be rather simple, whereas most concern is placed on data.

3.8. LabelTranslator

3.8.1. Functional Description

Multilinguality in ontologies is nowadays demanded by institutions worldwide with a huge number of resources available in different languages. To solve this problem we propose LabelTranslator, a plugin that automatically localizes ontologies. LabelTranslator takes as input an ontology whose labels are described in a source natural language and obtains the most probable translation into a target natural language of each ontology label.

Altogether, the plugin supports the following functionalities

- Obtains the most probable translation for each ontology label.

LabelTranslator relies on two advanced modules for this task. The first, the so-called translation service, automatically obtains the different possible translations of an ontology label by accessing different linguistic resources. This service also uses a compositional method in order to translate compound labels (multi-word labels). The second module, the translation ranking, sorts the different translations according to the similarity with its lexical and semantic context. The method relies on a relatedness measure based on glosses to disambiguate the translations. This is done by comparing the senses associated to each possible translation and their context.

- Captures all the linguistic information associated with concepts using a linguistic model repository (LIR).

The LIR, Linguistic Information Repository, is a portable model that can be associated to any term of an OWL ontology by means of an OWL meta-ontology. The main classes that compose the LIR (Lexicalization, Sense, Definition, UsageContext, Note and Source) are organized around the LexicalEntry class, which is related to any ontology term (by means of the hasLexicalEntry relation). The set of LIR concepts enables a complete description in natural language of the ontology term it is associated to. Additionally, by means of typical lexical relations either in the same language (e.g. hasSynonym) or across languages (hasTranslation), the LIR organizes linguistic information within the same natural language and among different languages in order to provide a multilingual set of information that enables ontology localization.

- Uses a synchronization mechanism to maintain the ontology and linguistic information synchronized

Addition of new terms in the ontology or deletion of existing terms is controlled using the advanced change tracking (based on Resource Delta⁵) used in NeOn toolkit. This mechanism is able to capture changes even when ontological terms have changed their position within the ontology model. By adopting this feature, LabelTranslator can accurately identify the minimal set of changes needed to adjust the structure of the linguistic model, a critical step to ensure that a matching change is made in the localized ontology.

3.8.2. User Documentation

The LabelTranslator plug-in is visible at two different locations in the NeOn toolkit. First, it provides an additional subpage for the linguistic information associated to each element of an ontology (classes, object properties and data properties). Secondly, the plug-in extends the context menu for all elements in the ontology. The context menu can be used for translating the current ontology element from a source language to a target language.

Additionally, LabelTranslator uses some views of the NeOn toolkit to load the ontology and store the multilingual results. More details about the subpage, the context menu and other functionalities of LabelTranslator are available below.

Initializing Linguistic Information of an Ontology

When the ontology editor user imports/creates a new OWL ontology in NeOn, then the LabelTranslator plug-in automatically builds an empty linguistic model associated to the ontology under consideration. This model is used by our plug-in to store the linguistic information associated with each ontology label. In order to show the linguistic subpage, the user has to select an ontology element (class or property, for example) in the Ontology Navigator.

Then (s)he chooses the Linguistic Information page shown in the Entity Properties View. All fields and tables that show linguistic information correspond to the LIR model.

⁵ A resource delta represents changes in the state of a resource tree between two discrete points in time.

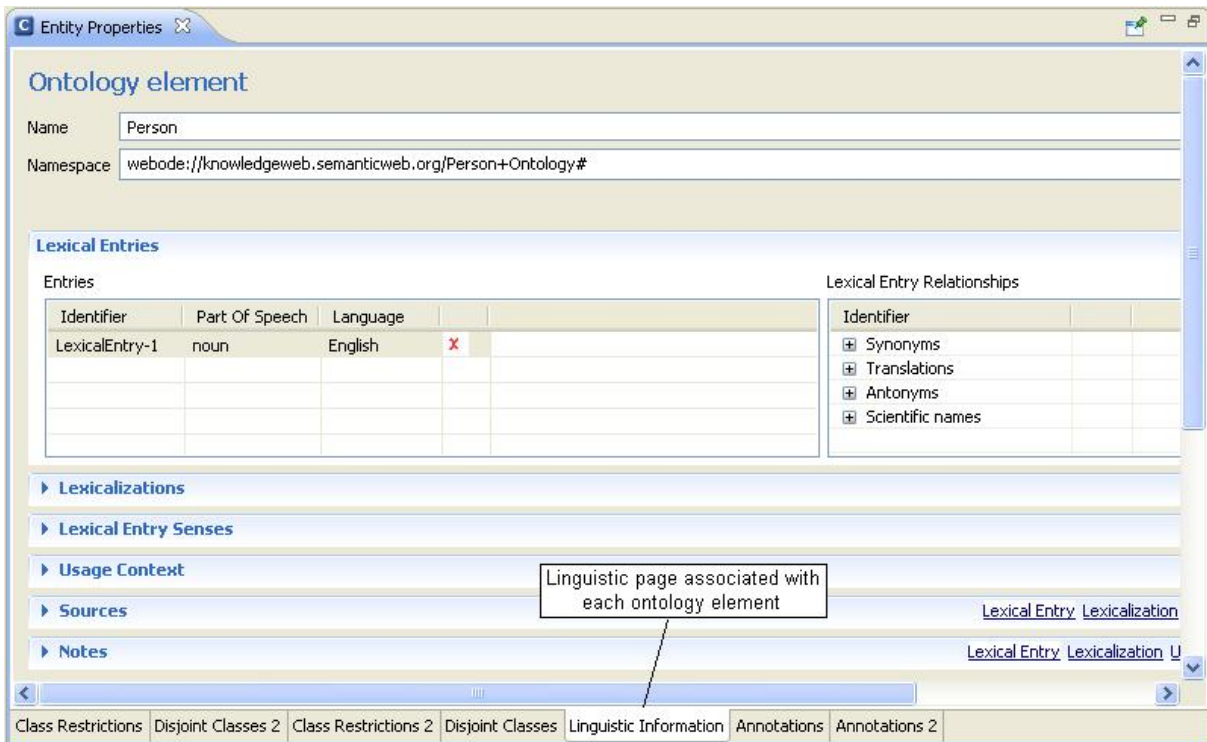


Figure 27: Linguistic Information Page

Localizing an Ontology Label

By right-clicking on an OWL ontology element, the user can access the menu "Translate" added by the LabelTranslator plug-in.

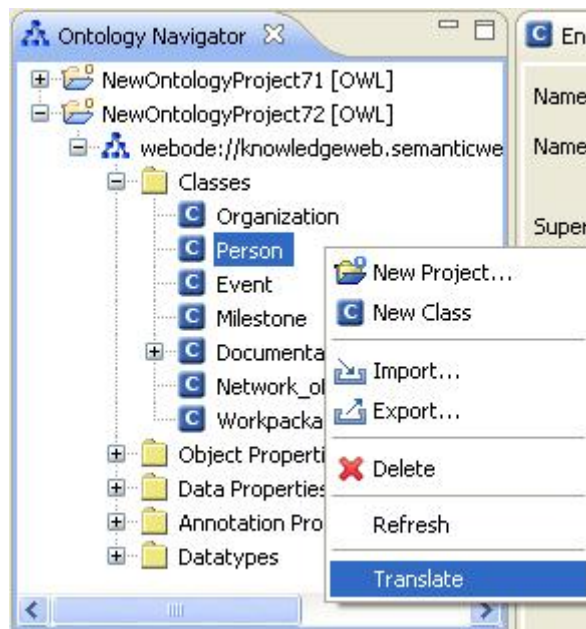


Figure 28: Label Translation Context Menu

Then, a user interface shows the ranked translations generated by LabelTranslator plug-in. The system proposes the most relevant translation to be selected, but the user can change this default selection.



Figure 29: Label Translation

Finally in the Linguistic Information page, the plug-in fills in runtime the fields that show linguistic information, according to the translations selected by the user. These fields represent the link between the conceptual knowledge and the discovered linguistic information.

Updating the Linguistic Information Repository (LIR)

The Linguistic Information page shows five sections that correspond to the lexical entries associated to the selected ontology element (“Person” in our example) and the corresponding linguistic information related to each lexical entry: lexicalizations, senses, usage contexts and sources. For instance, in this case the concept “Person” has two lexical entries, one in English and one in Spanish. The Lexical Entry’s section represents the master (in a master/detail model) from which it is possible to deploy the related information. The information shown in the different lexical entry sections depends on the selected lexical entry (“LexicalEntry-1” in our case).

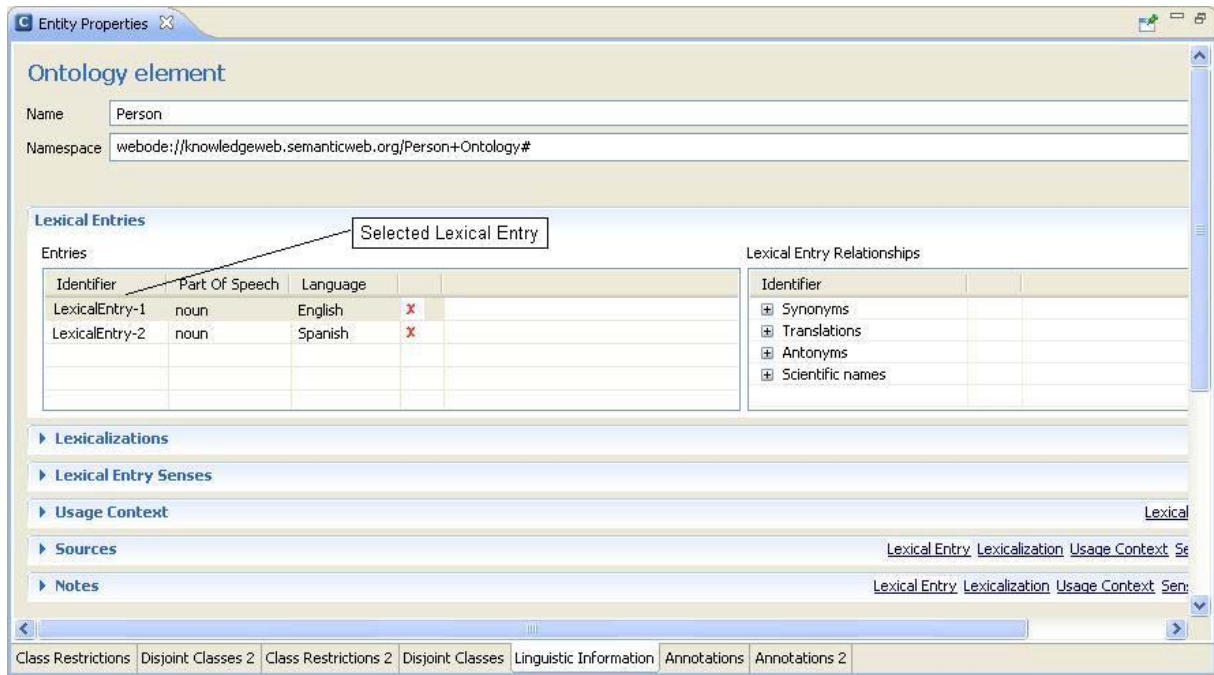


Figure 30: Updating the Linguistic Information Repository

Of course, every time that the user chooses a new entry, the interface automatically displays the information correlated in the different sections. For example, next figure shows the lexicalizations associated with the selected lexical entry shown in the previous figure.

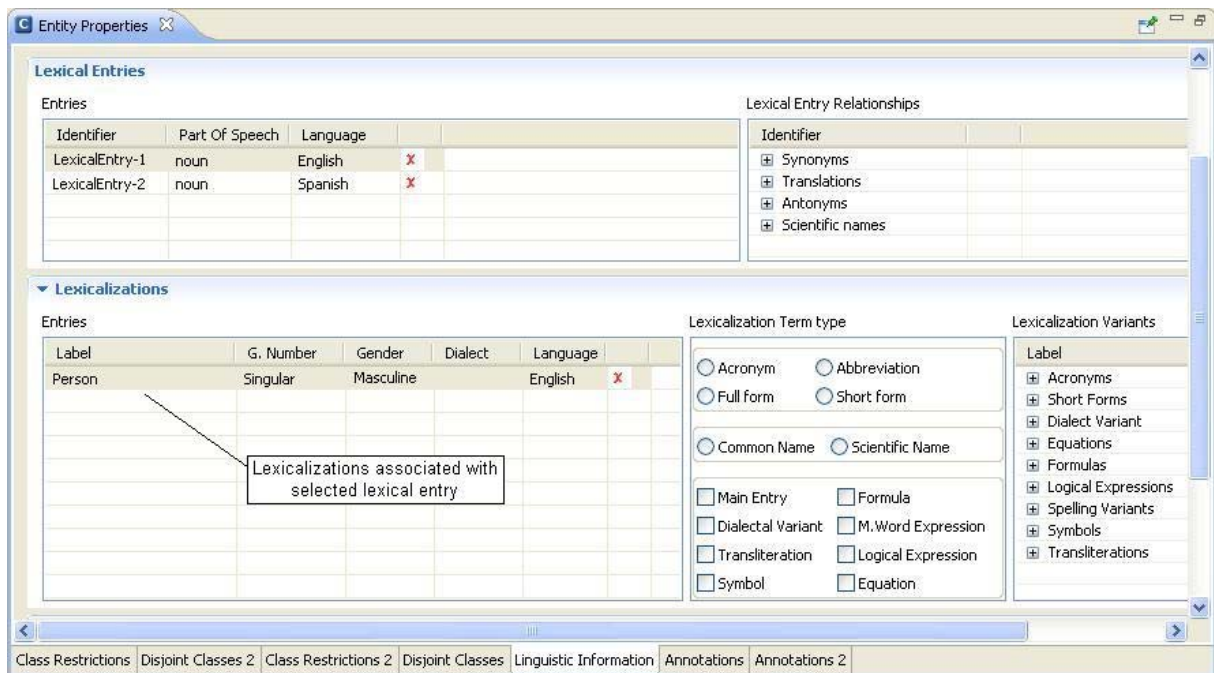


Figure 31: Lexicalizations associated with a lexical entry

Synchronizing Ontology and Linguistic Model

LabelTranslator provides a model where sets of ontology terms and their associated linguistic information (in different languages) are separately stored. In the NeOn Toolkit, an advanced change tracking is able to capture changes even when ontological terms have changed their position within the ontology model.

By adopting this feature, our system can accurately identify the minimal set of changes needed to adjust the structure of the linguistic model. This task is transparent to the user. Thus, for instance, when the user adds an element in the ontology, then, automatically, LabelTranslator performs the corresponding action in the linguistic model.

3.8.3. Integration into the NeOn Toolkit

In order to compare and contrast the two versions of the LabelTranslator plug-in, we use the three layered architecture used in NeOn. In the Figure below, the high level architecture of both versions is shown. The left part of the figure shows the components of the first version of LabelTranslator, while the representation on the right corresponds to the enhanced architecture of its second version.

- The first layer encapsulates the graphical user interface that enabled user interaction. While the first version uses the current NeOn Toolkit for storing the multilingual information related to a specific ontology label, the second version of LabelTranslator adds support to the new linguistic information model (LIR). Moreover, we have implemented a LinguisticView (“LinguisticReposService” in figure), which contains a set of fields for editing the linguistic information associated now to each ontology element.
- In the second layer, i.e. the one that represents the business functionality of the system, the second version of the prototype adds a new algorithm used to support semi-automatic translations of ontology elements and applying the characteristics of the LIR.
- Finally, in the third layer, which is used to store the multilingual information, the new prototype adds a new repository (LinguisticReposService) to store the linguistic information associated to each ontology element, unlike the previous version. Consequently, the linguistic information is stored in two places at the same time.

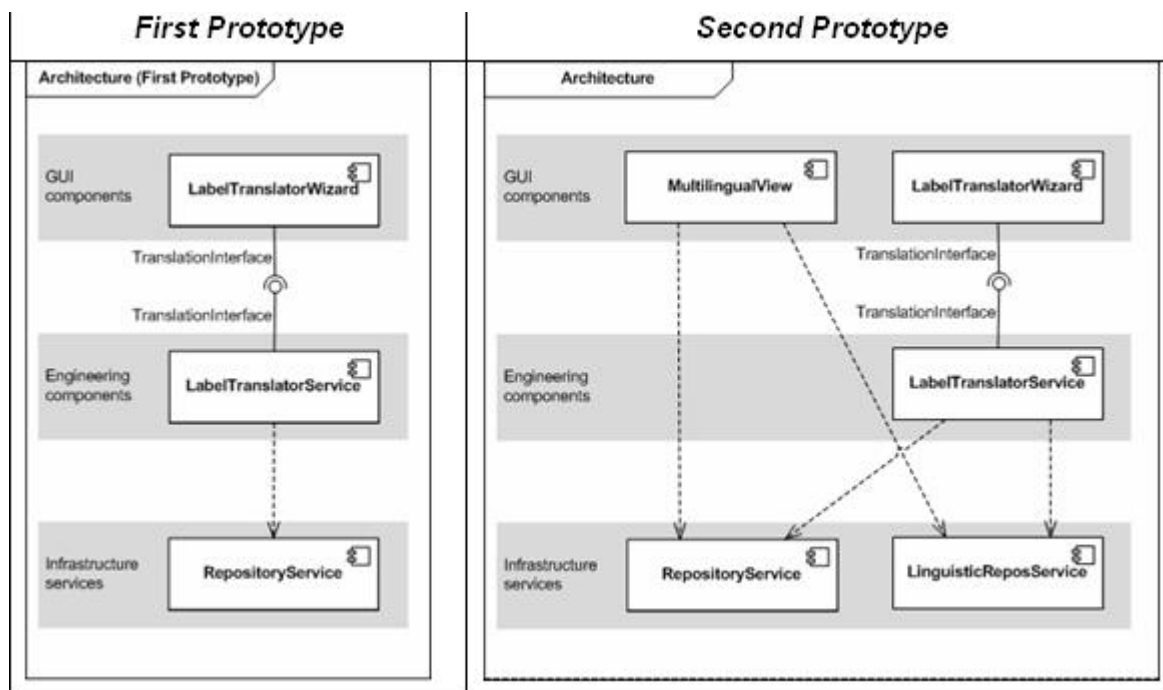


Figure 32: Architecture of LabelTranslator

The following extension points are used within the plug-in:

- Eclipse extension points:
 - org.eclipse.ui.popupMenus

- org.eclipse.ui.startup
- org.eclipse.ui.importWizards
- NeOn toolkit extension points:
 - org.neontoolkit.gui.entityProperties

Additionally, LabelTranslator has dependencies with other plugins:

- com.ontoprise.ontostudio.datamodel
- com.ontoprise.ontostudio.owl.gui
- com.ontoprise.ontostudio.owl.model
- com.ontoprise.ontstudio.gui

3.8.4. Intended Usage in the Case Studies

LabelTranslator will be used in WP7

- Motivation of using LabelTranslator

Within WP7, ontology engineers and ontology editors manage the multilingual aspect of ontologies. In particular, engineers specify which elements of the ontology should be multilingual (classes, object properties, data properties or the entire ontology). LabelTranslator provides ontology engineers and editors a tool to enrich an existing ontology with terms in different languages.

- Benefits/Advantages of using LabelTranslator

LabelTranslator supports the translation of ontological labels using relevant information obtained from different lexical resources. Depending on the needs of ontology engineers and ontology editors, LabelTranslator can be used in two different ways:

- As a computer-assisted translation (CAT) tool that helps users understand ontology labels described in foreign languages.
- As a tool for automatic multilingual enrichment of the different components of an ontology.

In both modes of operation, LabelTranslator proposes the most relevant translation to be selected. However, the user can change this default selection. This potential user intervention is justified because the context may not be enough to translate an ontology label according to the user's interpretation.

- Data sets

LabelTranslator works with lexical databases, bilingual dictionaries and terminologies (as linguistic resources) to translate an ontology label. Also, the tool uses some views of the NeOn Toolkit as repository of the multilingual information.

3.9. LeDA

3.9.1. Functional Description

LeDA is a tool for the automatic generation of disjointness axioms based on machine learning classification. The classifier, that determines disjointness for any given pair of classes, is trained

based on a gold standard of manually created disjointness axioms. Each axiom of the gold standard is represented by a pair of classes associated with a label - disjoint or not disjoint - and a vector of feature values. As in our earlier experiments, we used a variety of lexical and logical features, which we believe to provide a solid basis for learning disjointness. These features are used to build an overall classification model on whose basis the classifier can predict disjointness for previously unseen pairs of classes.

For performance and usability reasons the LeDA plugin works with the following reduced set of features. The full feature set as described in D3.8.1 is so far only available in the standalone version of LeDA (<http://ontoware.org/projects/leda/>).

```
leda.features.LabelSimilarity_JaroWinkler
leda.features.LabelSimilarity_Levenshtein
leda.features.LabelSimilarity_QGrams
leda.features.Ontology_Subsumption
leda.features.Ontology_Similarity
leda.features.Ontology_ObjectProperties
leda.features.TaxonomicOverlap_Subclasses
leda.features.TaxonomicOverlap_Instances
```

3.9.2. User Documentation

LeDA View

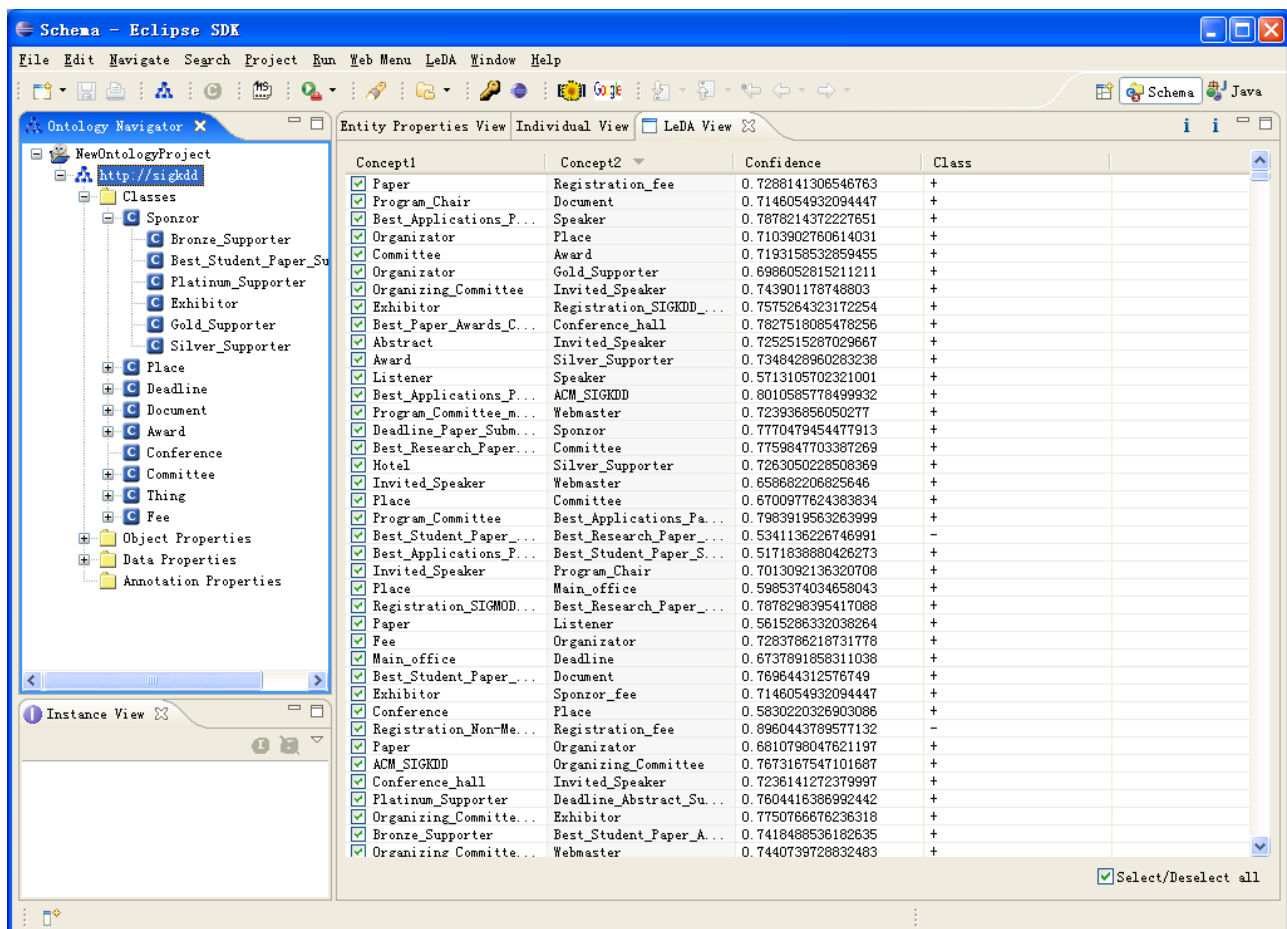


Figure 33: LeDA User Interface

This screenshots shows the main view of LeDA. A training or classification process can be triggered by means of a context menu, that is accessible by clicking on an OWL ontology in the navigator view on the left ("Learn Disjointness"), or by selecting "Run LeDA" from the main "LeDA" menu.

Preferences

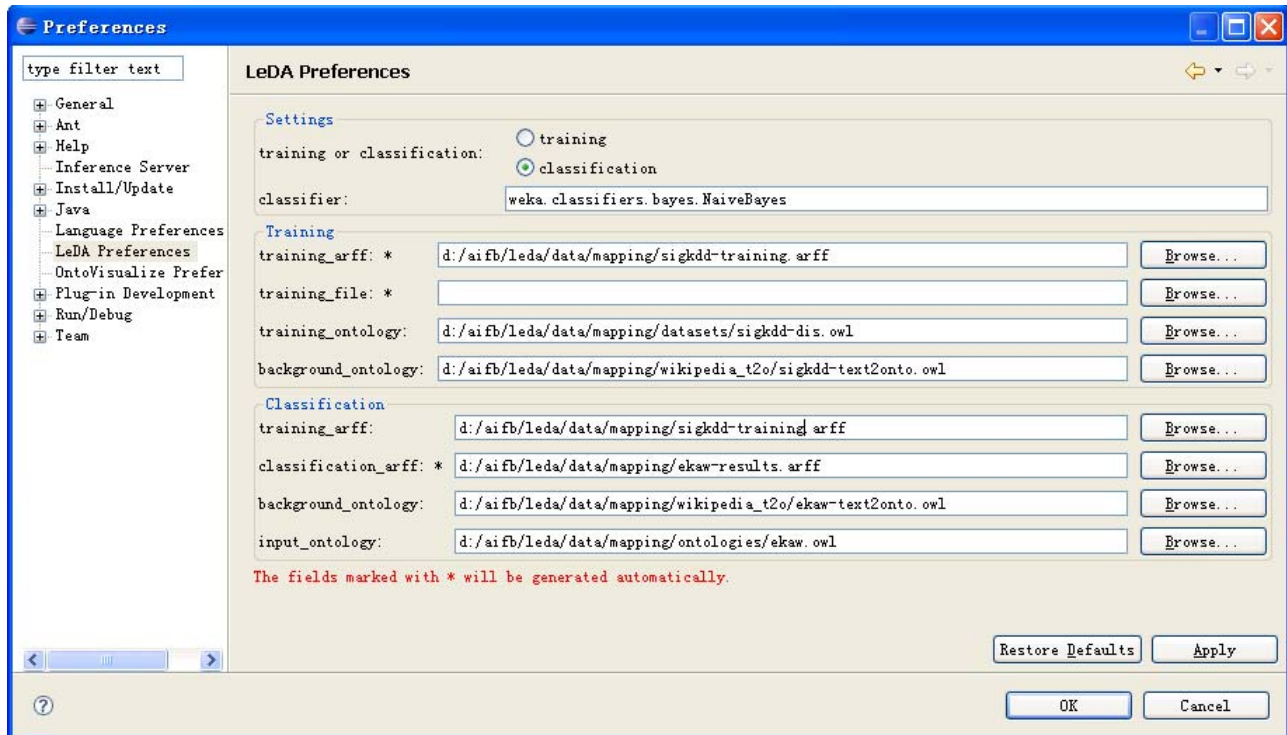


Figure 34: LeDA Preferences

The preference page is accessible from the main menu of Eclipse ("Window" -> "Preferences..." -> "LeDA Preferences"). It allows for setting a variety of parameters for both training and classification:

- **Training or classification**
- **Classifier:** The Weka (<http://www.cs.waikato.ac.nz/~ml/weka/>) classifier to be used by LeDA.
- **Training**
 - **Training ARFF:** This ARFF file will be generated automatically at the end of the training phase. At this point, it will contain all the necessary information for creating a classification model.
 - **Training file:** This is an optional, but recommended parameter. If no training file is specified, positive and negative examples will need to be generated from the training ontology - a process that is very time-consuming.
 - **Training ontology:** This has to be an ontology, which contains a complete set of manually created disjointness axioms.
 - **Background ontology:** The background ontology serves as additional background knowledge. This parameter is optional and will only be used in case appropriate features have been selected for the classification model.
- **Classification**

- **Training ARFF:** The training ARFF file must have been generated in a preceding training phase.
- **Classification ARFF:** This file will be generated automatically as soon as the classification phase is finished and mainly serves debugging purposes.
- **Background ontology:** See training phase.
- **Input ontology:** This parameter specifies the ontology, which is to be enriched by disjointness axioms.

More information with regards to this plugin can be found in NeOn D3.8.1.

3.9.3. Integration into the NeOn Toolkit

The tightly coupled GUI plugin serves as a graphical frontend for the original version of LeDA. It has been implemented as a single view with associated main menu and preference page.

- relies or depends on the following plugins
 - none
- Eclipse Extension Points
 - [org.eclipse.ui.perspectives](#)
 - [org.eclipse.ui.views](#)
 - [org.eclipse.ui.actionSets](#)
 - [org.eclipse.core.runtime.preferences](#)
 - [org.eclipse.ui.preferencePages](#)

3.9.4. Intended Usage in the Case Studies

The LeDA plugin is to be used in WP7 for ontology learning experiments.

3.10. Modules Plugin

3.10.1. Functional Description

The Modules Plugin provides the functionality to extract modules and the operators to manipulate and combine modules. Specifically, this Plugin offers the following functionalities:

- Modify the Export Interface elements for a module M and the Import Interface elements for those modules imported by M.
- Compute the union between two modules by considering or ignoring namespace
- Calculate the intersection between two modules by considering or ignoring namespace
- Compute the difference between two modules by considering or ignoring namespace
- Provide alignment between two modules

3.10.2. User Documentation

In this section, we first introduce the view of **Module Composition** and then the view of **Module Interface**.

Module Composition

1. Open the view of module composition: In the menu of "Windows", please choose Show view / Other / Module Composition.

2. Manipulate two modules: First of all, choose the two modules in any projects by using the corresponding "Refresh" button. Then, one could choose one of the operations (e.g. union, intersection or difference) and do a simple configuration to decide whether considering the namespace or not when executing the operation. After this, press "Execute Operation" to execute the chosen operation and the results will be shown in the text area. The following figure gives an example of the operations.

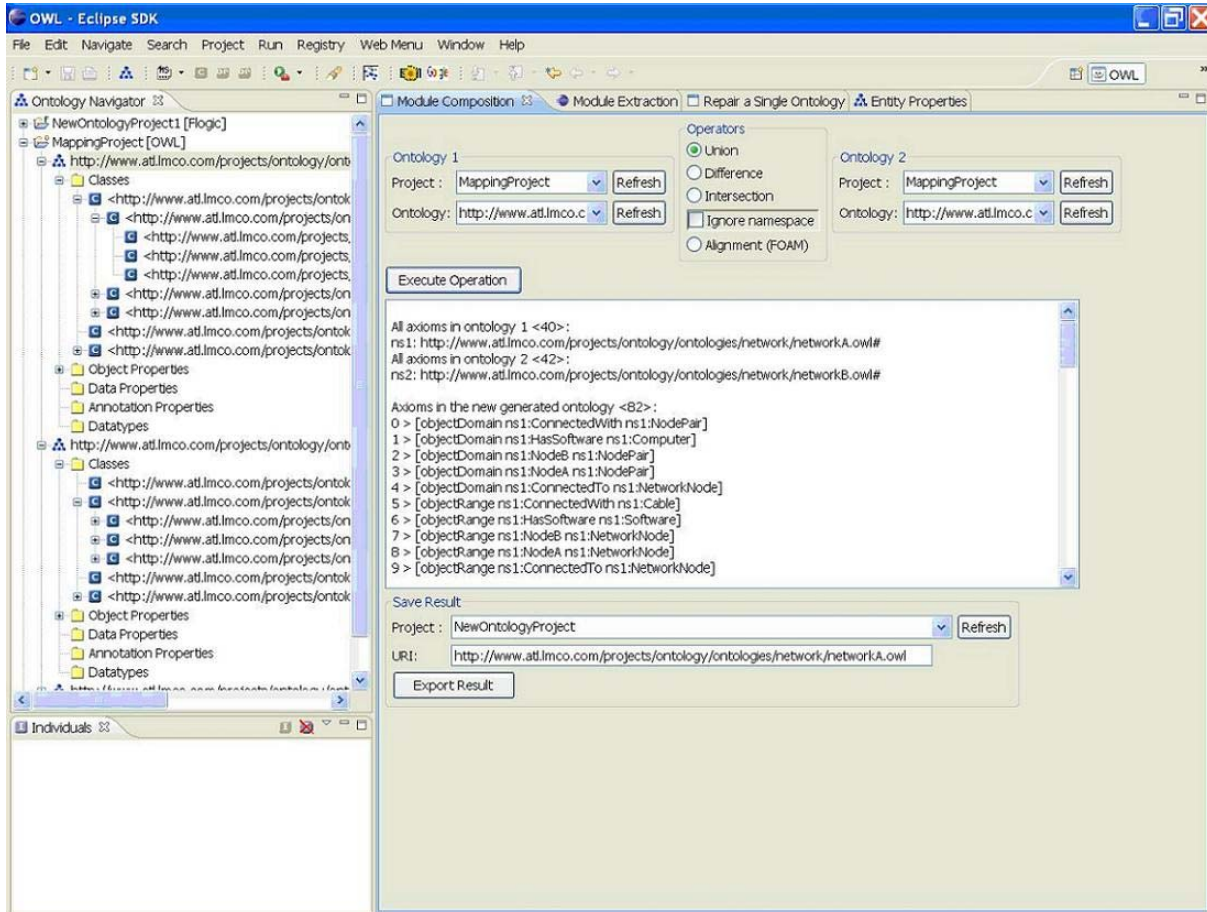


Figure 35: The result of union operator

3. Align two modules: Given two modules, the alignment between two modules is computed using FOAM, an ontology alignment tool. See the following figure as an example.

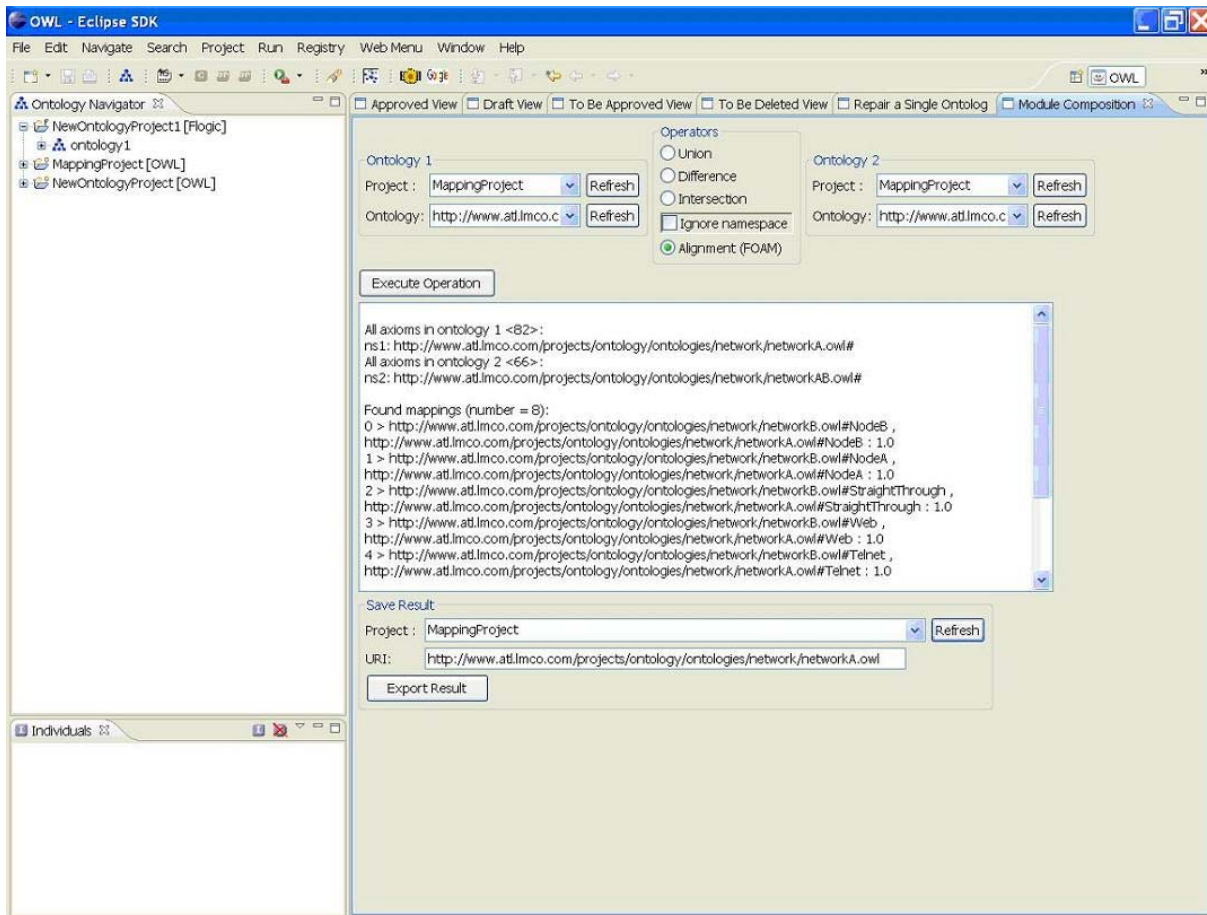


Figure 36: The alignment between two modules

4. Save results: This corresponds to the "Save Results" part in the view. For the union, intersection and difference operators, one can save the resulting module into an existing project. To do so, please "Refresh" the project list in "Save Results" part and then click "Export Result". The resulting module will be exported to the chosen project. While for alignment operation, the found alignment can be saved to a local file by simply clicking the button of "Export Result" and then choosing a directory.

Module Interface

1. Open the view of Module Interface: In the menu of "Windows", please choose Show view / Entity Properties. Then you can find the view of Module Interface as a tab in the Entity Properties page. This view can be seen when an OWL ontology in the Ontology Navigator is chosen.

2. Modify Export Interface elements: When the view of Module Interface is visible, you can see the URI and physical location about the ontology you have chosen in the Ontology Navigator on the top of the view. Besides, all the elements including classes and properties are shown automatically in the list of **All Elements in the Ontology**. If this ontology already has defined some Export Interface elements, they will be shown as well. Then you can modify the Export Interface elements by choosing more elements in the ontology or remove some elements in the Export Interface elements.

3. Modify Import Interface elements: The list of **Imported Modules** will be initialized when the view is first visible. Then you need to manually refresh the list by clicking the **Refresh** button right below the list if the imported modules have been changed in the tab of **Imports and Namespaces**

in the Entity Properties page. Then you could choose one of the imported modules and modify the Import Interface elements in this module which is similar to the functionality to modify the Export Interface elements above.

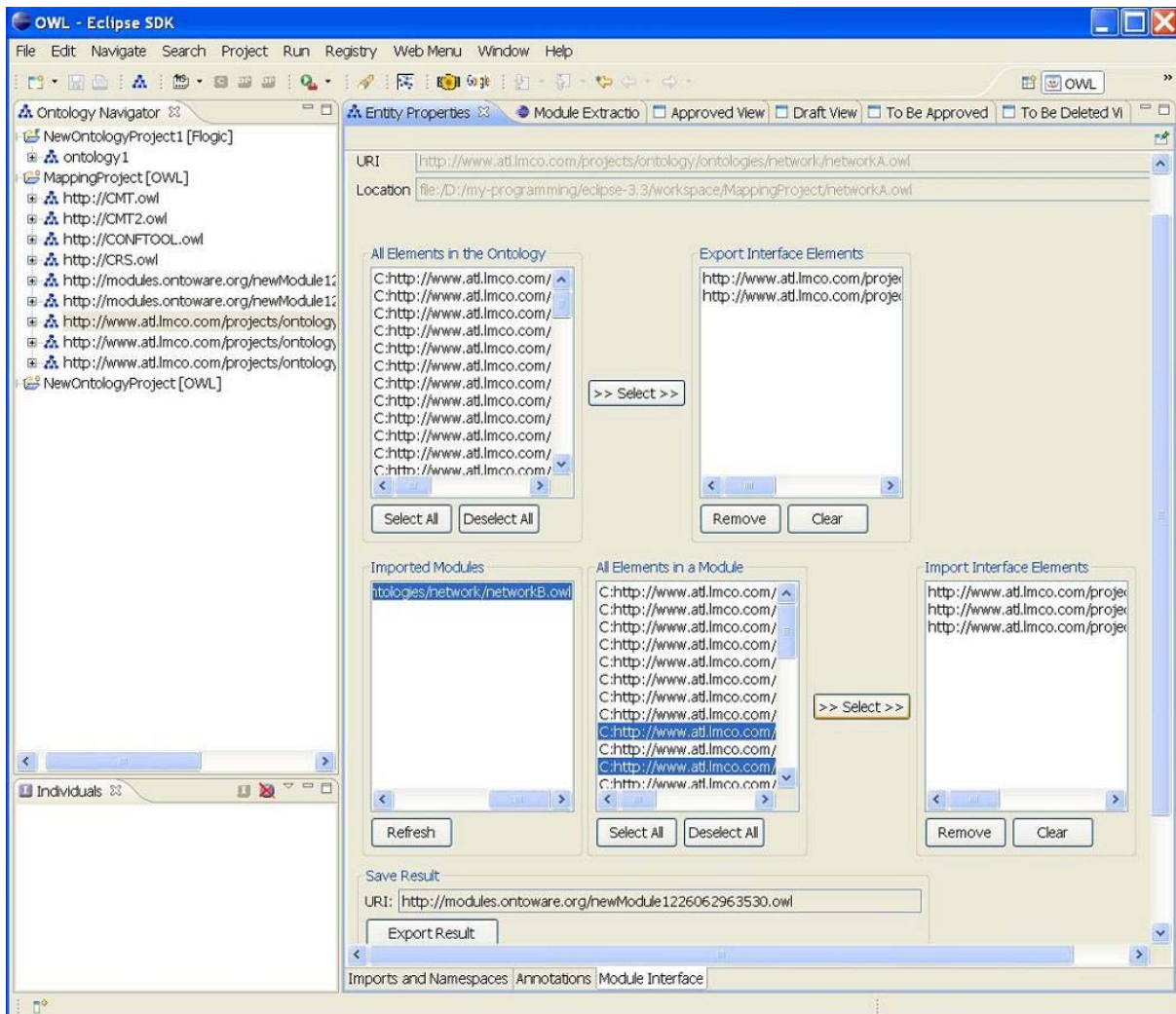


Figure 37: Module Specification

4. Export results: This corresponds to the "Export Results" group in the view. (1) If it is the first time to create a module for the ontology, the URI in this part will be editable and an initial value is given which can be changed by the user. If the button of Export Results is pressed, a new module will be inserted to the ontology project where the ontology is stored with the value in URI area as the default namespace of the module. (2) If the module for the ontology has been created already, the URI area will not be editable with the value of the module's default namespace. If the Export Interface elements or Import Interface elements have been changed, the user could press the button of Export Results to save the changes.

3.10.3. Integration into the NeOn Toolkit

Modules plugin consists of several views in the NeOn Toolkit. One is developed as a tab of "Module Interface" for entity properties. Another one is the view of "Module Composition" which can be operated on the ontologies which have been loaded or created in Ontology Navigator in NeOn Toolkit.

- Eclipse extension points

- org.eclipse.ui
- org.eclipse.core.runtime
- org.eclipse.core.resources

- NeOn Toolkit extension points
 - com.ontoprise.ontostudio.datamodel
 - com.ontoprise.ontostudio.owl.gui
 - com.ontoprise.ontostudio.owl.model
 - datamodel
 - datamodelBase
 - util

3.10.4. Intended Usage in the Case Studies

The Modules plugin can be applied in FAO case study to provide a mean for ontology engineers to develop ontologies in a modular way, specifying the behaviour and role of the ontology modules involved in this development. This has been mentioned as examples in NeOn deliverable 1.1.3.

3.11. ODEMapster

3.11.1. Functional Description

ODEMapster plugin allows users to create graphically mappings (expressed in R2O language), execute and query the mappings. This plugin just supports a subset of the R2O language, this subset includes the most used R2O primitives. This plugin works with OWL/RDF(S) ontologies and with MySQL databases. Currently only one R2O mapping per ontology can be created.

ODEMapster is the processor in charge of carrying out the exploitation of the mappings defined using R2O, performing both massive and query driven data upgrade.

- Query driven upgrade (on-the-fly query translation) each real object stays in the data sources themselves. At the run-time, the mapping converts the ontology-based query into a data source-based query to provide a variety of activities for retrieving instances.
- Massive upgrade batch process that generates all possible Semantic Web individuals from the data repository. Each real object in the data sources is migrated into the ontologies as a formal instance.

R₂O Language

Mappings are expressed in R₂O language and they are stored in the file system, within the NeOn Toolkit Workspace. R₂O is an extensible and declarative language to describe mappings between relational database (DB) schemas and ontologies. For a complete reference of the R₂O language we refer to the online documentation.

Queries

After you have created several R₂O mappings you are ready to query the ontology (using the R₂O mapping) to get the RDF data from the SQL database. To this end, this plugin has a GUI for querying the ontology.

3.11.2. User Documentation

How to install it

- Download the plugin release from http://droz.dia.fi.upm.es/plugin/org.neontoolkit.upm.odemapster_1.0.0.jar
- Copy the plugin JAR into the plugins directory inside the file system location where NeOn Toolkit is installed.

How to use it

- **Switch to R2OMapping perspective**

Choose the R₂O Mapping Perspective and click on the ok button (Figure 2).

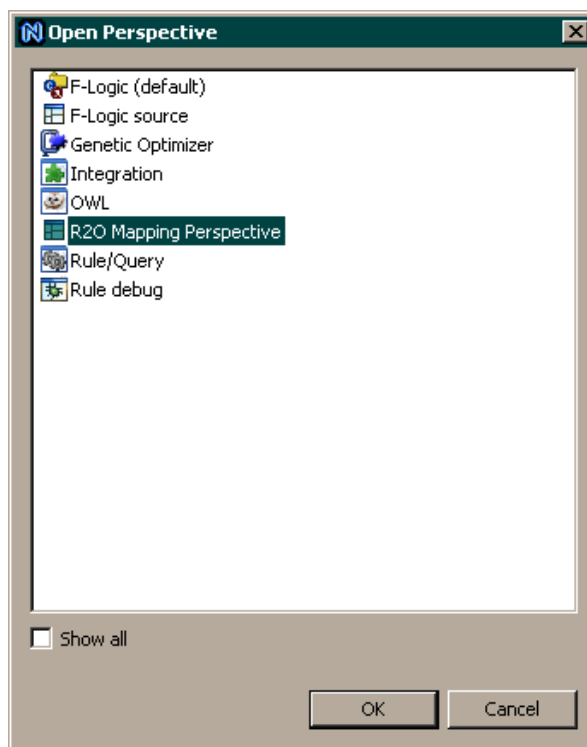


Figure 38: Open R2O Mapping Perspective

This R2OMapping perspective has the following areas:

1. *View mappings area*. In this area the user filters the type of mappings she/he wants to visualize.
2. *Attribute operations area*. In this area the user selects the type of attribute operations she/he wants to work with. These operations have as target an attribute. These operations cannot be combined with relation operations.
3. *Relation operations area*. In this area the user selects the type of relation operations she/he wants to work with. These operations have as target a relation. These operations cannot be combined with attribute operations.

4. *Information area.* In this area the information of the selected operations (attribute or relations) is displayed.
5. *Database area.* In this area the database schema information is shown. It contains the Expand all, Collapse all, and filter buttons.
6. *Ontology area.* In this area the ontology schema information is shown. It contains the Expand all, Collapse all, and filter buttons.
7. *Mapping area.* In this area the mappings are displayed.

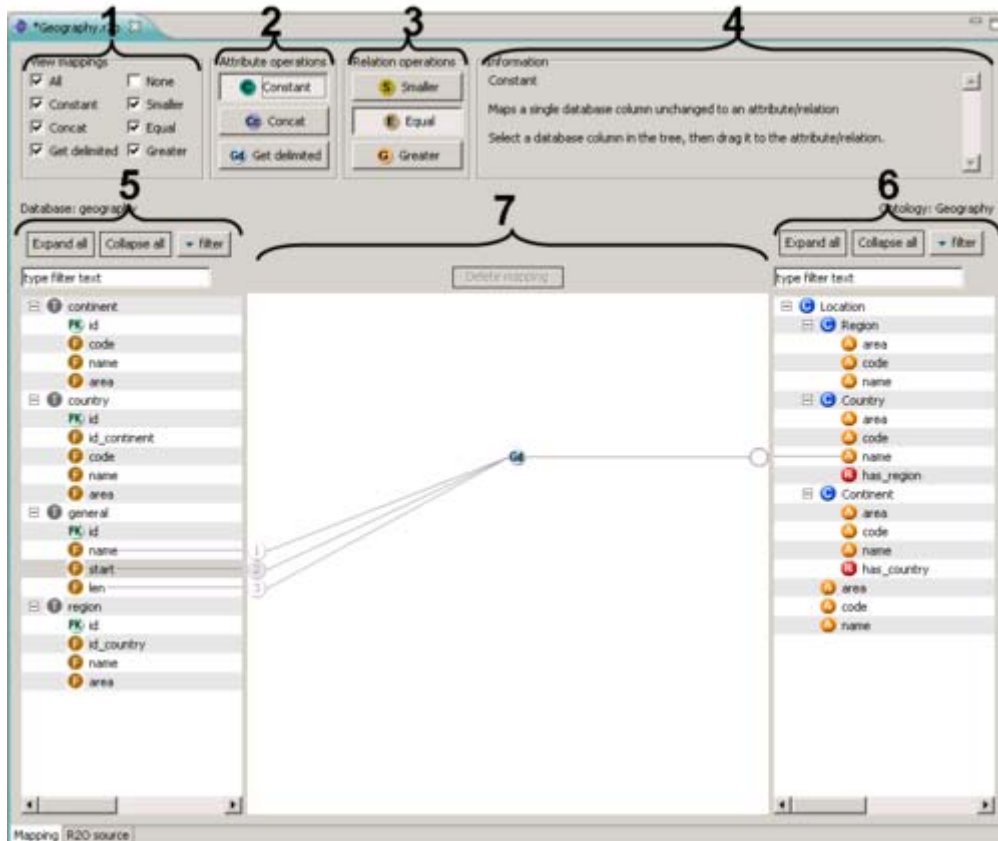


Figure 39: R2O Mapping perspective sections

Ontology schema visualization

In ontology area, the ontology schema information is shown. At the first level all the root classes are displayed. In the next level all the subClasses, of a given root class are shown; and in the next level for each subClass, the attributes and relations are displayed. At the end, after the all subClasses, the properties and relations of the root class are shown. In this particular case, there is only one root Location class, which has the following subClasses: Region, Country, and Continent. The Location class has the following attributes: area, code and name. These attributes are inherited to the Location subClasses (i.e. Region, Country and Continent).

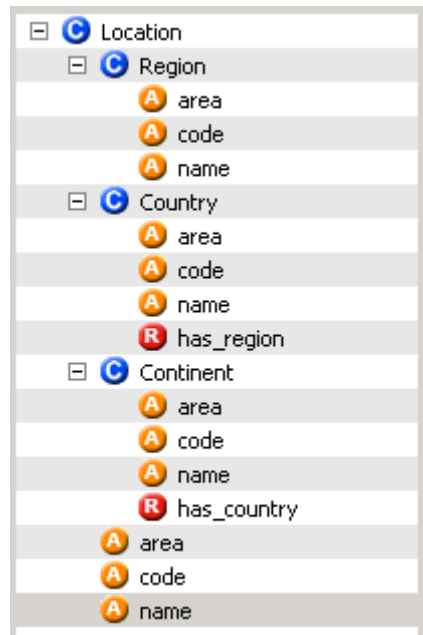


Figure 40: Ontology visualization

Create the R₂O Mapping

Right click on the R₂O Mapping folder and select New R₂O Mapping.

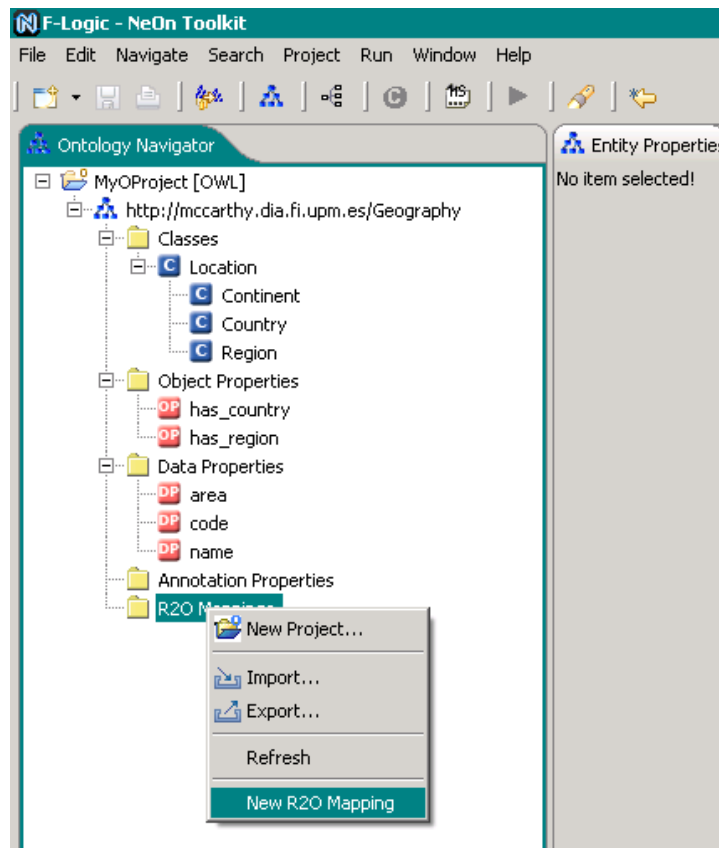


Figure 41: New R₂O Mapping item

The New R₂O mapping wizard opens where you have to select a database. Click on the New database to create a new one.

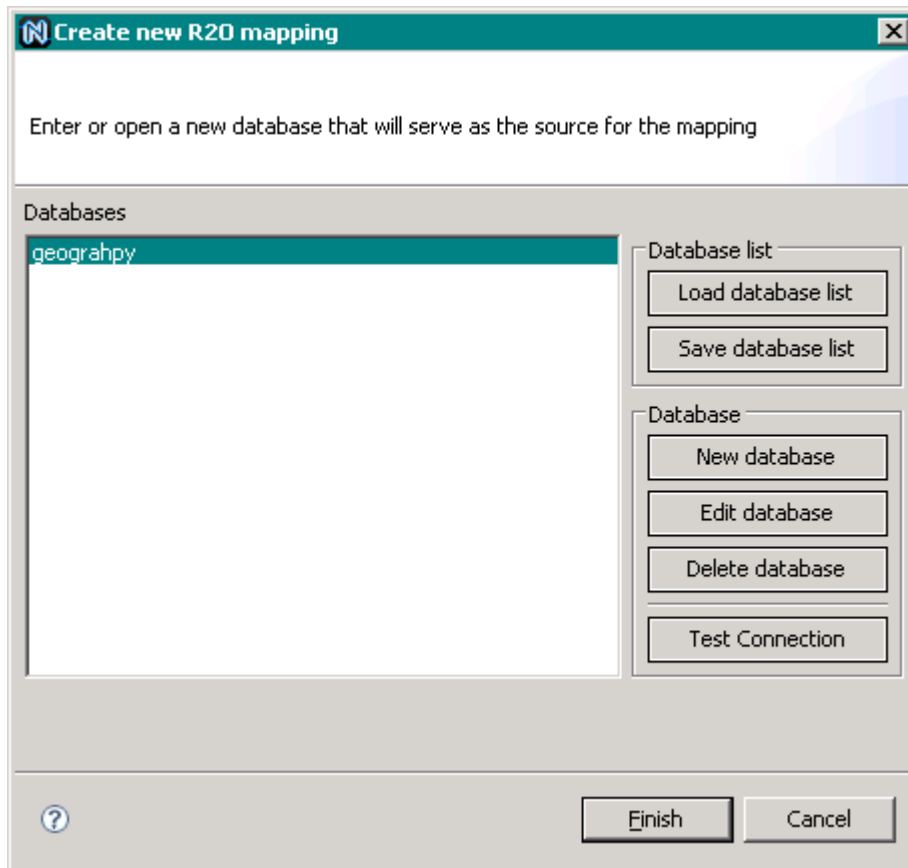


Figure 42: Create a new R2O mapping window

A dialog opens where you can enter the details of the database. That is the database name, the host address, the port, the username and the password. Currently the graphical user interface allows users to work with MySQL databases.

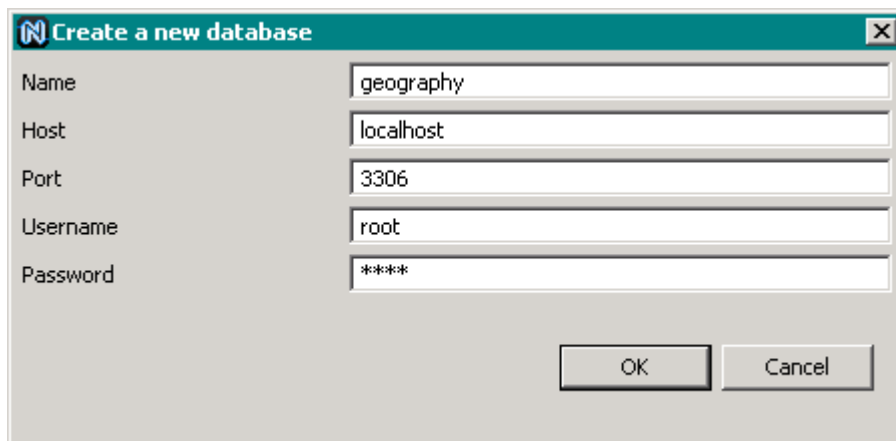


Figure 43: Create a database window

When you have entered all details click on the Ok button.

The newly created database appears in the list and you can now test if it works correctly. Therefore first select the database. Click on test connection to see if everything works fine. If not, you might probably have an error in the connection details, select the database and click edit to correct them. If it still doesn't work, your MySQL server might not have been started correctly. If everything was ok, click on finish to finalize the mapping creation.

The mapping editor will open with you database on the left and the ontology on the right hand side. You can now start to create the R2O mappings.

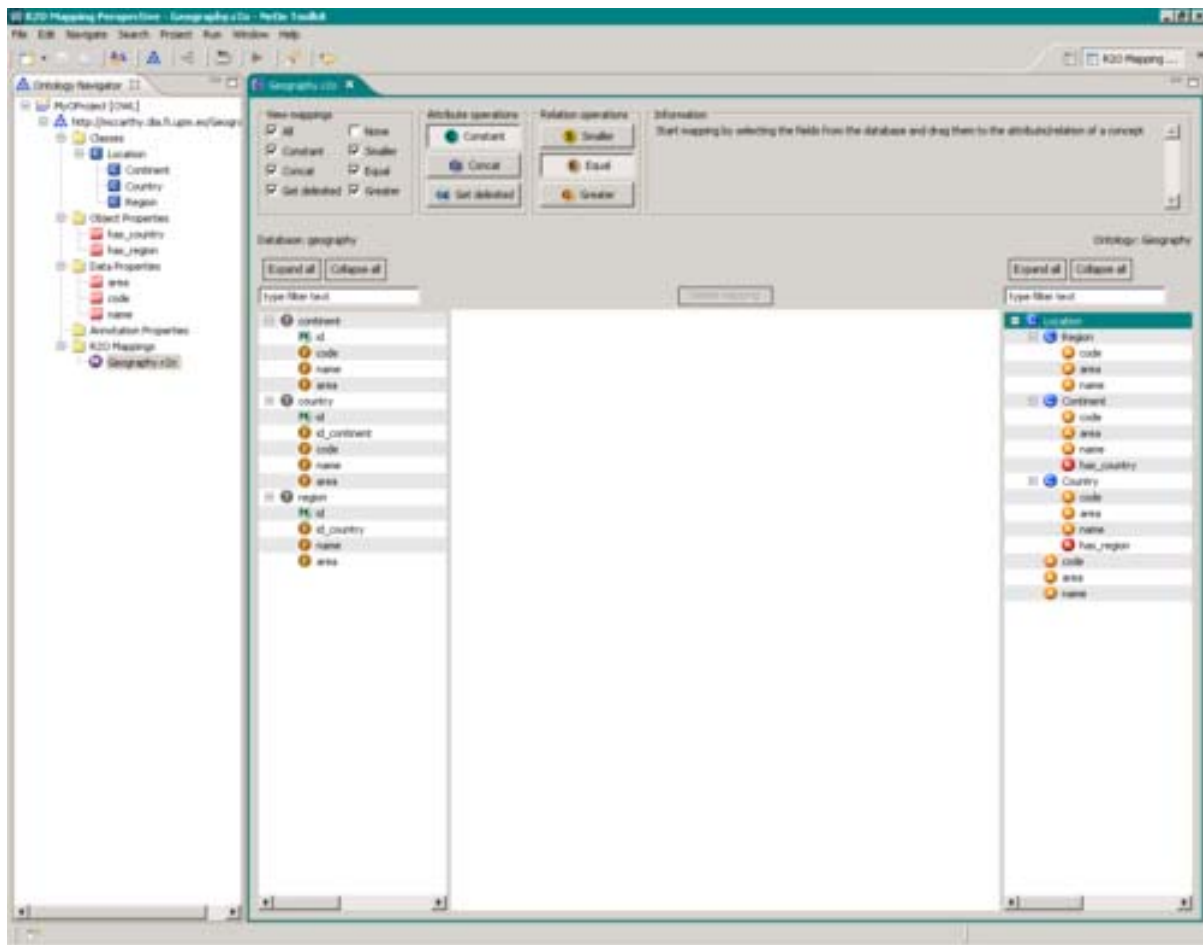


Figure 44: R2O Mapping perspective

Create a constant mapping

You can start now to create a constant mapping. Start by searching for a table column in the database tree on the left side. Each table column has an icon for its type, PK for primary key, FK for foreign key and F for a simple Field. Select a table column by clicking on it. Now start dragging the table column to the ontology tree on the right. Then drop it to an attribute of a concept, described by an orange icon. The red icons describe relations. As you can see the first mapping is established and you see a line from the database column going to the attribute of the concept. Save the mapping by clicking on the save icon of the NeOn Toolkit on the upper left.

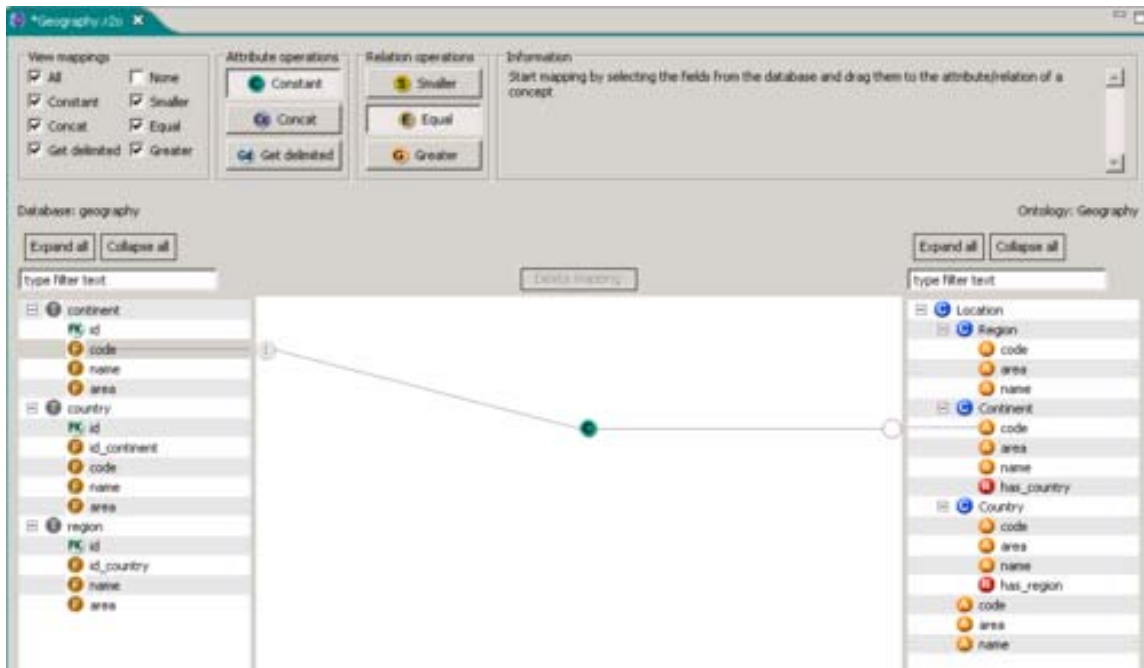


Figure 45: Constant mapping

The R₂O Mapping Editor allows you to see the R₂O code in which the mappings are described. To see it, click on the R₂O source tab in the bottom bar and the R₂O source code is displayed. Click on the Mapping tab to switch back to the Mapping view.

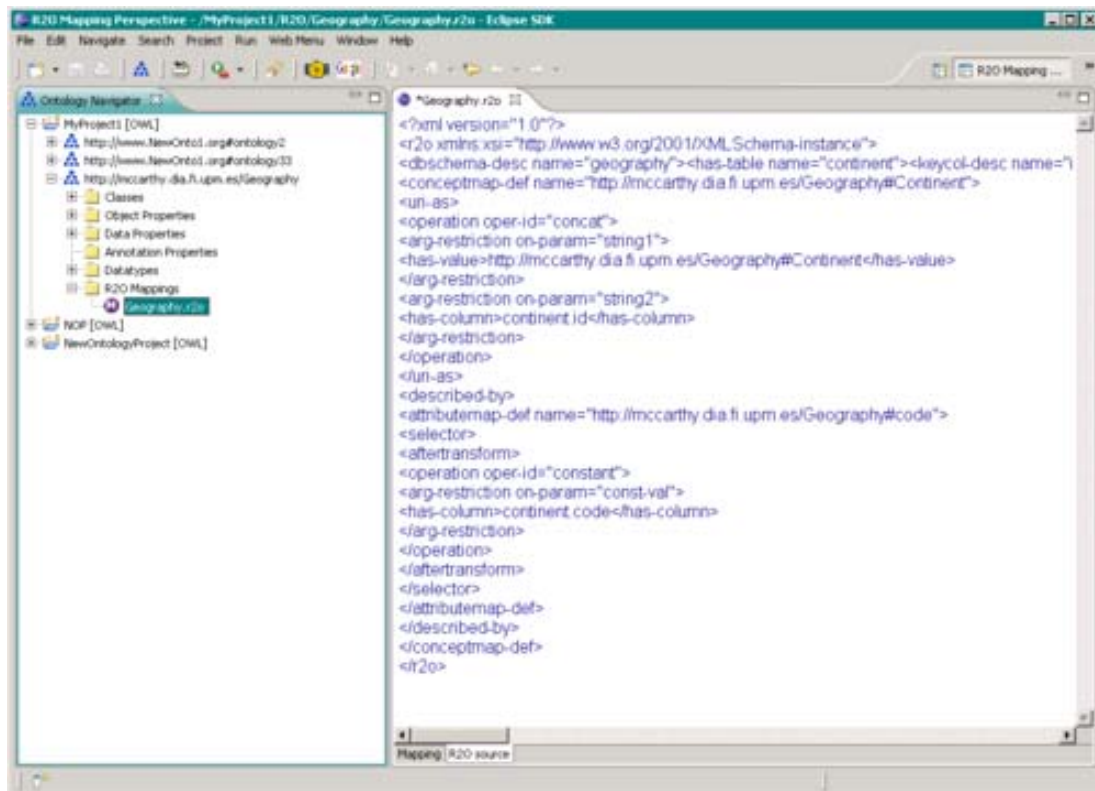


Figure 46: R2O Code

Click on the Mapping tab to switch back to the mapping view.

Create a concat mapping

In the top of the mapping editor you can see a set of buttons called attribute operations. Click on the Concat button. That way you change the mappings from a constant mapping, that is one database column to one attribute to a concatenation of two database columns to one attribute. Now start by selecting the first column in the database table you want to map. Then, hold down the ctrl key and select the second column. The two selected columns are highlighted. Then, release the ctrl key. Next, start dragging the two columns to the ontology tree and drop them at an attribute. The newly established mapping is shown, with another icon for showing that it is a concatenation. On the left side next to the database tree you can see two circles, indicating the order of the concatenation.

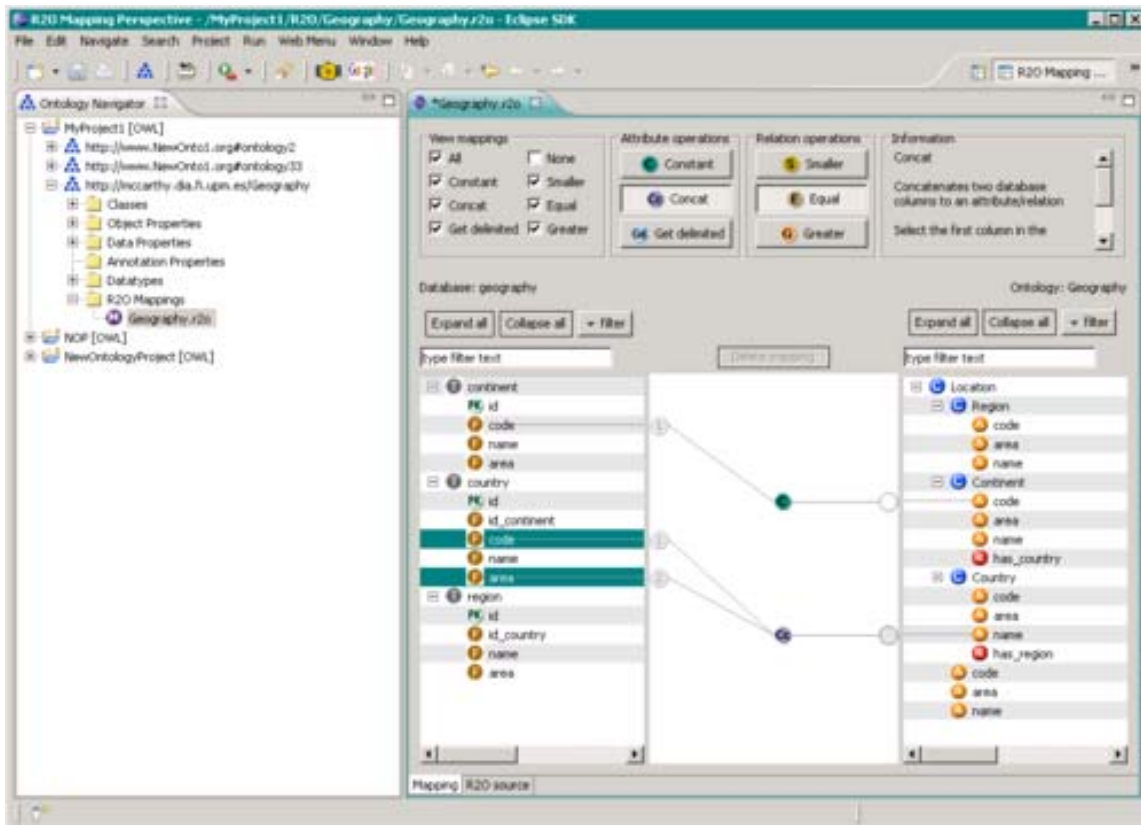


Figure 47: Concat mapping

Create a getDelimited (subString) mapping

In the top of the mapping editor you can see a set of buttons called attribute operations. Click on the Get delimited button. This mapping extracts 'length' characters of 'string' starting from 'start-delim' and set the result to one attribute.

Select the first column (which holds the source 'string') in the database tree, then hold down ctrl and select the second (which holds the 'start-delim') and then the third column (which holds the 'length') in the tree. Now release ctrl and drag the columns to the attribute. The newly established mapping is shown, with another icon for showing that it is a getDelimited operation. On the left side next to the database tree you can see three circles, indicating the order of the concatenation.

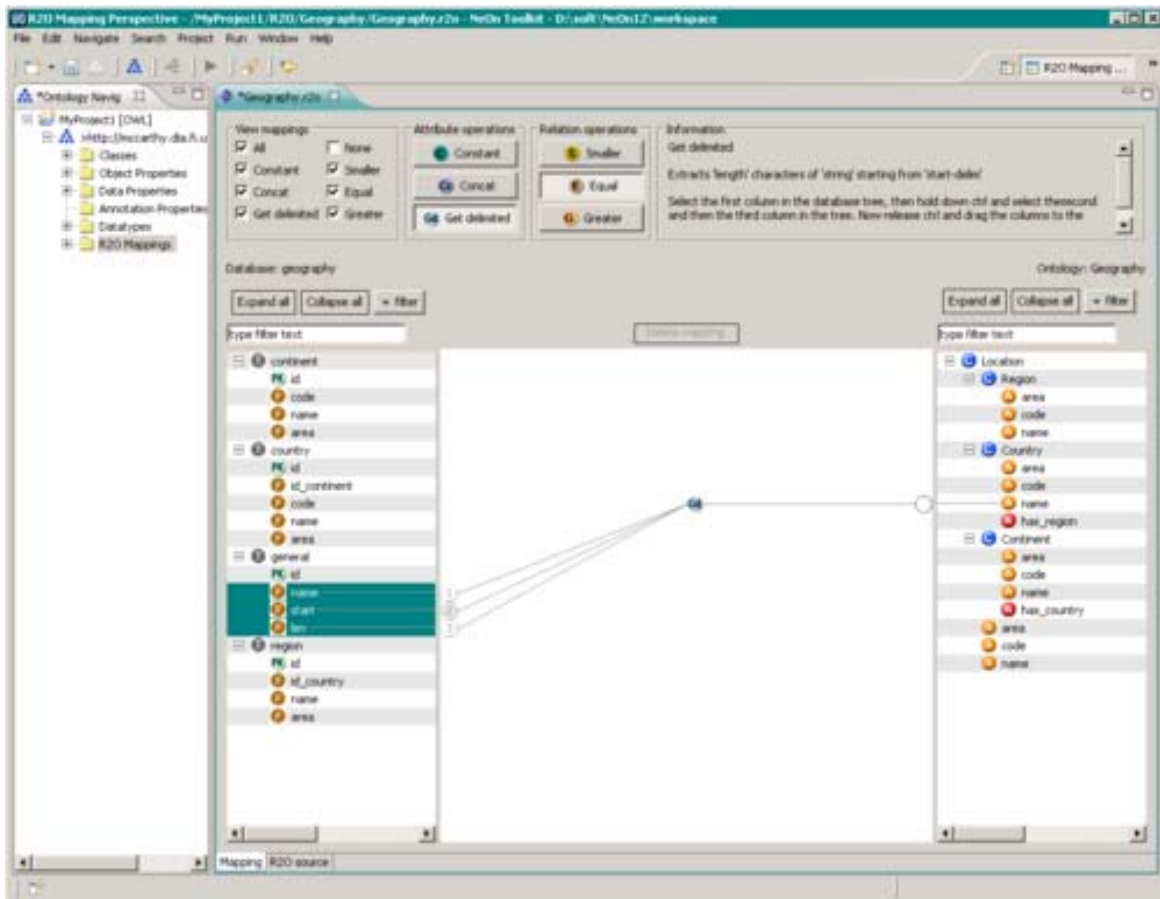


Figure 48: getDelimited mapping

Show and hide mappings

The ODEMAPSTER plug-in allows you to show and hide different types of mapping, to get a better overview when you have a lot of mappings. In the top menu you can see a set of checkboxes under the name View mappings. Click on the checkbox Constant.

The Constant checkbox now is unchecked. As the result you can see that all constant mappings disappear from the view. Check the Constant checkbox again to make it reappear.

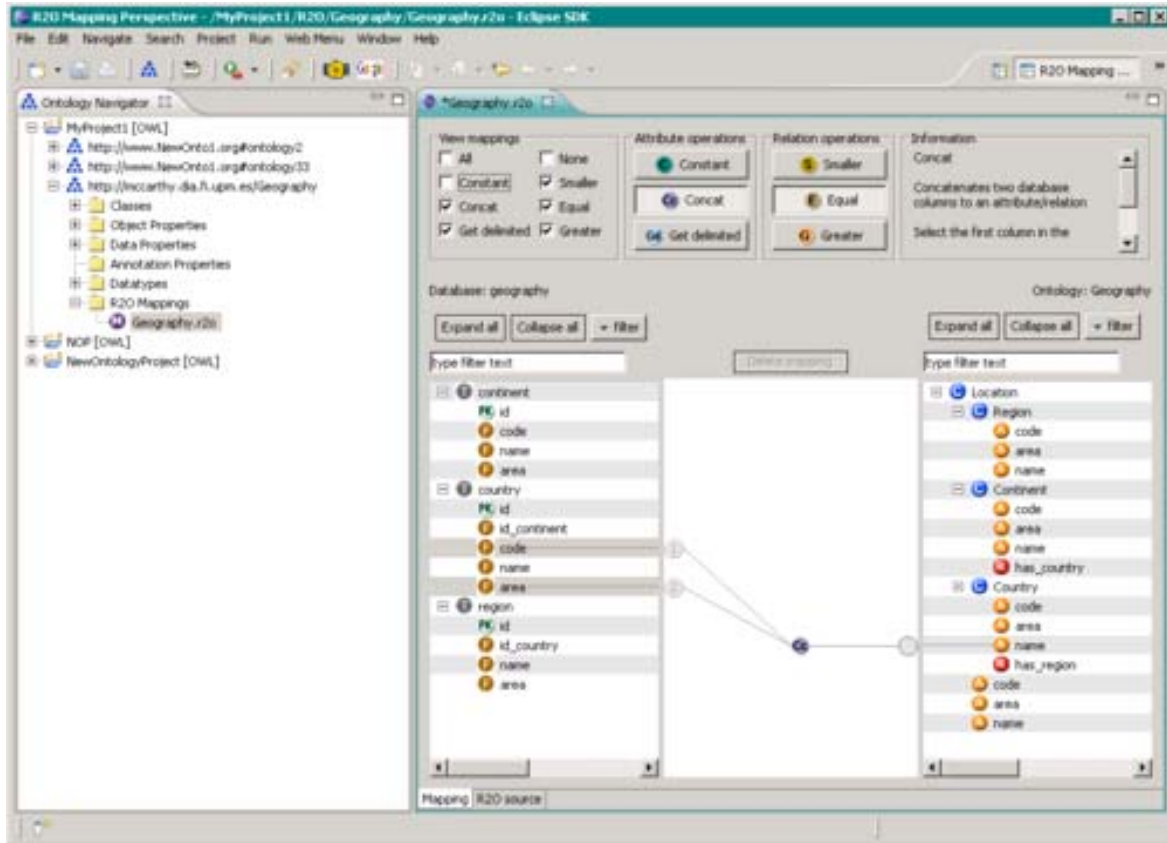


Figure 49: Show and hide mappings

Filter items in the database tree

If you have a large database schema you might want to filter the items to get a better overview of the schema. Click on the filter button above the database tree. A popup menu will appear. Click on Simple columns. As a result you can see that all simple columns are filtered out and only the primary keys and foreign keys are visible. Click on the filter button and deselect the Simple columns entry again to make them reappear again.

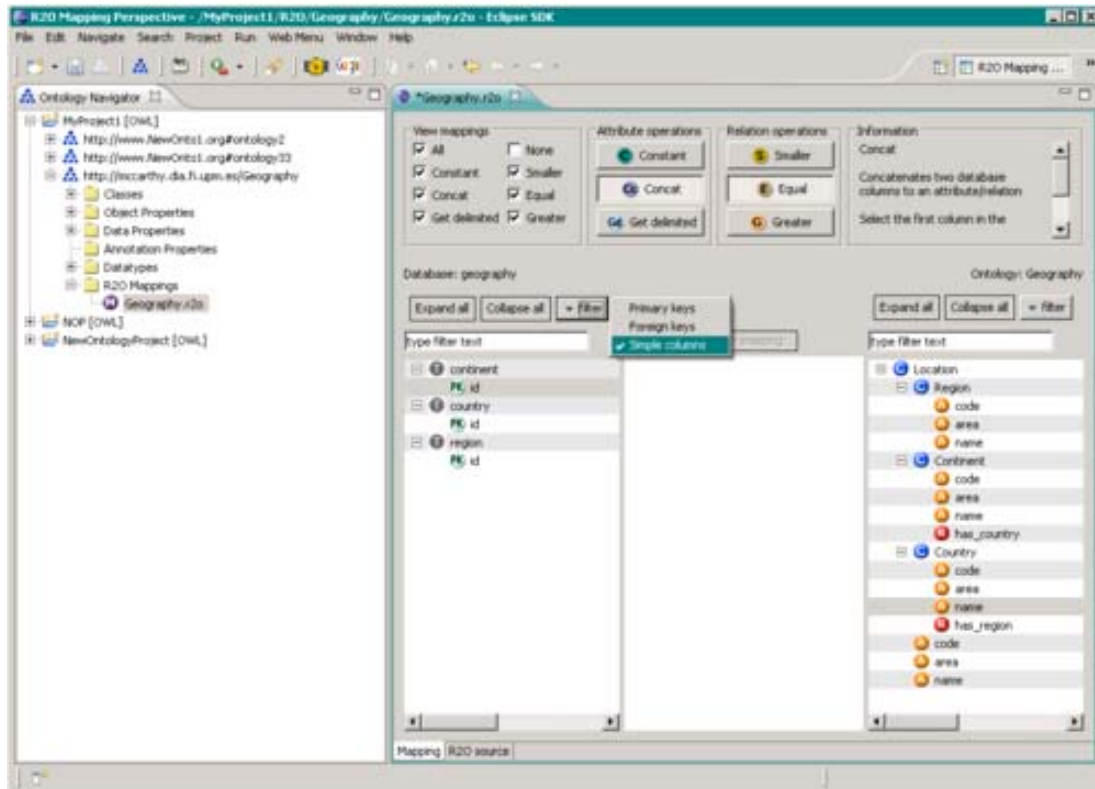


Figure 50: Filter database items

You also have the possibility to filter the database columns by name. Therefore click in the text box above the tree and type some letters. We typed are in the text box, and as the result you can see that only columns that begin with the letters are visible and all others are filtered out. Click on the button right of the text box to clear it and reset the filter to the normal view.

Filter items in the ontology tree

As with the database, you can also filter items out of the ontology tree. Click on the filter button above the ontology tree and click in the appearing popup menu on attributes. You can see that all attributes disappear from the ontology tree.

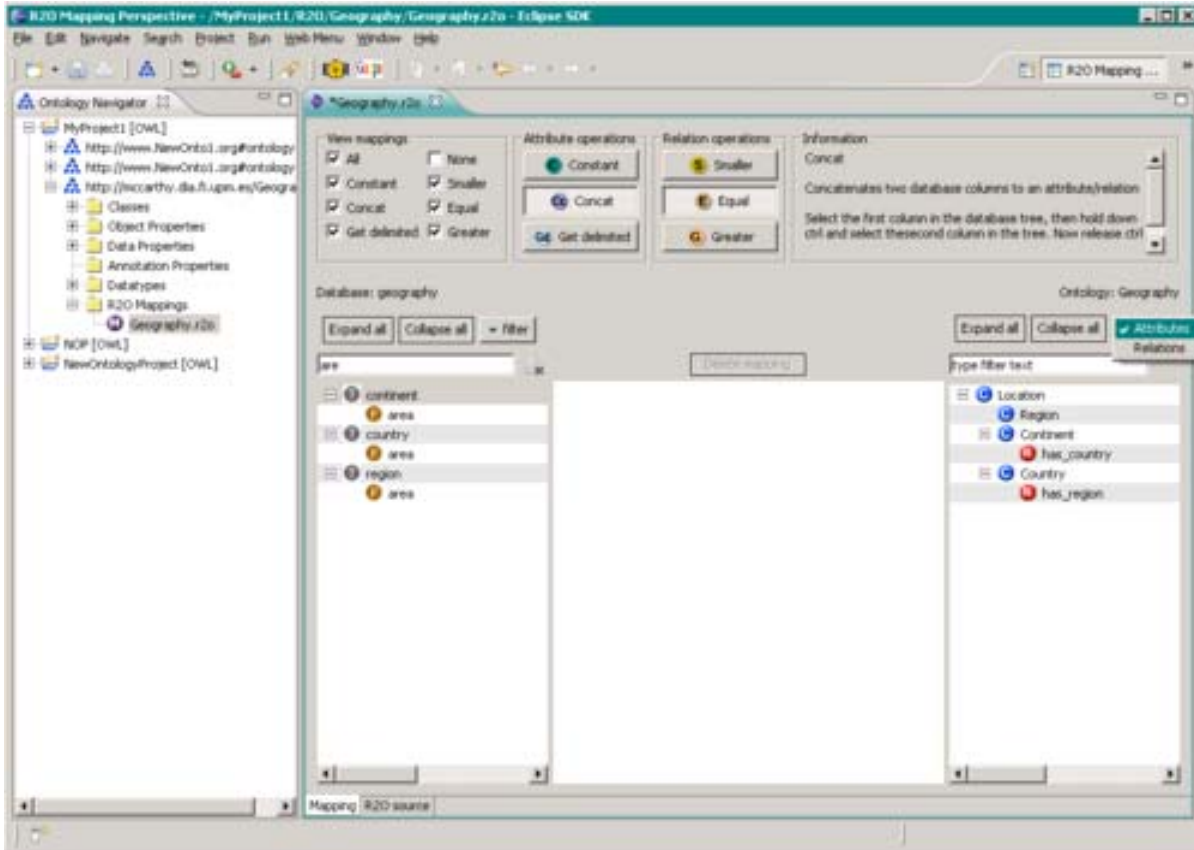


Figure 51: Filter ontology items

You also have the possibility to filter the ontology elements by name. Therefore click in the text box above the tree and type some letters. We typed cod in the text box, and as the result you can see that only ontology elements that begin with the letters cod are visible and all others are filtered out. Click on the button right of the text box to clear it and reset the filter to the normal view.

Expand and collapse the ontology tree

As the ontology can build up a very deep hierarchy, the editor allows you to collapse the tree to get a better view of the hierarchy. One can collapse the tree to every level in the hierarchy, making every element below that level invisible. Click on a concept in the ontology hierarchy. Now click on the collapse all button. The concept will collapse, and all its sub elements will disappear. The same happens for all other concepts on the same level.

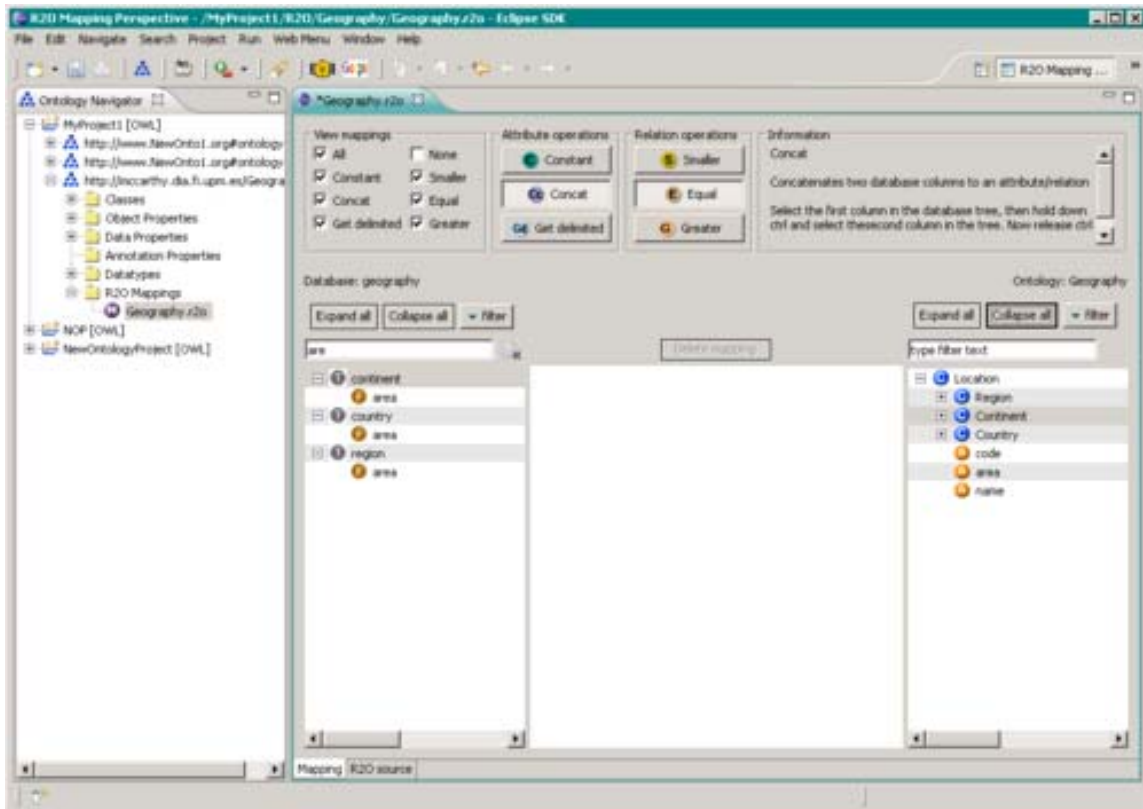


Figure 52: Expand and collapse the database tree.

Remove a mapping

If you do not want a mapping anymore, you can easily remove it. Select the mapping that you want to remove in the ontology tree. Then click on the delete mapping button. The mapping is then removed.

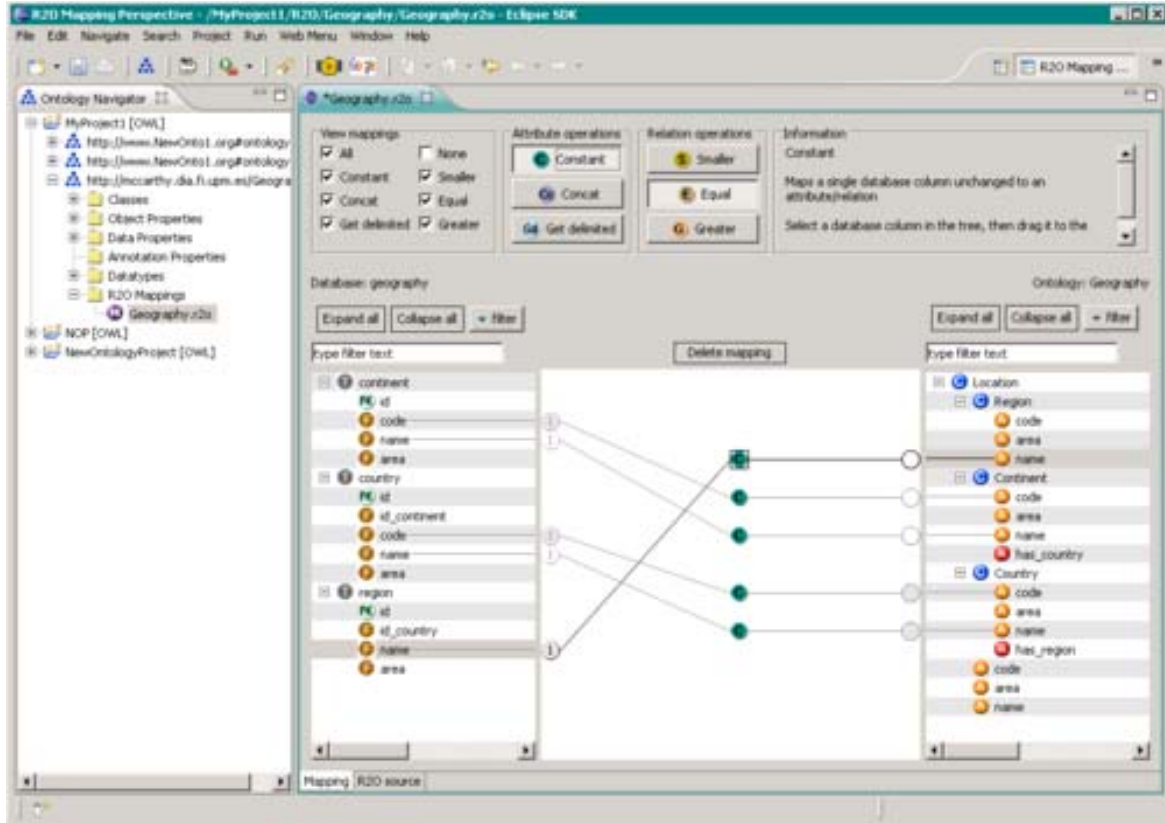


Figure 53: Remove a mapping

Query a mapping

After you have created several mappings you are now ready to query the mapping to get the RDF data from the SQL database. First of all you should save your mapping by clicking on the save icon in the top left corner of the NeOn Toolkit. In the NeOn ontology navigator click on the plus to extend the R2O mappings. Now you can see your mapping. Select it and right click on it to get the popup menu. Click on Query R2O Mapping to open the query editor.

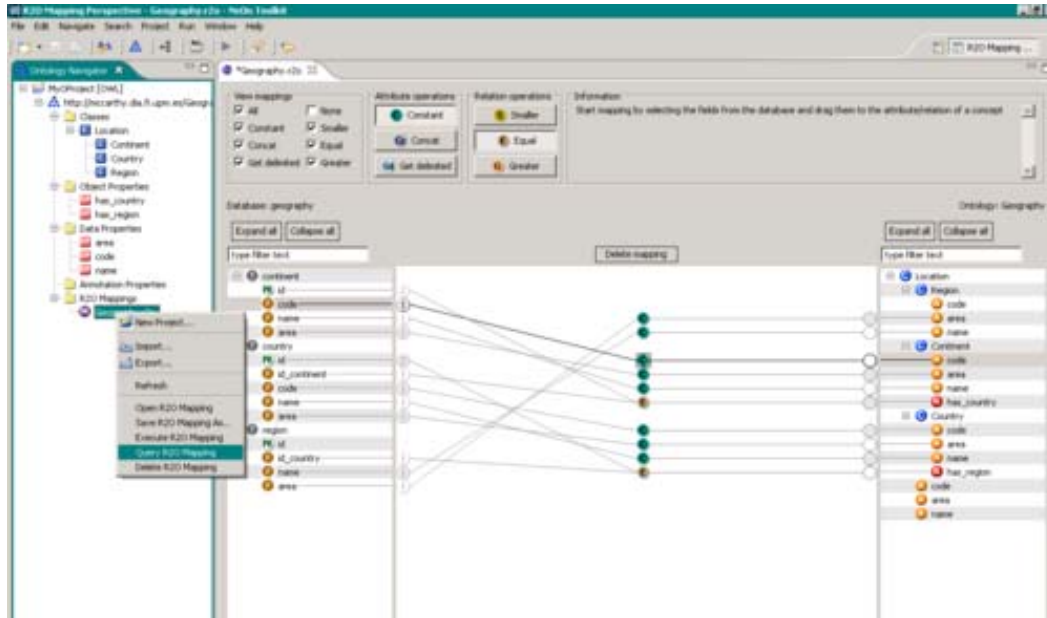


Figure 54: Query mapping menu item

The query editor opens and lets you select all the concepts with its attributes and relations, for which you have created a mapping. In the concepts tree select one of the concepts. All attributes and relations that have a mapping and can be selected appear now on the right side. You can manually select the attributes or choose predefined operations like e.g. Instances with attributes. This automatically selects all attributes of that concept. Now you are ready to start the processor. Click on Execute query to start the RDF creation. The RDF instances are created and you can see them on the right side of the query editor. This can usually take some time. You can stop the execution at any point by clicking the Stop execution button.

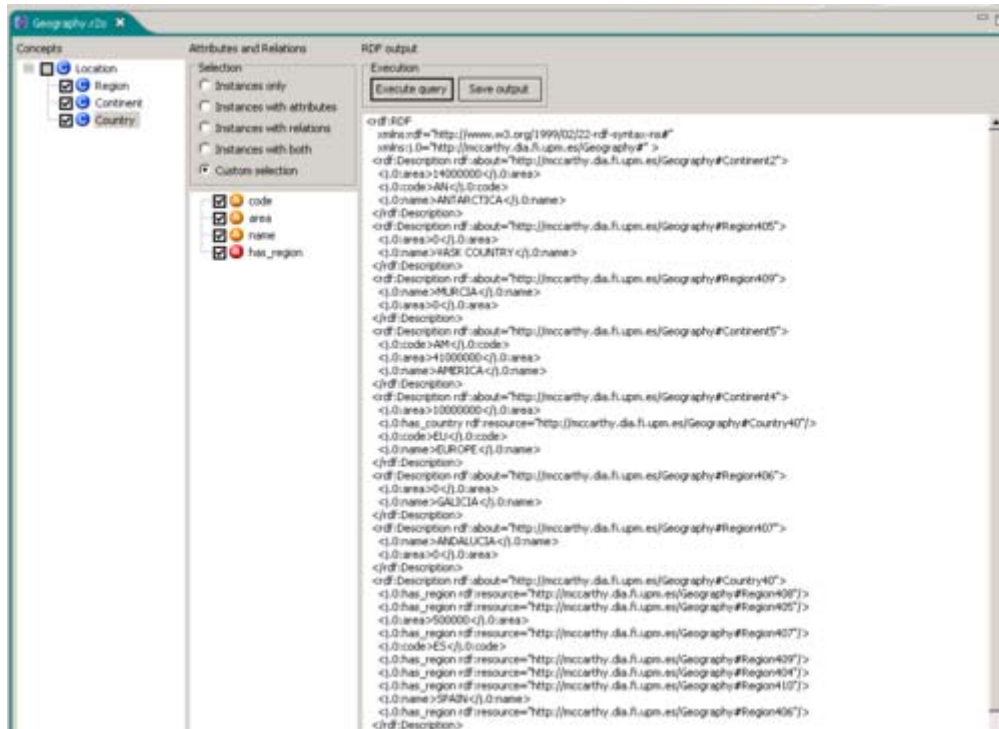


Figure 55: Query mapping window

3.11.3. Integration into the NeOn Toolkit

In this section we present the set of components according to the architecture proposed in WP6. The following figure depicts the proposed architecture.

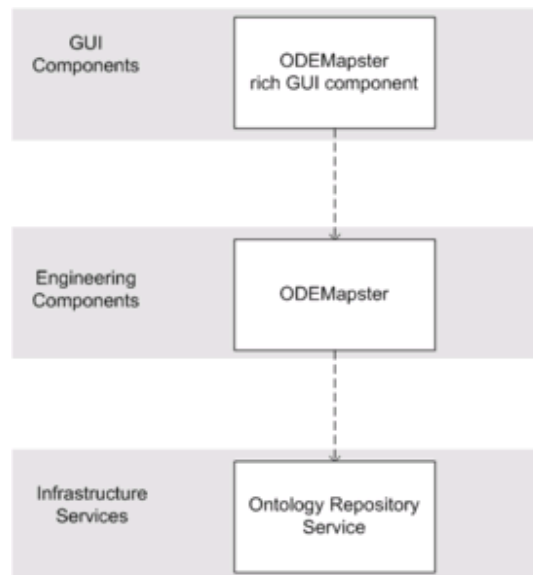


Figure 56: ODEMapster component within NeOn Toolkit.

3.11.4. Intended Usage in the Case Studies

ODEMapster is being used within WP7 and WP8.

- Motivation of using ODEMapster:

Within WP7, 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 relational database) into corresponding instances in the conceptualized fisheries ontology.

- Benefits/Advantages of using ODEMapster:

As it was mentioned before, ODEMapster is an engine that executes mappings between an ontology model and a database by means of a declarative language, R₂O. FAO, and ATOS are already using this plug-in; and they will use it for ontology population following the two approaches mentioned in the WP7 and WP8 deliverables: - Population based on lifting/upgrading/migration. - Population based on query driven.

- Data sets:

By now, ODEMapster works with MySQL and ORACLE databases, although graphical user interface allows users to work with MySQL database. ODEMapster works with OWL/RDF(S) ontologies.

- An usage example aligning to WP7 case study.

So far, FAO used ODEMapster to automatically populated the following ontologies (see figure): Land areas - Fishing areas - Biological entities - Fisheries commodities - Vessel types and size - Gear types.

For each ontology they created an R₂O document (that describes the correspondences between FIGIS DB and each ontology). Then, they ran the ODEMapster processor to populated the ontologies. These populated ontologies are available at <http://www.fao.org/aims/aos/fi/>. FIGIS DB is stored in a MySQL database and the ontologies are expressed in OWL (Figure 57).

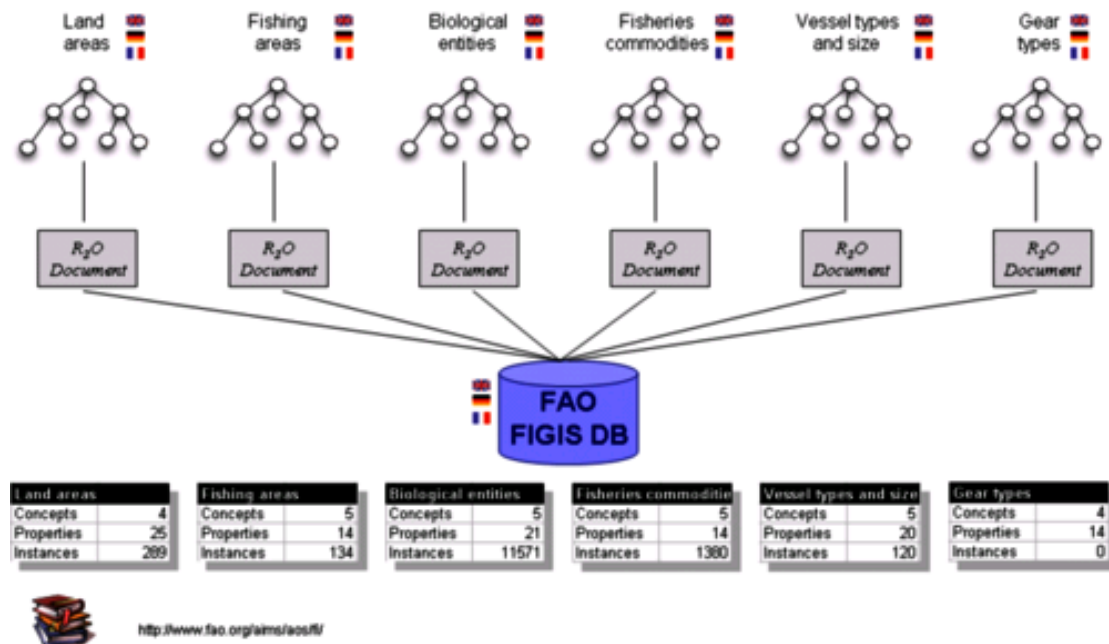


Figure 57: FAO Databases

3.12. OntoModel

3.12.1. Functional Description

OntoModel is a UML-based editor that works with OWL ontologies. The usage of the UML extension mechanism makes it possible to develop and maintains OWL ontologies with MDA technologies. In order to work with large ontologies it is possible to use several diagrams for a single ontology.

Over a module extraction mechanism it is possible to separate a part of an existing ontology into a module.

Provide a description of what the plugin does from a functional perspective. Specify which lifecycle activities are supported by the plugin.

3.12.2. User Documentation

After the installation process with the NeOn Toolkit update mechanism, a new node called "diagram" will appear in the Ontology Navigator, as a subnode of the ontology.

Create a new Diagram

To create a new diagram use the context menu of the "diagram" node in the Ontology Navigator and choose "Add View".

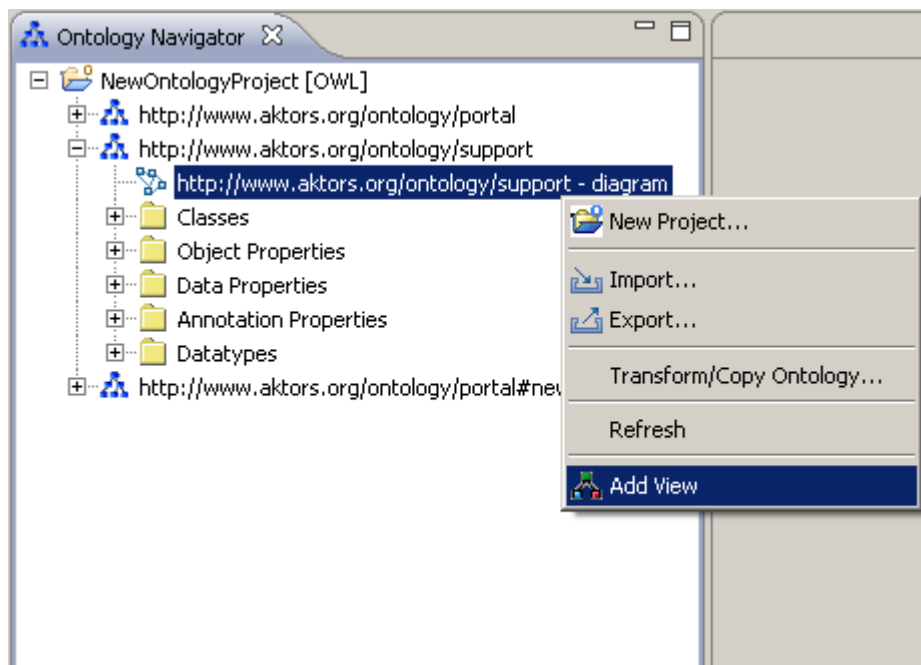


Figure 58: Adding a view

This will open the diagram perspective and show a blank diagram. To visualize elements in this diagram, drag and drop the element you want to show from the Ontology Navigator into the

diagram. It is also possible to drag and drop a selection of elements. Connections between dropped elements and existing ones in this diagram will appear immediately.

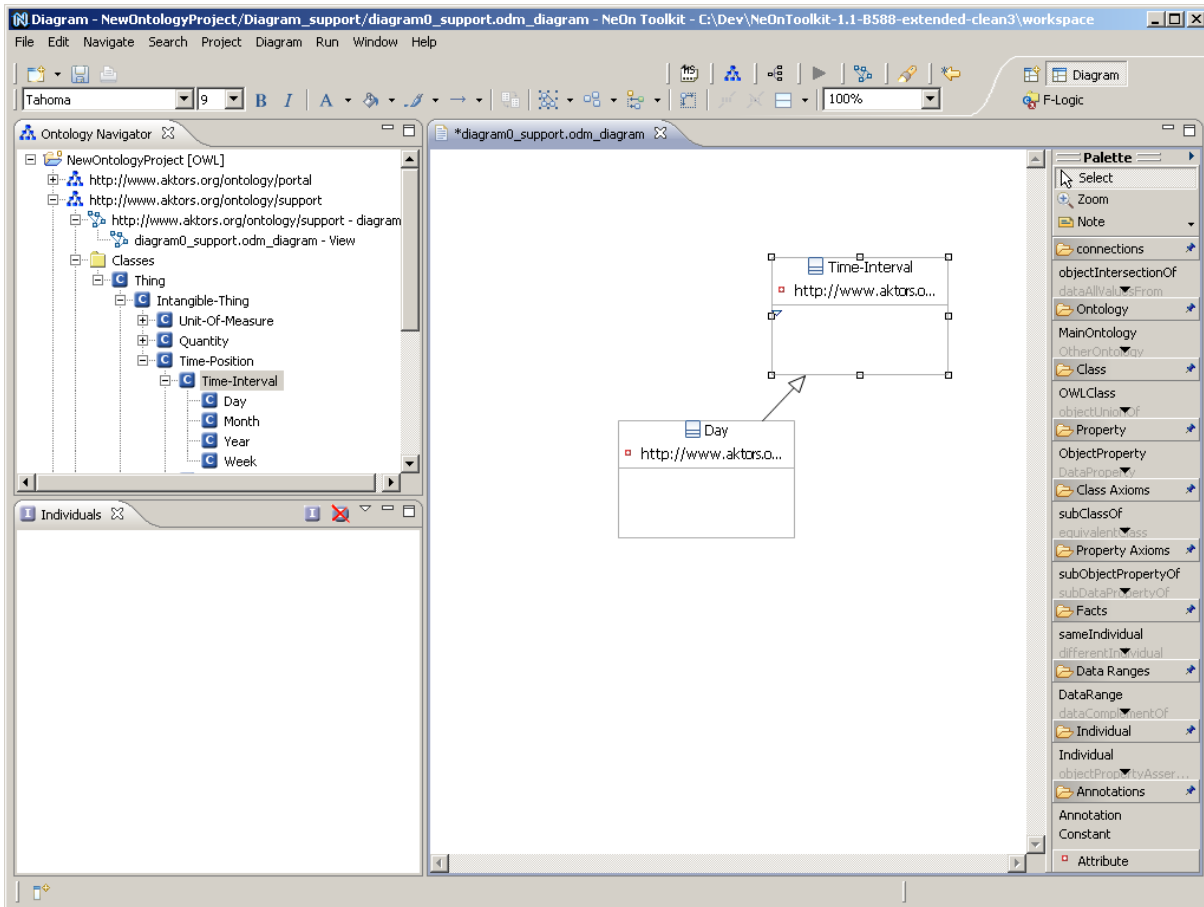


Figure 59: Ontology diagram

Over the "Add View" action you can add multiple views of the same ontology. Every time this action is invoked it will create a new blank diagram, which can be filled by the already described drag and drop mechanism with any element from the ontology. With this mechanism you can display and edit the same element in different diagrams. If you change a property of an element, this change will be synchronized immediately with other diagrams that show this element and with the Ontology Navigator. These diagrams will be saved in the project directory. To open an existing diagram, expand the diagram node and choose the diagram you want to show.

Changing a Ontology in the Diagram

With the tool box on the right it is possible to create the most common constructs for classes, individuals and properties. This new elements will also be created in the ontology and shown in the Ontology Navigator.

Extract a Module

OntoModel includes also a module extraction mechanism. You can split elements from an existing ontology into a self standing module. This mechanism will help you to keep the correctness of the separate elements.

To create a new module from an existing ontology, select the classes or individuals and properties you want to separate in the Ontology Navigator or in the diagram and choose "Create New Module" from the context menu. With the selection of some or all Object Properties you can configure the connections, which will be followed for one step if one included concept is in the domain of one of the selected Object Property. This means, that if an included class or the class of an included individual is in the domain of a selected Object Property, the property and the class in the range of this property will also be included in the new module. This assures, that each included property has a correct domain and range regarding to the included elements in the module.

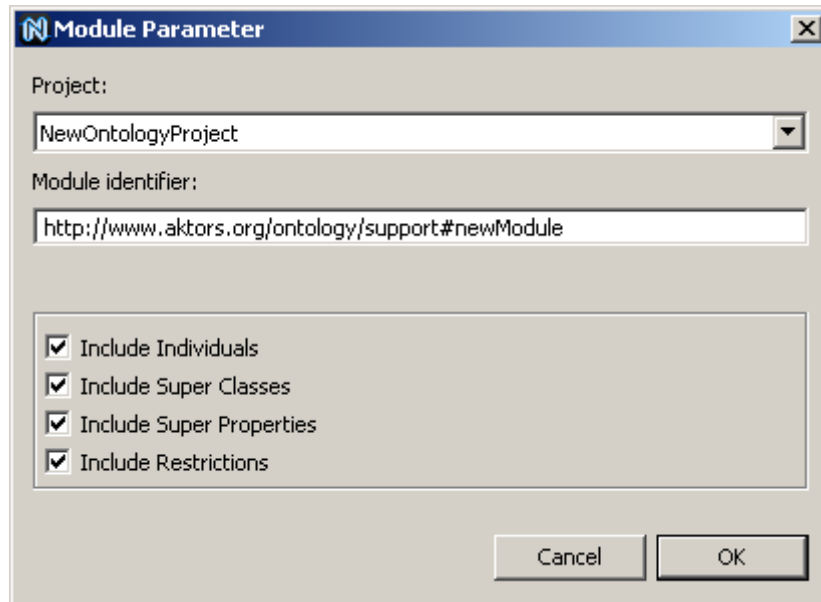


Figure 60: Module Parameters

In the displayed dialog you can configure the module extraction. You can choose the target project and set the namespace of the new module. Through the checkboxes it is possible to include more corresponding elements:

- **Include Individuals:**
This includes all individuals of all included classes
- **Include Super Classes:**
If an included class has a super class, this class will be recursively added to the module
- **Include Super Properties:**
This includes the super properties of Data and Object Properties like the super classes
- **Include Restriction:**
This will include restriction and the classes in a restriction of a included class, if the corresponding property is selected

The click on "OK" will start the extraction process. This can take a few seconds.

Add Elements to an existing Module

If you want to add elements from an ontology to an existing module, select the elements in the Ontology Navigator and drag and drop them into a diagram of the target module. This brings on the same dialog as described above. Except the target project and the namespace, you have exactly the same configuration options with the result, that the extracted elements will be added to the target module.

3.12.3. Integration into the NeOn Toolkit

OntoModel provides a new Eclipse perspective within the NeOn Toolkit.

In addition, it makes use of the EMF, GEF and GMF Frameworks.

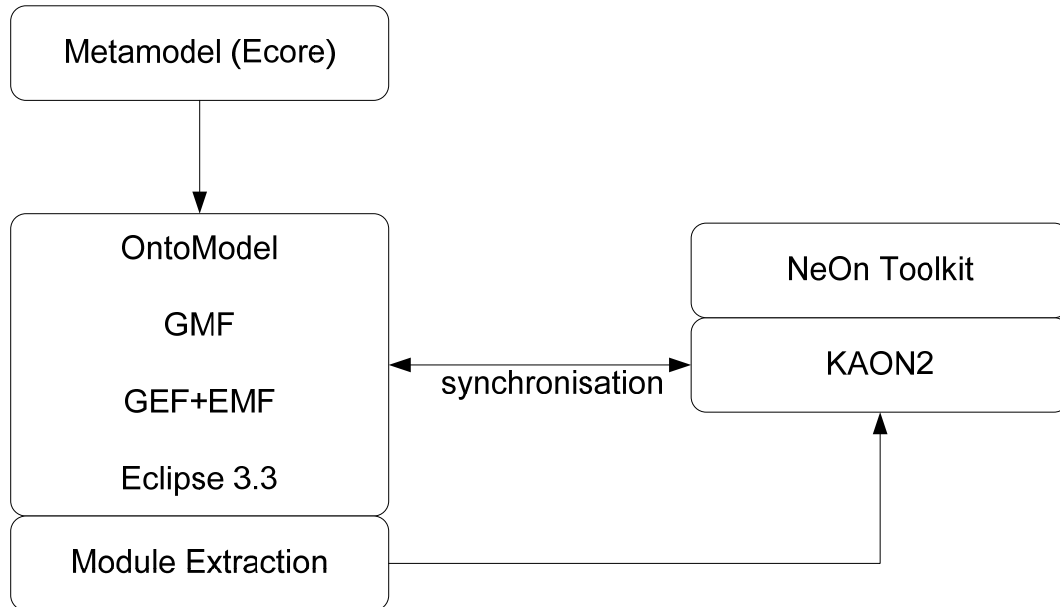


Figure 61: OntoModel Architecture

The Figure shows how OntoModel builds on Eclipse and some of its available plug-ins: it builds on the Graphical Modeling Framework (GMF⁶), which in turn builds on the Graphical Editing Framework (GEF⁷) and the Eclipse Modeling Framework (EMF⁸).

EMF is a code generation facility for building applications based on a structured model. It helps to turn models into efficient, correct, and easily customizable Java code. Out of our Ecore metamodel, we created a corresponding set of Java classes using the EMF generator. The generated classes can be edited and the code is unaffected by model changes and regeneration. Only when the edited code depends on something that changed in the model, that code has to be adapted to reflect these changes.

EMF consists of two fundamental frameworks: the core framework and EMF.Edit. The core framework provides basic generation and runtime support to create Java classes for a model, whereas EMF.Edit extends and builds on the core framework, adding support for generating a basic working model editor as well as adapter classes that enable viewing and editing of a model.

EMF started out as an implementation of the MOF specification. It can be thought of as a highly efficient Java implementation of MOF, and its MOF-like metamodel is called Ecore.

The EMF adapter listens for model changes. When an object changes its state, GEF becomes aware of the change, and performs the appropriate action, such as redrawing a figure due to a move request. GEF provides the basic graphical functionality for GMF.

⁶ <http://www.eclipse.org/gmf/>

⁷ <http://www.eclipse.org/gef/>

⁸ <http://www.eclipse.org/modeling/emf/>

GMF is the layer connecting OntoModel with GEF and EMF. It defines and implements many functionalities of GEF to be used directly in an application and complements the standard EMF generated editor.

The Module Extraction component is build separately from the EMF data model. It's operates with the KAON2 API of the NeOn Toolkit. The synchronisation functionality keeps the EMF and KAON2 data models in synch.

3.12.4. Intended Usage in the Case Studies

The OntoModel system has already been applied in the pharmaceutical case study.

With OntoModel, we addressed the uses case of building, adapting and visualizing the common Pharmainnova ontology as well as the partners' ontologies.

As the typical end user in this scenario is not experienced in modeling ontologies, we provided the user with the Pharmainnova ontology. The user can adapt this ontology until it reflects the partner's invoice structure, instead of modeling the ontology from scratch themselves.

Thus, the demanded input from the end users is expected to be considerably lower. When the user finished adapting the ontology, he specifies which elements of both ontologies (the Pharmainnova ontology and the own ontology) can be mapped to each other, for example using an equivalence relation.

3.13. OWLDoc

3.13.1. Functional Description

The OWLDoc plugin adds to the NeOn Toolkit an option to export an OWL-DL ontology as an HTML Documentation. This plugin uses the KAON2 datamodel API to extract information from the OWL Ontology and creates an output that contains an organized set of HTML files that provide the documentation about the ontology and all its resources.

3.13.2. User Documentation

To use the OWLDoc plugin, simply select the Export option in the File Menu or after right-clicking in the explorer. In the Export menu, select the OWLDoc Export Wizard in the NeOn Toolkit category. In the OWLDoc menu, select the project and the ontology you want to export. Select the directory where the HTML files of the documentation should be stored, and click Finish.



Figure 62: OWLDoc Export

3.13.3. Integration into the NeOn Toolkit

The OWLDoc plugin basically creates a new option in the export menu of the NeOn toolkit, to export an ontology into an HTML Documentation. The plugin extracts the ontology from the NeOn toolkit in an OWL model, with the use of the KAON2 API.

- Eclipse Extension Points
 - org.eclipse.ui
 - org.eclipse.core.runtime
 - org.eclipse.core.resources
 - org.eclipse.ui.views
- NeOn Toolkit extension points
 - com.ontoprise.ontostudio.gui
 - com.ontoprise.ontostudio.datamodel
 - org.neointoolkit.gui
 - com.ontoprise.ontostudio.io
 - dependencies
 - util
 - datamodel
 - datamodelBase

3.13.4. Intended Usage in the Case Studies

The OWLDoc plugin will be applied in the WP7 FAO case study in the context of OWL ontologies documentation. Continuous feedback from use case is obtained in order to improve OWLDoc.

3.14. Oyster Registry

Oyster is a distributed registry that exploits semantic web techniques in order to provide a solution for exchanging and re-using ontologies and related entities. As an *ontology registry*, it provides services for storage, cataloguing, discovery, management, and retrieval of ontology (and related entities) metadata definitions. To achieve these goals, Oyster implements the proposal for metadata standard OMV (<http://omv.ontoware.org>) as the way to describe ontologies and related entities, supporting advanced semantic searches of the registered objects and providing services to support the management and evolution of ontologies in distributed environments. Hence, Oyster provides services for the propagation of ontology changes to distributed copies of the ontology metadata using a synchronization mechanism. For more information we refer the reader to <http://oyster.ontoware.org>.

Synchronization: Oyster follows a synchronization approach that is a combination of a push and pull mechanism. During the synchronization, nodes contact other nodes in the network to exchange updated information (pull changes) and optionally they can push their changes to a specific node (called the push node) such that if a node goes offline before all other nodes pull the new changes, the node changes are not lost.

3.14.1. Functional Description

The Oyster Registry feature consists of a number of individual plugins which provide the following functionality:

Configuration/management of the registry

- **Start/Stop the Ontology Registry:** This functionality is available from the Registry menu and/or from the action associated to the Oyster icon in the main toolbar;
- **Configure the initialization parameters:** This functionality is available from the Oyster storage preference page. It provides the following configuration options:
 - **Super Node IP:** This option allows to configure the IP address of a "super" node that will be used by the registry to store/retrieve the metadata. Any node in the network running Oyster can be configured as the "super" node, however in general this is a node running Oyster in server mode. Usually this option is configured when the user will be working with a NTK collaboration server;
 - **Push Node IP:** This option allows configuring the IP address of a specific node in the network to which changes will be push propagated every time the Synchronization process is invoked. Usually this option is configured when users are working collaborative on the development of ontologies in a totally distributed environment (i.e. users are working with local copies of the same ontology and there is no NTK collaboration server) to ensure that every node in the network interested in the same ontology will be able to pull propagate changes from other nodes even if the nodes go offline;
 - **Read ontologies locally:** This option allows to specify that the ontologies necessary to start the registry (e.g. OMV and related) are loaded from the local filesystem. By default ontologies are loaded from the internet and therefore it takes a little bit longer to start the registry. If this option is selected, the user should ensure that the required ontologies are available in the O2ServerFiles directory within the NTK working directory (i.e. where the NeOn_Toolkit.exe file is located). Users can download the pack of ontologies from <http://oyster2.ontoware.org/ontologiesPack>.
 - **Reset Oyster:** This option allows the user to reset Oyster i.e. delete the current information and settings. To apply this action, the user should ensure that the registry is not running.

Registry GUI

The registry GUI is an interface integrated into the NeOn Toolkit that allows to access Oyster P2P network through any available node running Oyster Web Service (aka Oyster server).

Main Features:

- Submit new OMV instances to an Oyster server.
- Update already existing OMV instances at an Oyster server.
- Remove OMV instances from an Oyster server.
- Import ontologies which metadata is retrieved from an Oyster server.
- Look for OMV instances in an Oyster server.

3.14.2. User Documentation

How to install it

- Download the compressed Plugin release from <http://ontoware.org/projects/oyster2>
- Extract and copy the JARs files into the plugins directory inside the filesystem location where NeOn Toolkit is installed.
- If the user wants the registry to read locally the ontologies required to start (see above), he should ensure that the required ontologies are available in the O2ServerFiles directory within the NTK working directory (i.e. where the NeOn_Toolkit.exe file is located). Users can download the pack of ontologies from <http://oyster2.ontoware.org/ontologiesPack>.

How to use it

The following figures show the Oyster Registry feature functionalities:

Configure/Manage the registry

a) Start the registry:

The registry can be started from the Registry menu and/or from the action associated to the Oyster icon in the main toolbar.

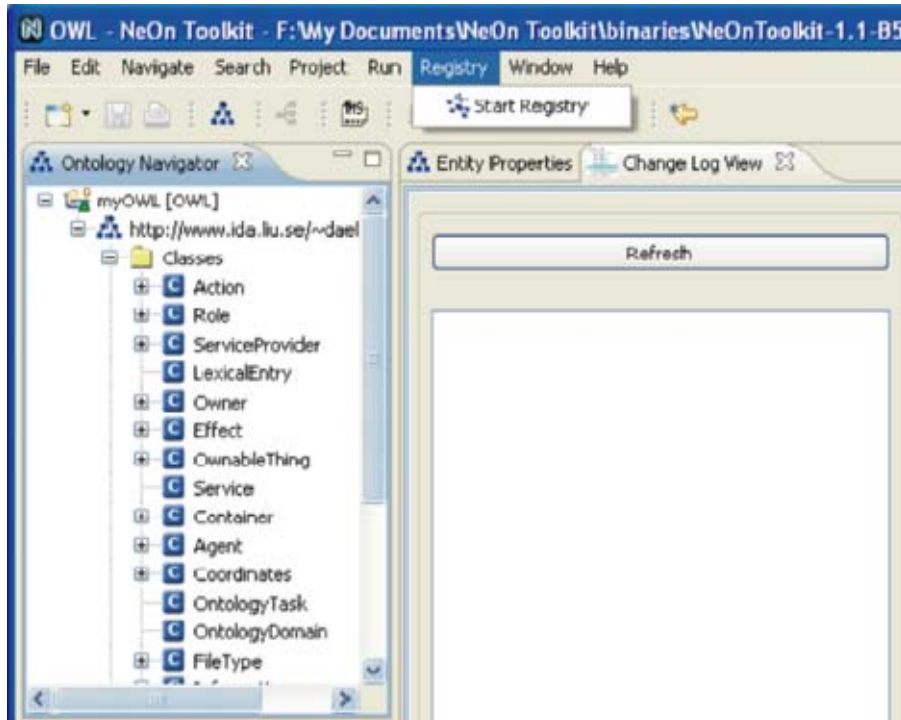


Figure 63: Starting Oyster

Starting the registry usually takes some time, depending whether the ontologies are read locally or from the internet. In any case, the process is associated to a progress bar that can be run in the background.

The user can easily tell if the registry is running by checking the icon or by checking in the registry menu: when it is running, the icon is pushed and the registry menu only shows the option to stop the registry (and vice versa when it is not running).

b) Stop the registry

Similar to the start functionality, the registry can be stopped from the Registry menu and/or from the action associated to the Oyster icon in the main toolbar.

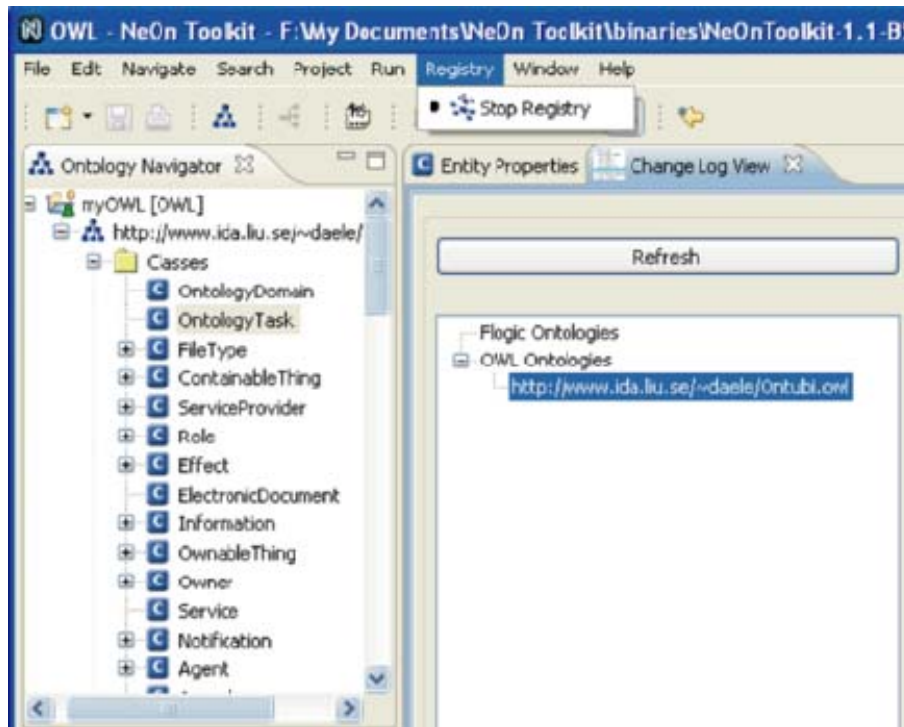


Figure 64: Stopping Oyster

c) Configure initialization parameters

The configuration of the startup parameters of the registry can be done from the Oyster storage preference page:

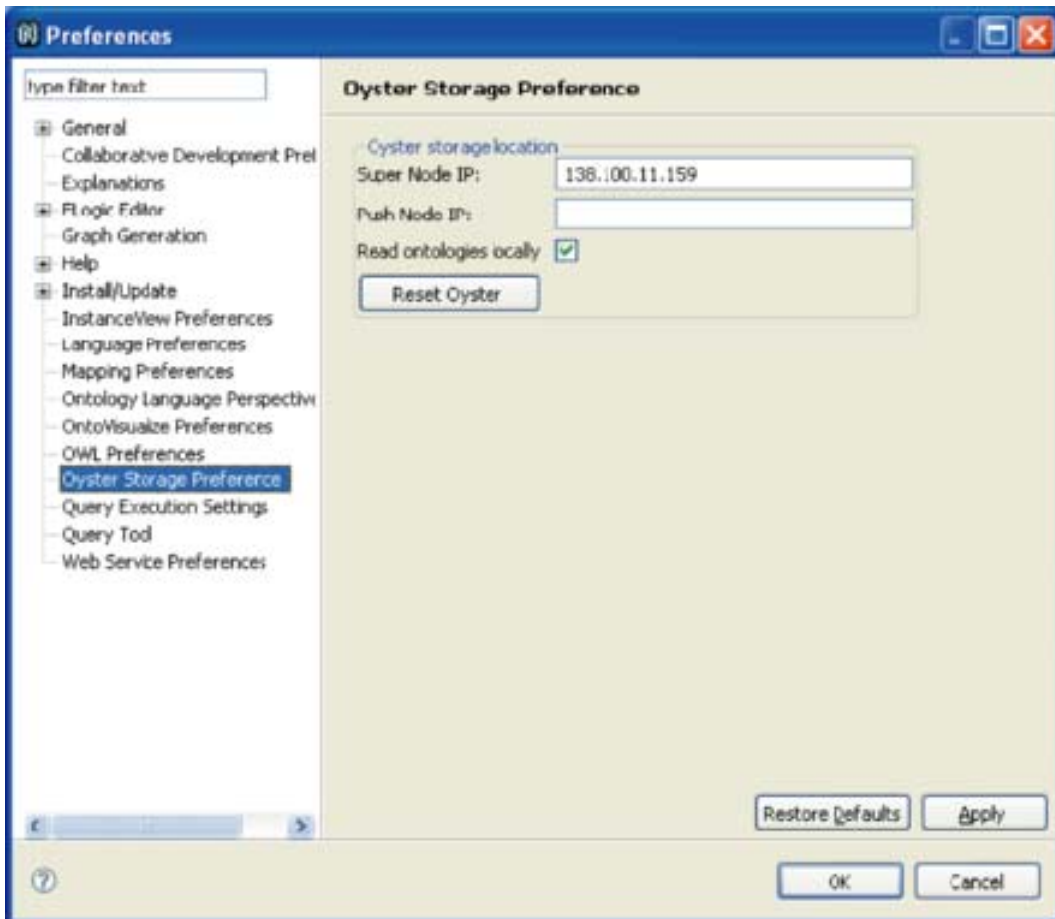


Figure 65: Oyster Preferences

Registry GUI

Server selection

The NeOn Toolkit Oyster GUI connects to a server to perform almost any operation. These are submit metadata, update metadata, delete metadata and query for instances of OMV classes.

When performing any operation that needs to connect to a server, there will be always the chance to select it, or to edit the list of servers. This page presents two examples, and the same composites are used for every feature, so it's always the same.

There will be always a combo holding the list of known servers, like this one:



Figure 66: Server selection combo

To select the server just click on it and a complete list of all known servers will be shown.

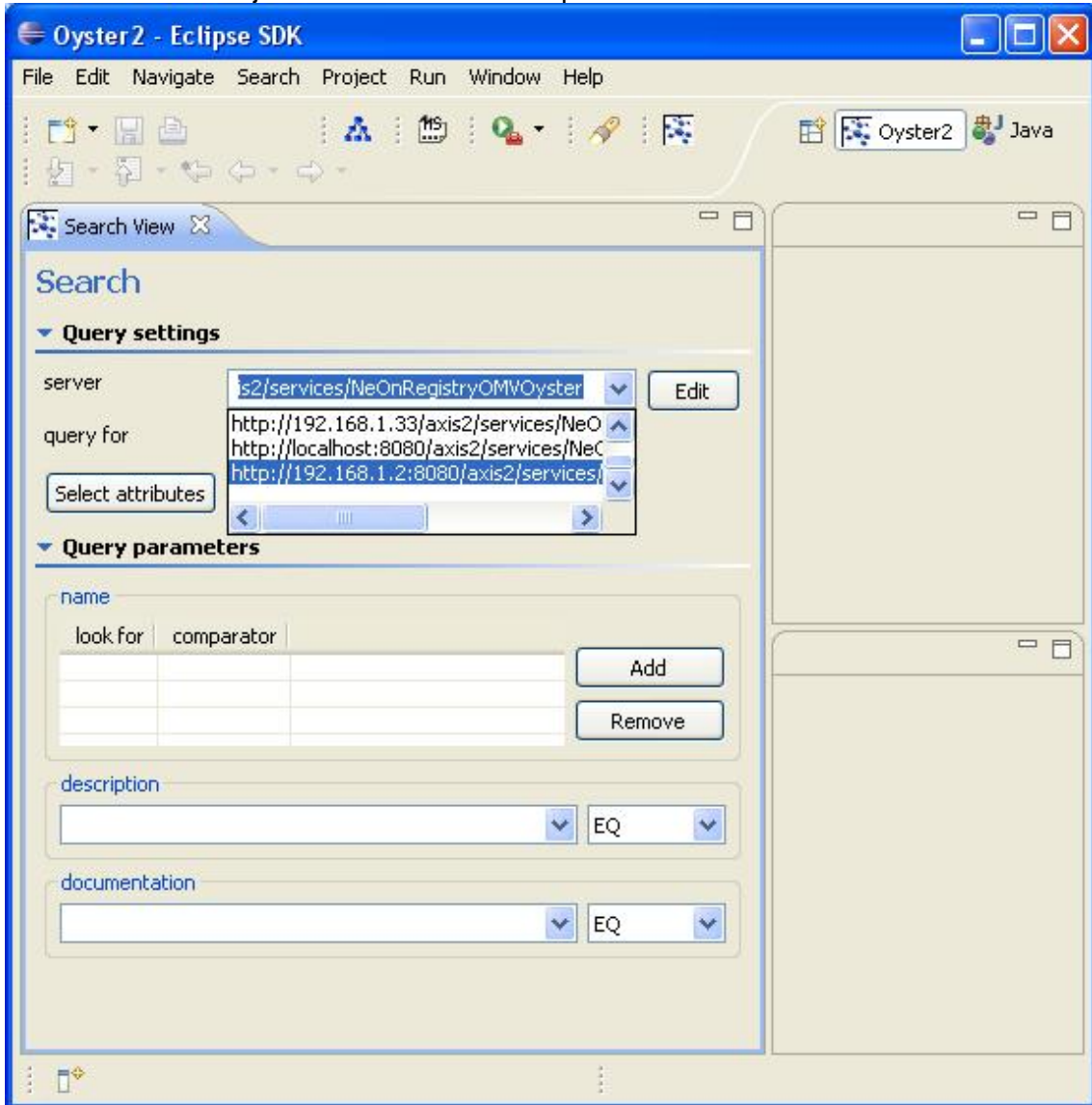


Figure 67: Server list shown by the server selection combo

At any time when the server selection combo is displayed, there will be also an edit button, to let you change the known servers list. Clicking on it will show the following dialog:

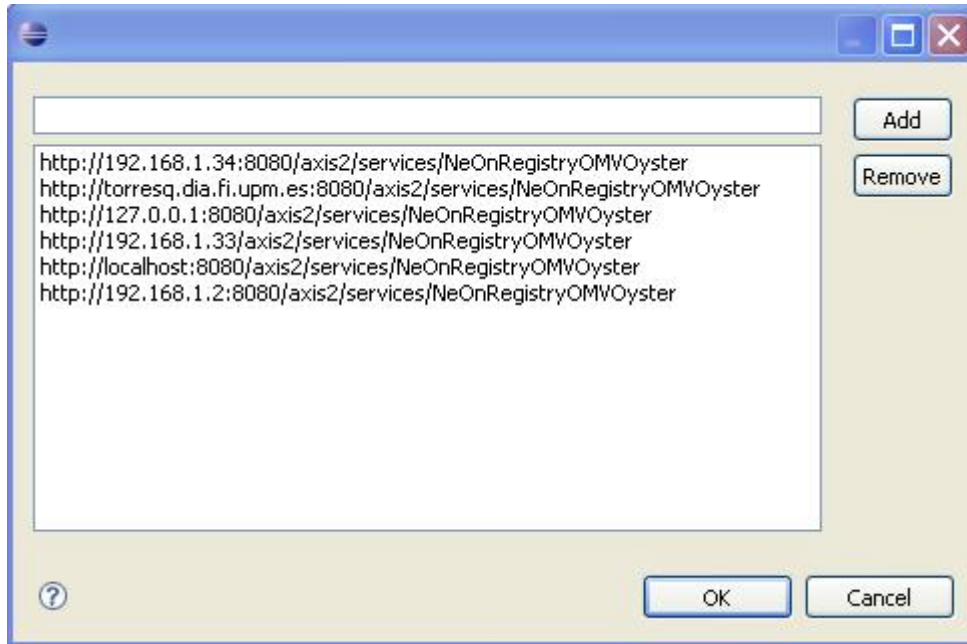


Figure 68: Server list dialog

This dialog lets the user type a new URL in the text box (in the top of the dialog) and add it to the list using the add button, and select a server of the list and remove it using the remove button.

Searching OMV Instances

The search feature can be used from the Oyster perspective, using the search view. It will make use of at least another view (the search results view), so although it's possible to open the search view from any perspective it's recommended to use it while in the Oyster perspective, to avoid cluttering the NeOn Toolkit interface with too many views.

The search view has to sections:

- The configuration section. This section allows the user to select the target server that will receive the queries, the OMV class which instances you are looking for, and select the attributes used to perform the query
- The parameters section. This section is meant to receive the query parameters. These parameters are property values that must be present in the instances returned by the query.

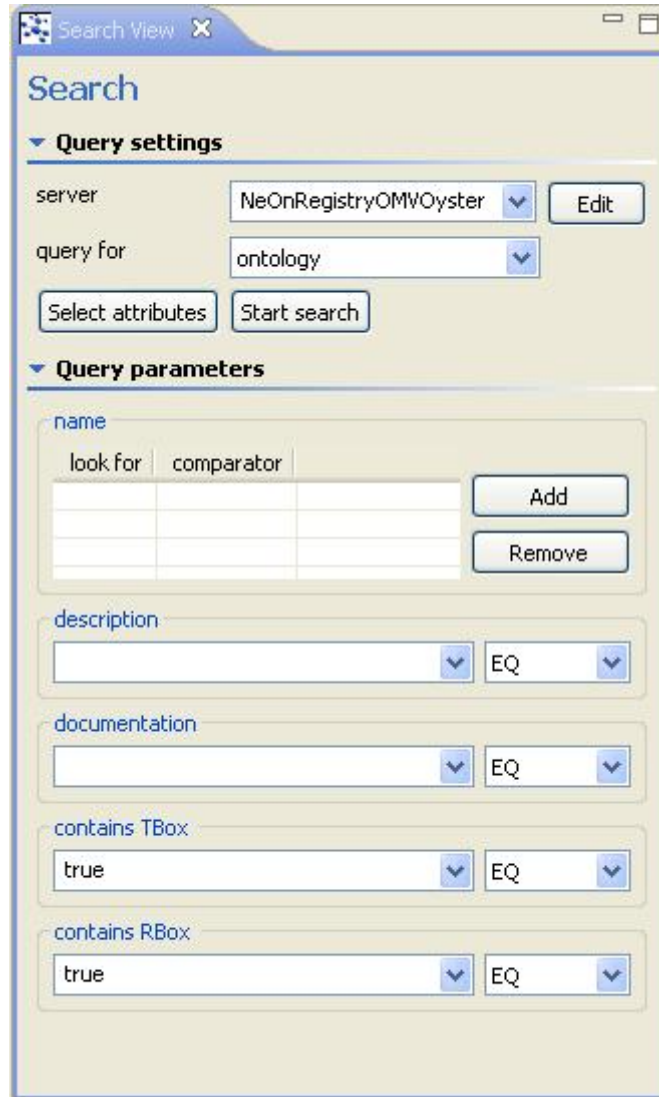


Figure 69: The search view

The configuration section

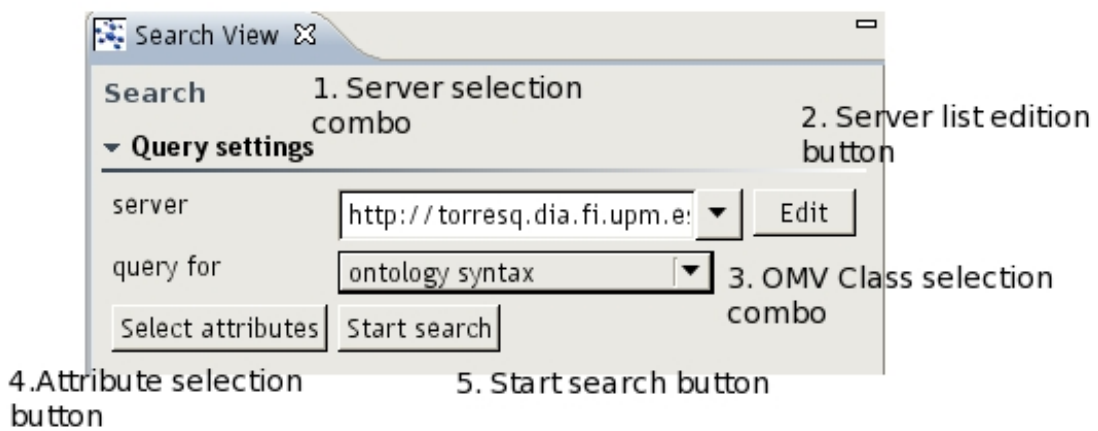


Figure 70: The configuration section

The configuration section

As it was said earlier, here are the following controls:

1. Server selection combo (see server selection).
2. Server selection list edition button.
3. OMV class selection combo. It allows to select the class of the instances to be looked for.
4. Attribute selection button. This allows to select which attributes from the OMV class are to be used in the query.
5. Start button. This starts the search with the given query parameters.

Selecting query parameters

Once the OMV class is selected, the "select attributes" button will show a list of the attributes of the selected class.

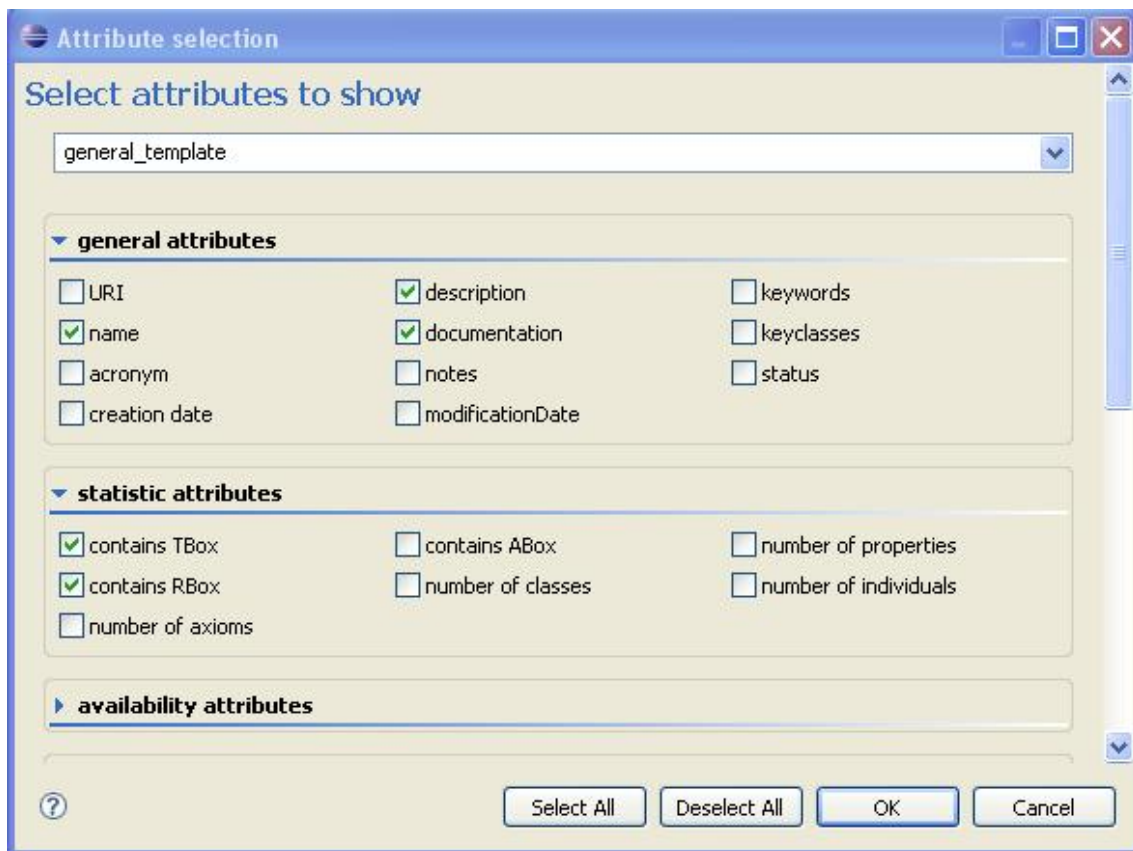


Figure 71: Properties selection dialog

The main controls of the attribute selection dialog are:

- Template selection combo. Lets the user select between a predefined list of templates. Selecting a template will check the attributes of the template, and uncheck others.
- Select all and Deselect all buttons. They mark every property as selected or clear the selection, respectively.
- The attributes section. Lets the user check or uncheck properties individually.

Once the property selection has been changed by pressing the "Ok" button in the attribute selection dialog, the changes will be reflected instantly in the fields section of the search view.

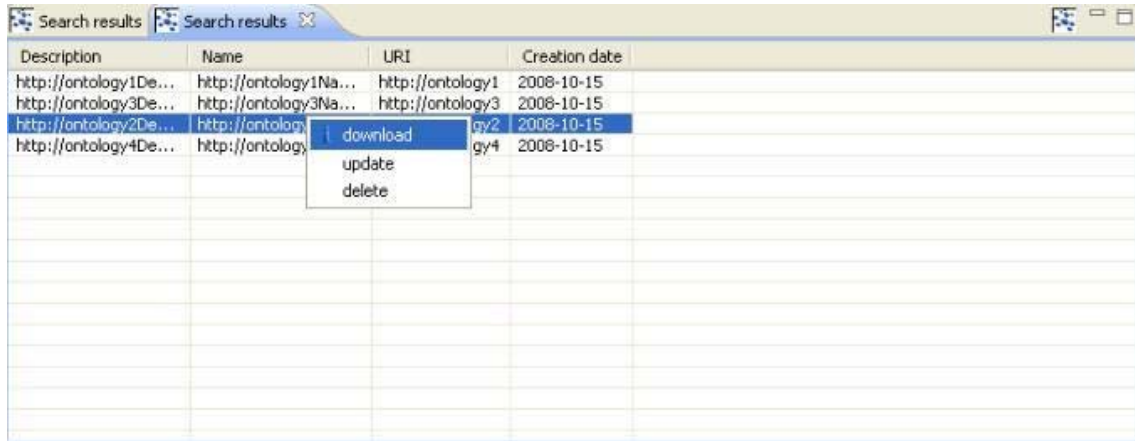


Figure 73: The import ontology entry in the context menu

Once you have done so, a new dialog will appear. It will show the selected ontologies that are about to be imported, and a list of existing projects. At this point the only choice is to select the project where the ontology will get imported into. All ontologies in the dialog will be imported, regardless of the selection of them the user makes here (thus the user must be careful selecting from the results view).

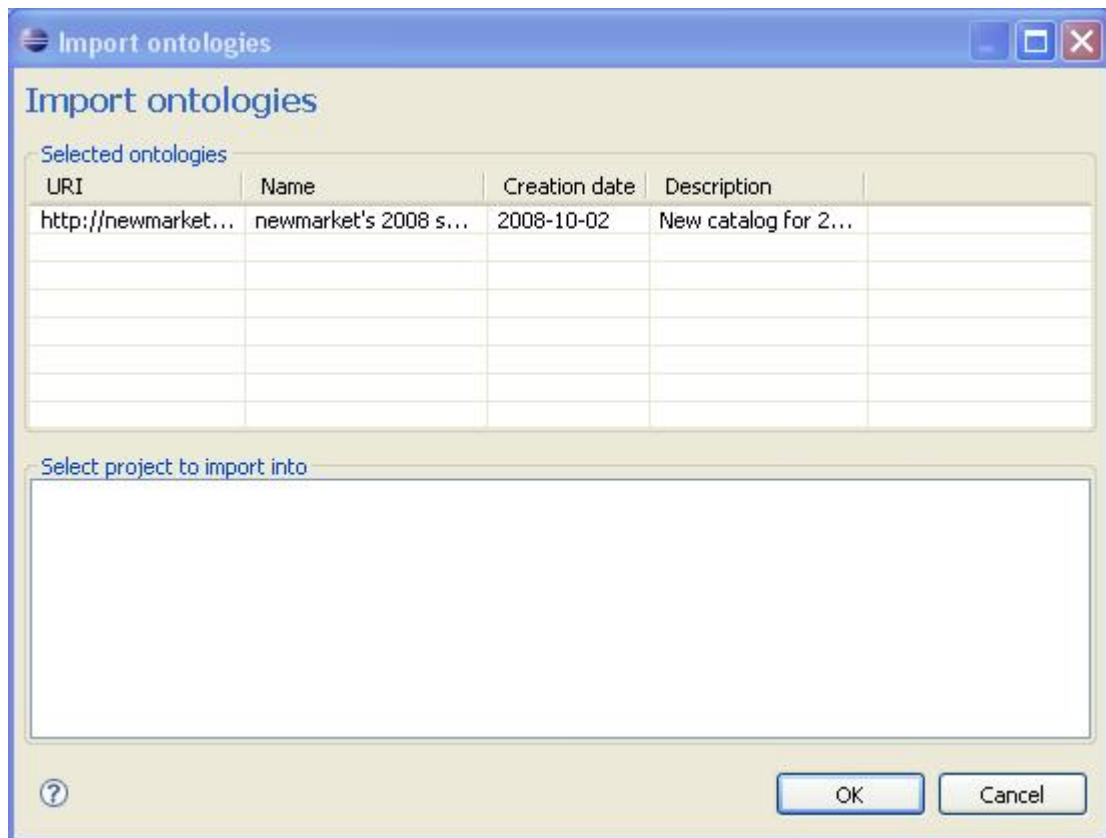


Figure 74: The import ontology dialog

When the user presses the OK button the importation process begins. Bear in mind it may take a while, since the ontologies must be got from web and ftp servers, which in turn may be busy or even not working anymore.

The submit metadata feature

The NeOn Toolkit Oyster plugin comes with the capability of submitting new instances of OMV classes to an Oyster server. This feature is always accessible through a button on the toolbar, as depicted in the following picture:



Figure 75: Toolbar detail with the submit metadata button on the top right corner

Once the dialog has been brought up, the user has the choice to

- Select the server to receive the metadata.
- Select the class of the OMV instance that is going to be submitted.
- Select the properties that will have a value in the instance.

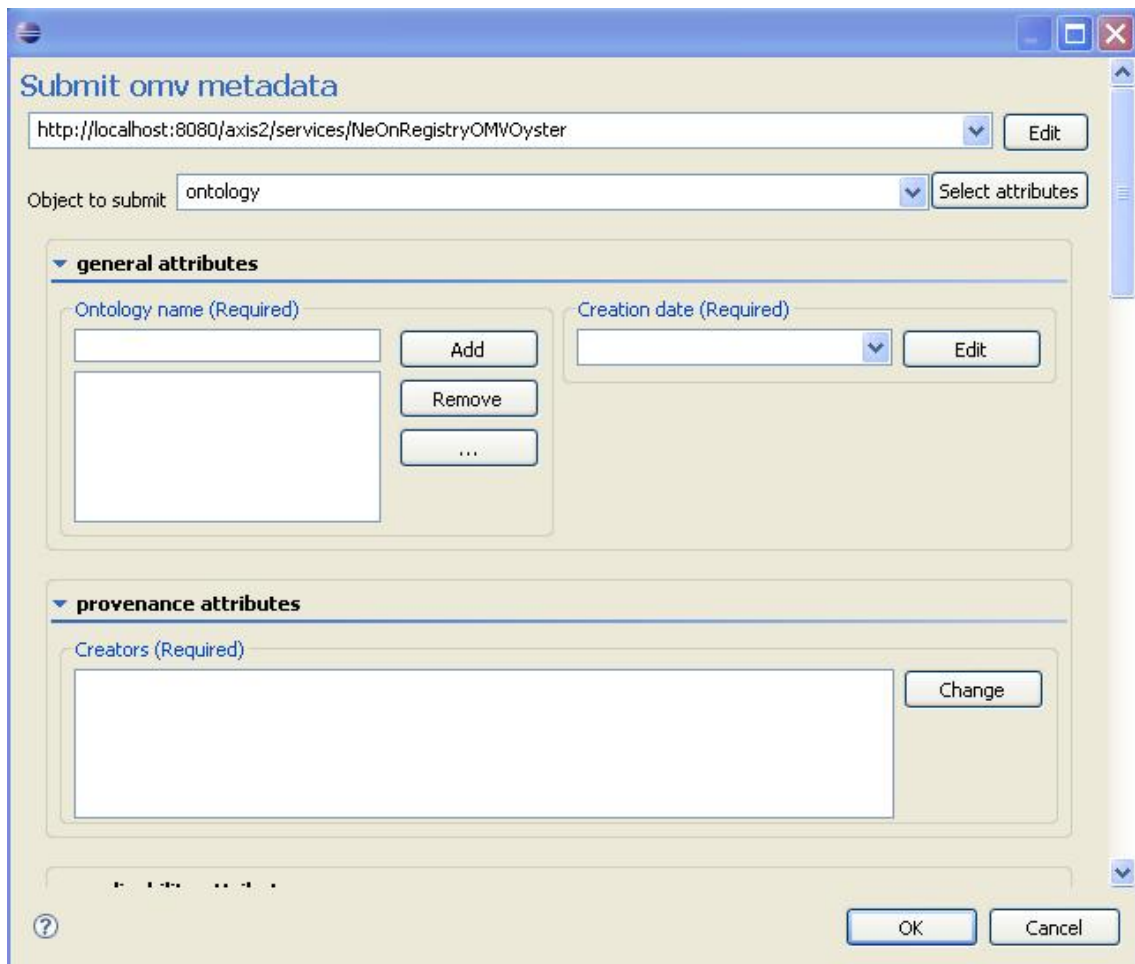


Figure 76: The submit dialog

Selecting the OMV Class

The first step is to choose the OMV Class to make the instance from. Currently almost all OMV Core classes are supported, with the exception of "Location", which is not supported by the Oyster server, thus the client does not make use of it.

Selecting the properties

OMV classes have required and optional properties, and because of that the user is not forced to supply values for every property of the class. When the user pushes the Select attributes button, a new dialog will appear, allowing the user to select which properties will get a value. On the top of this dialog there is a combo box that will let the user choose between templates, so if the class has many properties (such as Ontology) the user may start with a predefined set of properties selected. Note that there may be no more than a single template when the class has too few properties.

Providing values for the properties

Once the user has selected the properties that will have a value, he must start filling the input controls. Depending on the kind of value the property accepts, the control will be different to help the user with his task.

Here is a list of available The update metadata feature

In addition to submit metadata (OMV class instances) the user may want to update existing metadata about a concept. To do so the user must perform a search first, so the NeOn Toolkit Oyster GUI will populate the update dialog with the current property values of the instance.

The first step, assuming the user has at least a result displayed in the search results view, is to open the context menu using the mouse right button to click on the result to be updated. In this menu, choose the update option.

Description	Name	URI	Creation date
http://ontology1De...	http://ontology1Na...	http://ontology1	2008-10-15
http://ontology3De...	http://ontology3Na...	http://ontology3	2008-10-15
http://ontology2De...	http://ontology2Na...	http://oi	
http://ontology4De...	http://ontology4Na...	http://oi	




Figure 77: The update entry in the context menu

The dialog shown to the user is exactly the same as the one from the submit metadata feature, but values are already assigned. There are two things the user has to bear in mind regarding this dialog:

The properties with party values are not filled. The user must provide the values if he wants them to have any values at all. This is because the server won't specify which kind of party are the returned values, and the query for parties are not currently supported, so the NeOn Toolkit Interface has no means to know about party types.

If the user uses the Select attributes button to hide properties, they will lose their values in the update process, so they must not be hidden if they are to get a value or preserve it.

The controls found in this dialog are the same as in the dialog of the submit feature.

The delete metadata feature

In addition to the submit and update metadata features, the NeOn Toolkit Oyster GUI has the capability of removing metadata from the Oyster server (in other words, removing instances of OMV classes from the registry).

To remove an OMV instance, the user needs to retrieve the OMV instance to be deleted using the search feature. When the results view shows the instance the user must select it and bring up the context menu to select the delete entry.

3.14.3. Architecture Description of integration with NeOn toolkit

The NeOn basic infrastructure layer consists of a set of core services (e.g. Repository services, Registry services, etc). Oyster is an implementation of the registry services. As it is required, it is based on the OMV ontology meta model. Further, it provides an API and a Web Service interface to query, create, and manipulate ontology metadata information according to the OMV model. The web service provides a loosely couple service at the Engineering components layer. Additionally, Oyster provides additional services to support engineering components that rely on the registry services (e.g. Change Capturing). Finally, Oyster also provides GUI components (i.e. plugins) that implement user interfaces to the registry services.

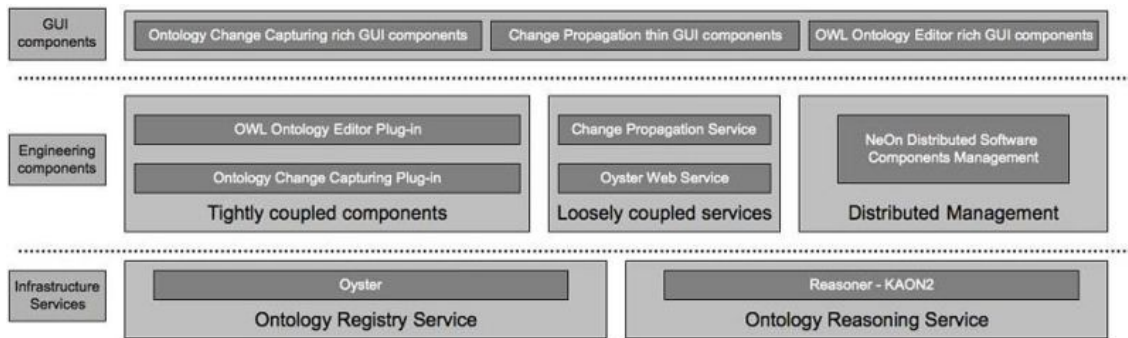


Figure 78: Oyster architecture

The following table shows the dependencies and compatibilities of the current versions of the individual plug-ins:

Version	Compatible with	Necessary plugins
Oyster-API 2.3.1	-----	-----
Oyster-menu 1.8.1	NeOn Toolkit 1.2	org.neontoolkit.registry.api com.ontoprise.ontostudio.gui org.junit4
Oyster-GUI 1.03	NeOn Toolkit 1.2	org.eclipse.ui org.eclipse.core.runtime org.eclipse.ui.forms org.eclipse.core.resources

		com.ontoprise.ontostudio.gui com.ontoprise.ontostudio.io com.ontoprise.ontostudio.datamodel datamodel datamodelBase util org.neontoolkit.registry.api org.junit4
--	--	---

3.14.4. Intended use in case study

This component will be used in WP7 to access the ontology registry.

Motivation of using Oyster

One of the goals of the FAO use case partner is that fisheries ontologies produced within WP7 will underpin the Fisheries Stock Depletion Assessment System (FSDAS). However, for such a dynamic domain like fisheries that is continuously evolving, we will need to provide the appropriate support for a successful implementation and service delivery of the FSDAS. In particular, for this task we need to support a collaborative editorial workflow that will allow Ontology editors to consult, validate and modify ontologies keeping track of all changes in a controlled and coherent manner. In this scenario, ontology editors are usually developing/maintaining ontologies collaboratively in a distributed environment. Additionally, ontology editors will have to perform many different tasks (e.g. create mappings, populate ontologies) that require to select the appropriate ontology. The registry is crucial component in the infrastructure to aid in the selection of appropriate ontologies and to support the editorial workflow: first, changes have to be monitored and captured from the ontology editor. Those changes should be formally represented and stored in Oyster which is in charge of the management of the different versions of the ontology. Also, using the registry functionalities, those changes will be searched and retrieved by the visualization components to show e.g. the differences between versions of the ontology and also to provide different views to ontology editors (depending on their role) where they can see the state of those changes. Finally, the registry will be in charge of the propagation of those changes to the distributed copies of the same ontology. This Plug-in provides access to the required registry services.

Benefits/Advantages of using Oyster

- Oyster provides a shared access to the registry services.
- Oyster allows users to customize the behaviour of the registry.

Data sets

FSDAS ontologies.

A usage example aligning to WP7 case study.

For the management of ontologies provenance and statistics (UC-9) this component will provide access to ontologies information including changes, use statistics and ontology statistics.

3.15. RaDON Plugin

3.15.1. Functional Description

The purpose of the RaDON plug-ins is to deal with inconsistency and incoherence occurring in networked ontologies. Specifically, RaDON provides two plugins to deal with a single ontology or an ontology network. In the plugin of "Repair a Single Ontology", the following specific functionalities are provided:

- **Handle incoherence:** This functionality corresponds to the button of "Handle Incoherence" which can be activated if the ontology is incoherent. That is, there is at least one unsatisfiable concept in the ontology. All the minimal unsatisfiability-preserving subsets (MUPS) can be computed for each unsatisfiable concept;
- **Handle inconsistency:** This corresponds to the button of "Handle Inconsistency" which is activated if the ontology is inconsistent. That is, there is no model for the ontology. All the minimal inconsistent subsets (MIS) can be calculated;
- **Repair automatically:** This corresponds to the button of "Repair Automatically". If the button of "Repair Automatically" in this section is pressed, our algorithm will provide some axioms to be removed to keep the coherence of the ontology. This is based on the found MUPS or MIS. If some MUPS have been found, the minimal incoherence-preserving subsets (MIPS) will be computed automatically;
- **Repair manually:** This corresponds to the button of "Repair Manually". If this button is activated, a new dialog will be shown with the information of MIPS or MIS. The user could choose the axioms to be removed by them.

In the plugin of "Repair and Diagnose Ontology Network", the similar functionalities in the plugin above are given. The main difference is that this plugin is to repair and diagnose a mapping between two ontologies by assuming the two source ontologies are more reliable than the mapping itself.

3.15.2. User Documentation

In this section, we will show how to use the plug-ins of "Repair a Single Ontology" and "Repair and Diagnose Ontology Network". You can find some data sets for testing from the website of <http://radon.ontoware.org/downloads/debugDatasets.zip>.

How to use the plug-in of "Repair a Single Ontology"

1. The user interface for debugging and repairing a single ontology is a view of "Repair a Single Ontology".
2. **How to invoke our view?** To debug or repair an ontology, please right-click the ontology in the Ontology Navigator in NeOn Toolkit and then select the item of "Debug and Repair...". The view of "Repair a Single Ontology" will be activated. At the same time, the logical and physical URIs of this ontology will be shown. Besides, we also show some information about whether this ontology is inconsistent or incoherent and how many axioms and unsatisfiable concepts (see Fig. 1).

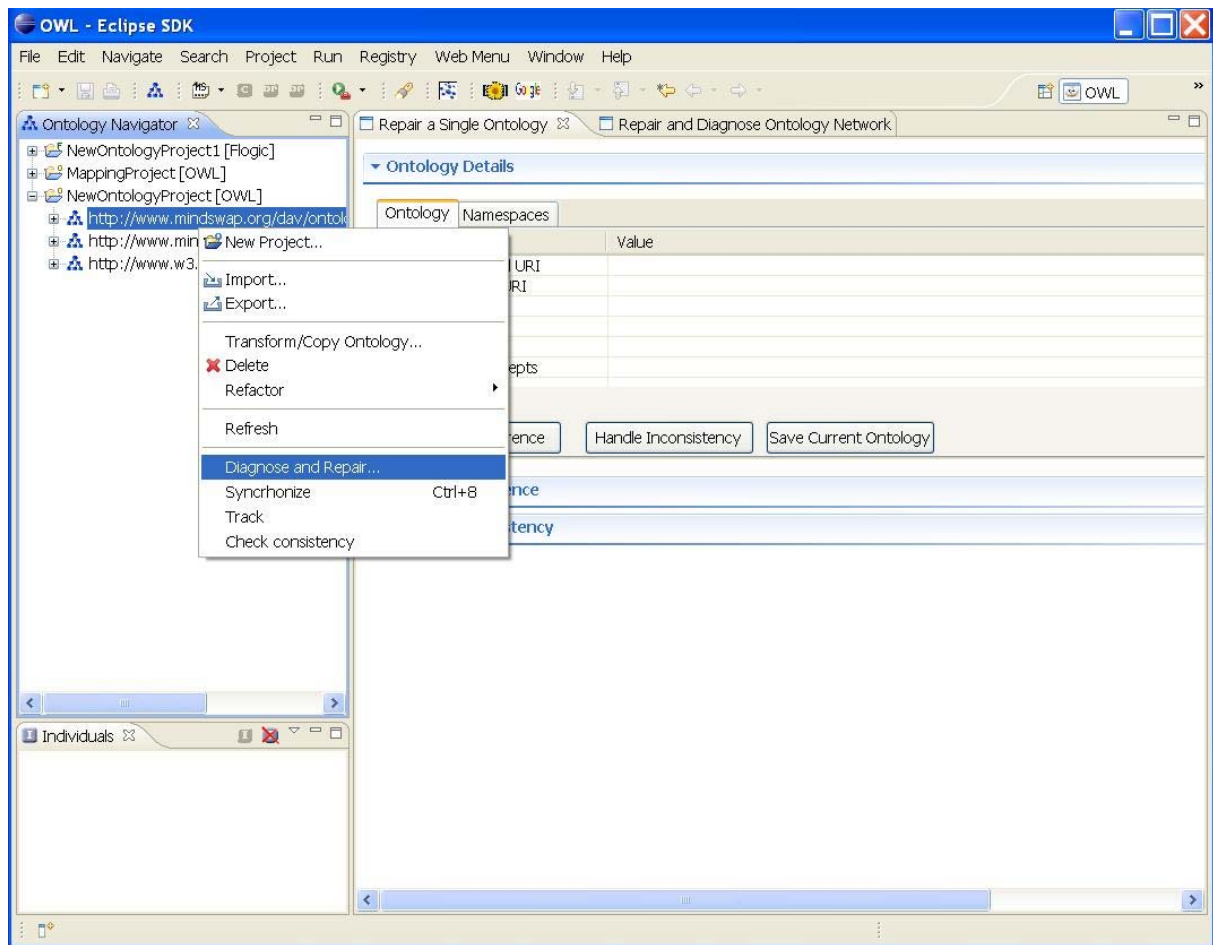


Figure 79: Invoking Diagnosis and Repair

3. To handle incoherence. If the test ontology is incoherent, all the unsatisfiable concepts will be listed when "Handle Incoherence" button is pressed. A user can compute all justifications for an unsatisfiable concept by pressing "+" before the concept (see Fig. 2). When the ontology is inconsistent, all the minimal inconsistent subsets will be computed if "Handle Inconsistency" is pressed.

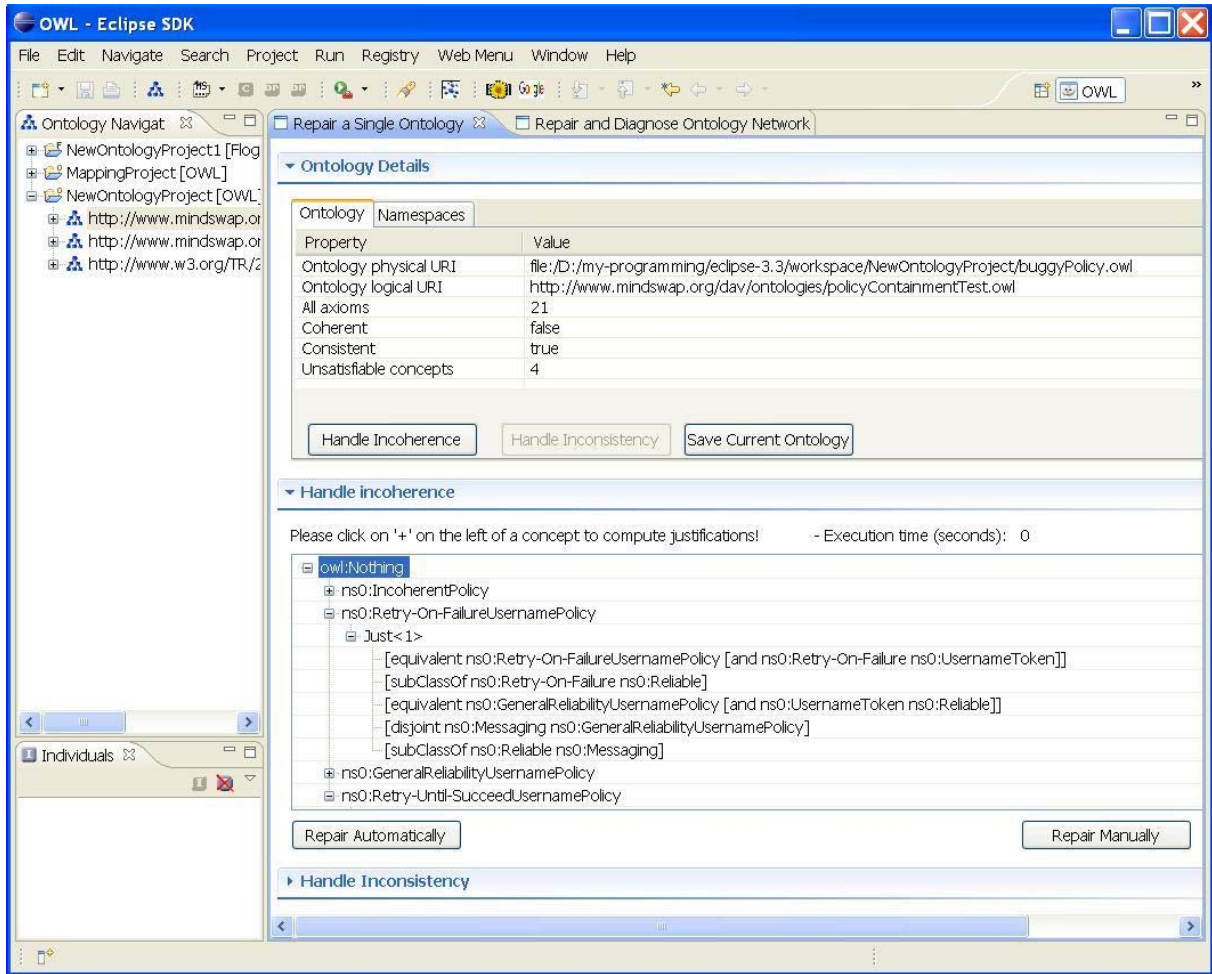


Figure 80: Handling incoherence

4. **To repair an ontology automatically.** A user can repair the ontology automatically by clicking the corresponding button. The proposed axioms to be removed to keep the coherence or consistency of the ontology will be shown in a new dialog. In this new dialog, the proposed axioms will be removed from the test ontology if the button of "OK" is pressed. Otherwise, no action is performed.

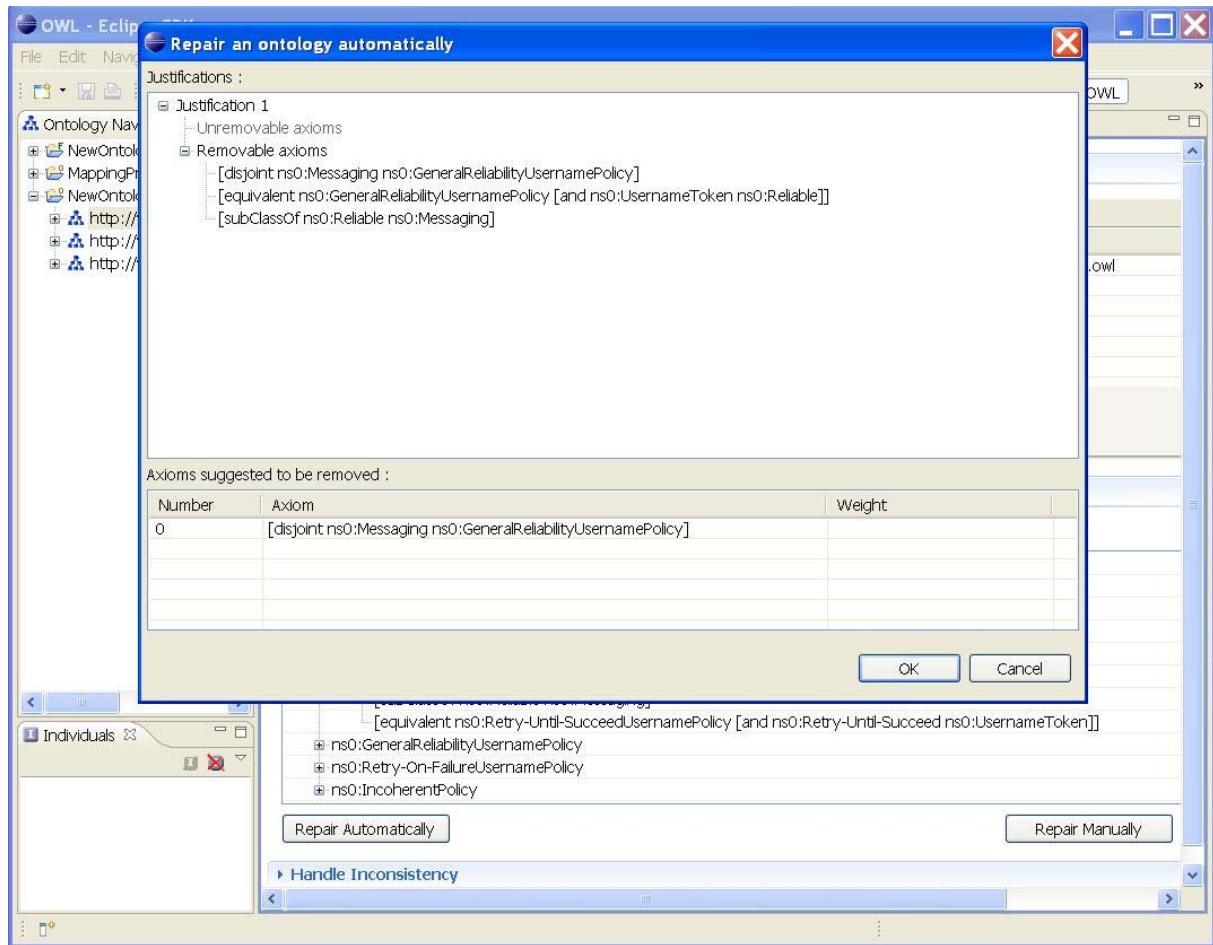


Figure 81: To repair an ontology automatically

5. To repair an ontology manually. If the button of "Repair Manually" is pressed, a new dialog will be displayed to the user. In this new dialog, the MIPS or MIS will be shown. The user could choose the axioms in the removable part to remove by clicking on the axiom. The selected axioms will be shown in the area of "Axioms to be removed". Also, the user could withdraw his/her decision by clicking "Unremove" label. After deciding the axioms to remove, the user can confirm his/her decision by clicking the button of "Confirm Removing" which will physically remove the selected axioms from the test ontology.

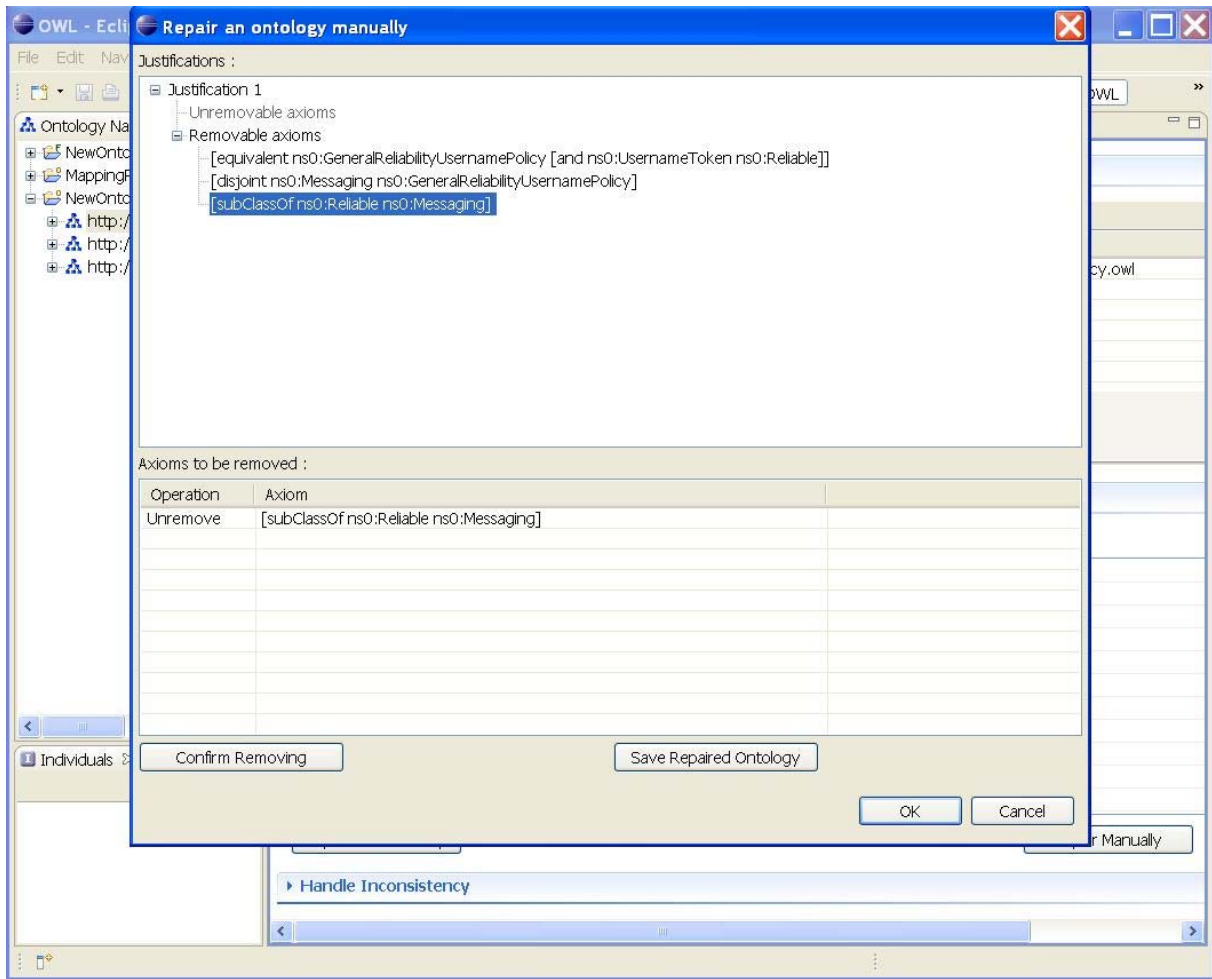


Figure 82: To repair an ontology manually

How to use the plugin of "Repair and Diagnose Ontology Network"

1. How to invoke the view. In NTK, Windows --> Show View --> Other... --> RaDON --> Repair and Diagnose Ontology Network.
2. Input. The input for this view includes: two source ontologies and the mapping between them, which consist of an ontology network. If the network is incoherent or inconsistent, the corresponding buttons to handle incoherence or inconsistency will be ready for use. Currently, we support the alignment format like INRIA format (example), Notation3 (example), Karlsruhe format (example) and Manheim format (example).

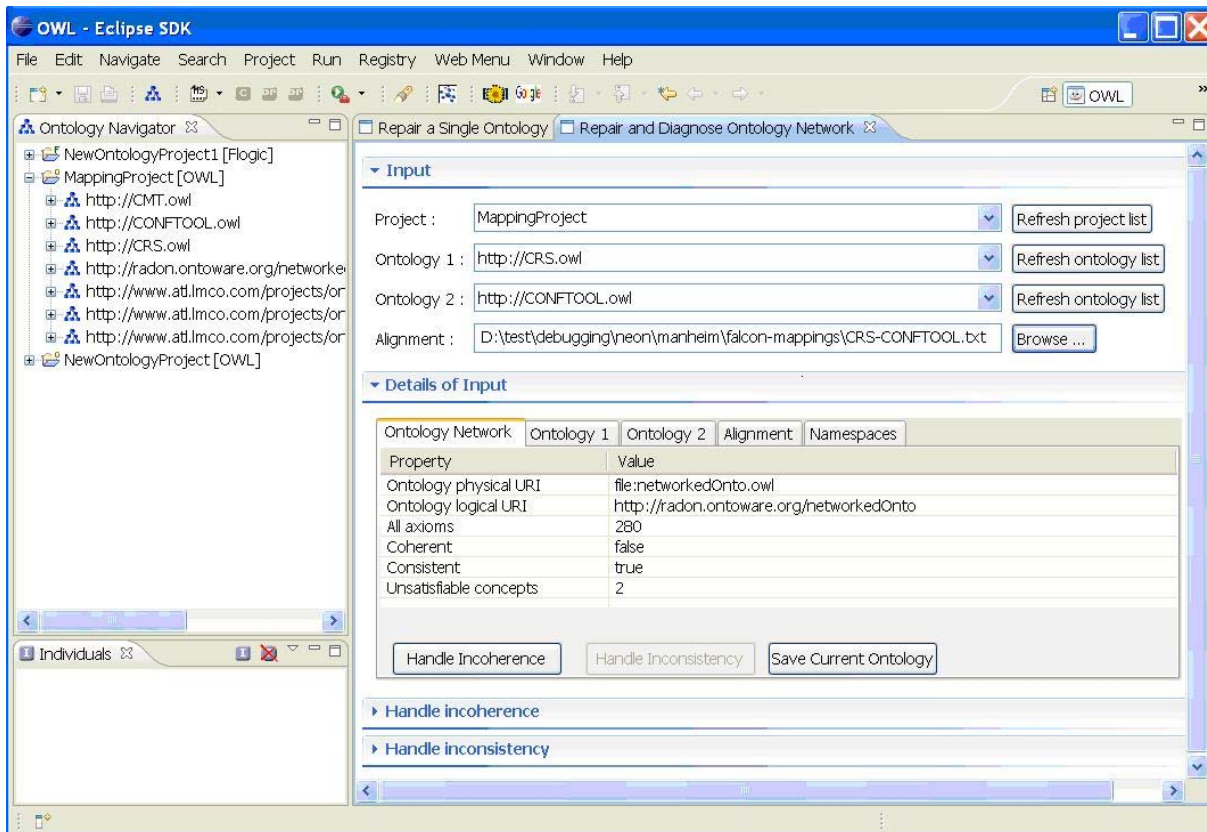


Figure 83: The input for an ontology network

3. Handle incoherence / inconsistency. Similar to the corresponding functionalities to deal with a single ontology.

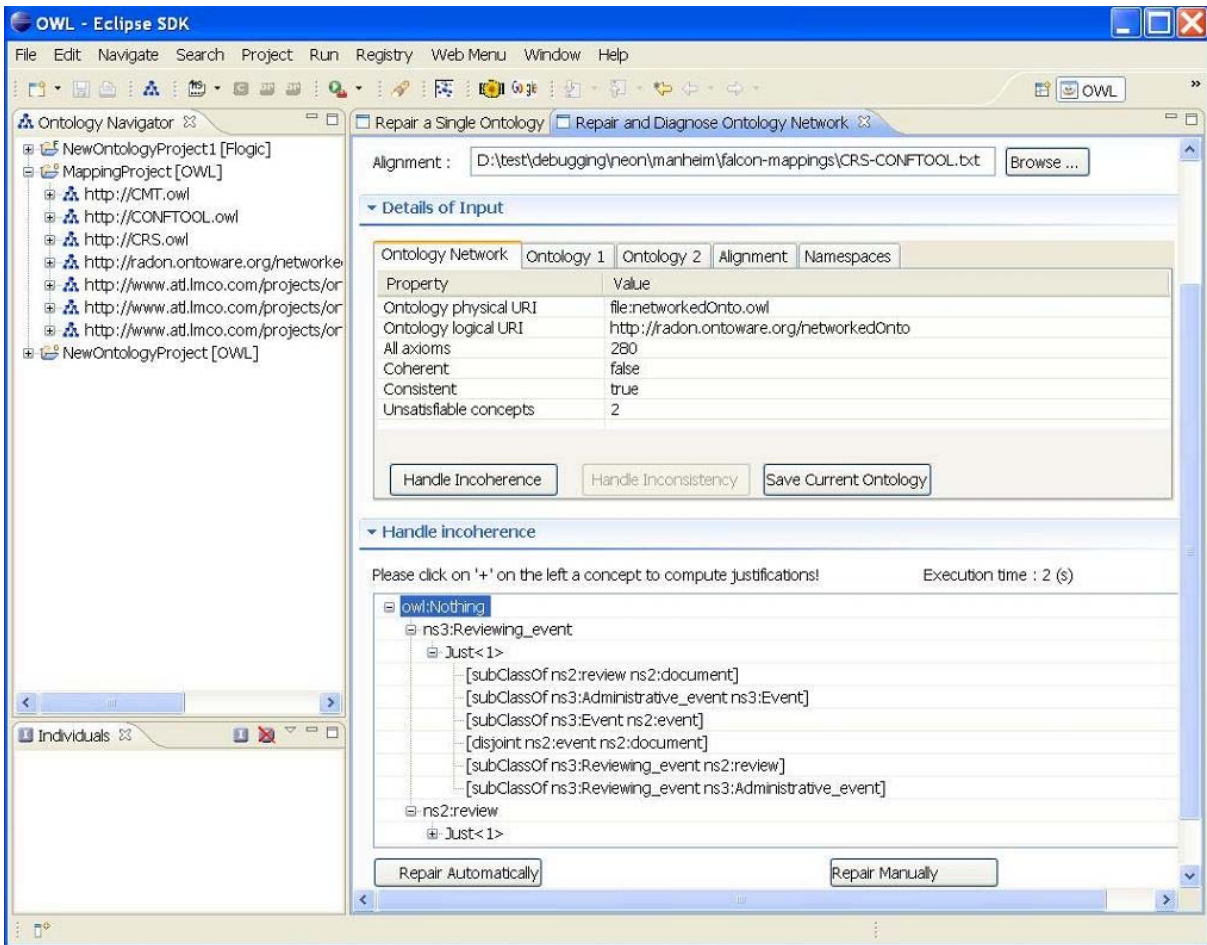


Figure 84: Compute the justifications for an ontology network

4. **Repair an ontology network automatically or manually.** This is quite similar to the corresponding functionalities when repairing a single ontology. The main difference is that the axioms in the unremovable part are in the two source ontologies and the axioms in the removable part belong to the mapping between the source ontologies. Additionally, the confidence values for each correspondence in the mapping will be shown.

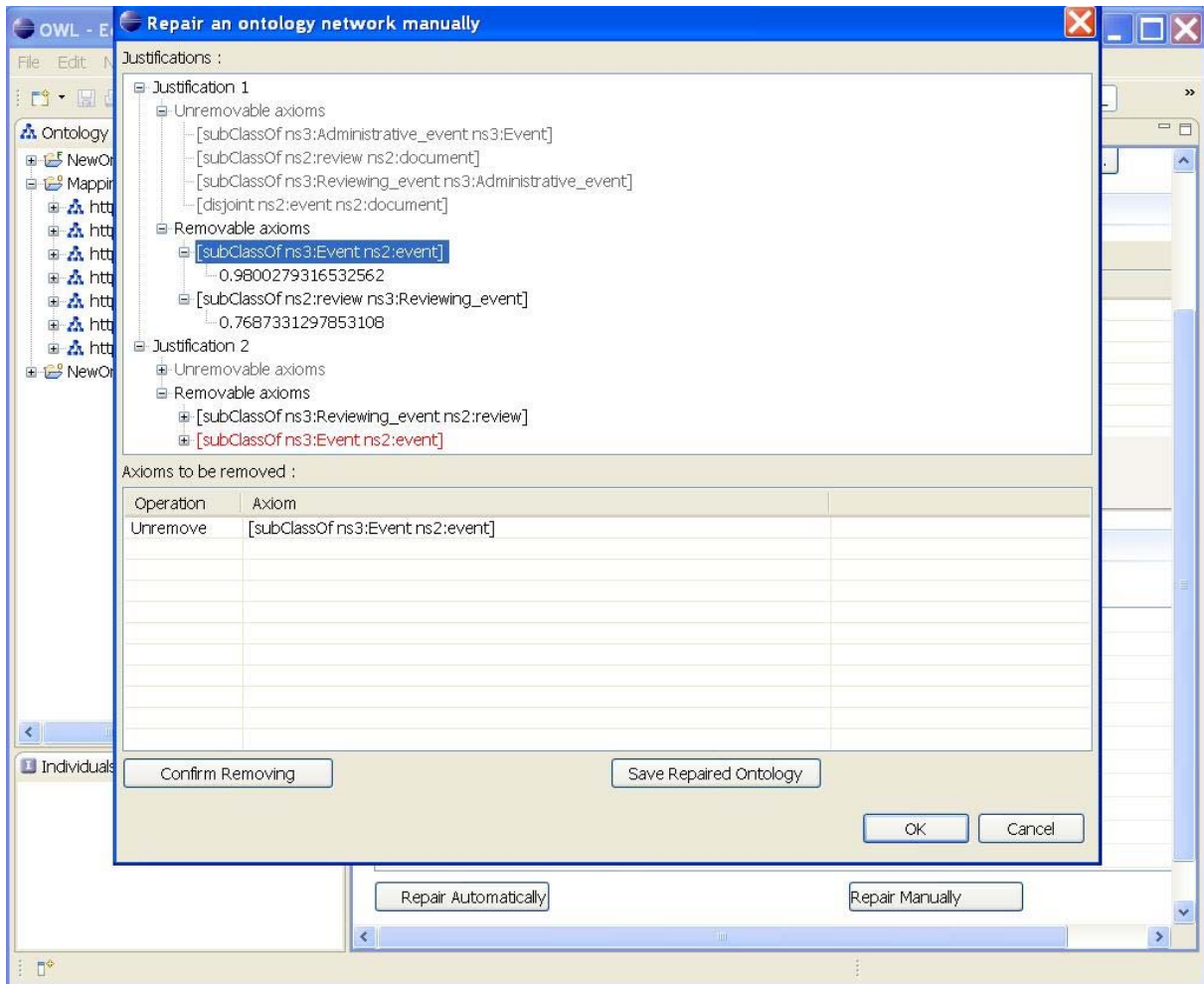


Figure 85: Repair an ontology network manually

3.15.3. Integration into the NeOn Toolkit

RaDON plugin is a view in the NeOn Toolkit and can be invoked by right click an ontology in the tree-like ontology navigator and choosing "Debug and Repair ... ". This view includes three parts:

- Ontology information part: This part provides information about ontology URI, size of this ontology, whether this ontology is incoherent or inconsistent, etc.
- Debugging part: Here one can debug the ontology if it is inconsistent or incoherent.
- Repair part: User could repair the inconsistent or incoherent ontology automatically or manually according to the debugging results.
- Eclipse Extension Points
 - org.eclipse.ui
 - org.eclipse.core.runtime
 - org.eclipse.ui.forms
 - org.eclipse.ui.ide
- NeOn Toolkit extension points
 - com.ontoprise.ontostudio.gui
 - com.ontoprise.ontostudio.owl.gui
 - com.ontoprise.ontostudio.datamodel
 - datamodelBase
 - datamodel
 - util

3.15.4. Intended Usage in the Case Studies

The RaDON plugin has already been applied in the WP7 FAO case study in the context of diagnosing and repairing automatically learned/extracted ontologies. Results of these applications have been reported in NeOn Deliverable D1.2.2.

3.16. SPARQL Plugin

3.16.1. Functional Description

The SPARQL Plugin allows a user to do conjunctive queries. It receives an OWL ontology and a query expressed using SPARQL syntax as inputs and outputs the answers in a table. This reasoning task is performed using the inference engine in KAON2.

3.16.2. User Documentation

1. Open the view of "SPARQL Query": In the menu of "Windows", please choose Show view / Other / SPARQL Query.

2. How to use it: First of all, choose a project and an ontology in the project. The default namespace of this ontology will be shown automatically. Then the user can input a SPARQL query in the text area. Take the following as an example:

PREFIX wine: <<http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>>

PREFIX rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

SELECT ?x

WHERE { ?x wine:hasFlavor wine:Strong . ?x wine:locatedIn wine:NewZealandRegion .

?x wine:hasSugar wine:Dry }

After pressing "Execute Query", the result of the query will be shown in the table if the syntax of the input query is correct. Otherwise, a warning dialog will be shown to the user. Finally, the user could save the query results to a local file.

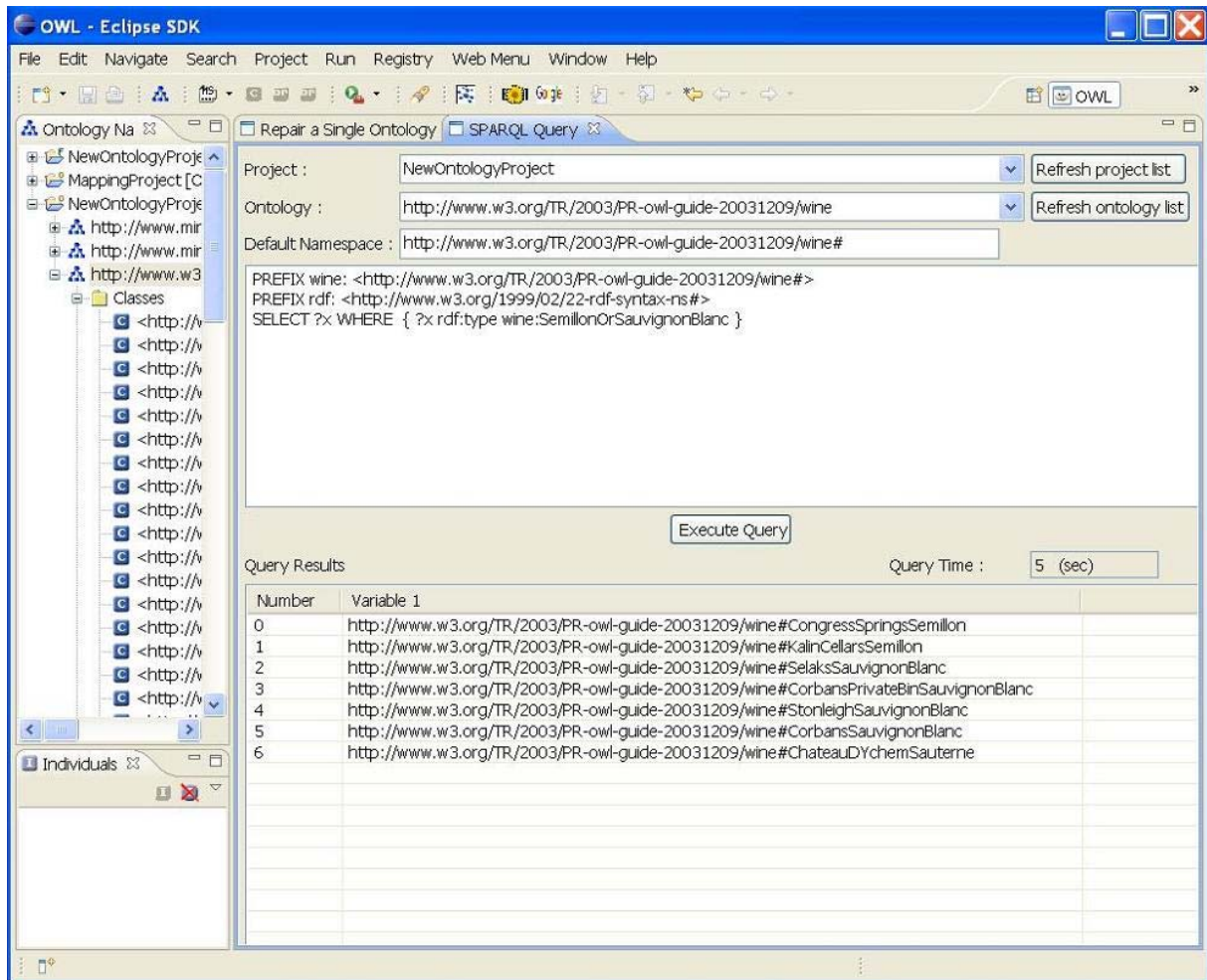


Figure 86: SPARQL Query View

3.16.3. Integration into the NeOn Toolkit

SPARQL query has been integrated with NeOn toolkit as a separate view to do queries about those OWL ontologies in the Ontology Navigator.

- Eclipse Extension Points
 - org.eclipse.ui
 - org.eclipse.core.runtime
 - org.eclipse.ui.forms
 - org.eclipse.ui.ide
- NeOn Toolkit extension points
 - com.ontoprise.ontostudio.gui
 - com.ontoprise.ontostudio.owl.gui
 - com.ontoprise.ontostudio.datamodel
 - datamodelBase
 - datamodel
 - util

3.16.4. Intended Usage in the Case Studies

The SPARQL plugin can be applied in the FAO case study to do queries.

3.17. Text2Onto

3.17.1. Functional Description

Text2Onto is an ontology learning framework which has been developed to support the acquisition of ontologies from textual documents. Like its predecessor, TextToOnto, it provides an extensible set of methods for learning atomic classes, class subsumption and instantiation as well as object properties and disjointness axioms.

3.17.2. User Documentation

Technical reports, papers, presentations and demo videos for the standard version of Text2Onto are available from <http://www.aifb.uni-karlsruhe.de/WBS/jvo/text2onto/>. Detailed information with regards to this plugin can be found in NeOn D3.8.1.

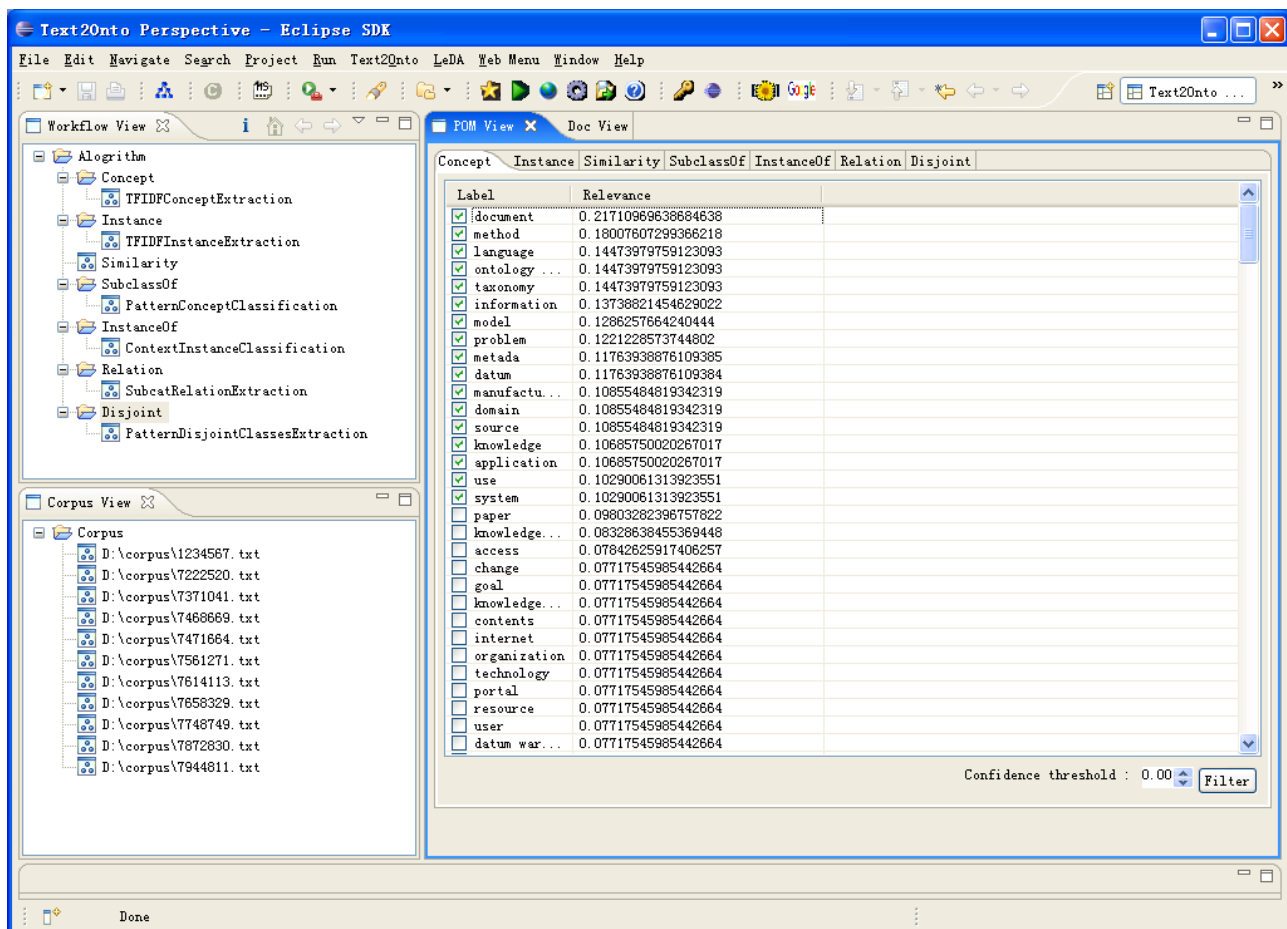


Figure 87: Text2Onto User Interface

The graphical user interface of the plugin is composed of different views for the configuration of the ontology learning process and the presentation of the results.

Workflow view

The upper left corner contains the workflow view, which is used to set up the ontology learning workflow. By right-clicking on the individual ontology learning tasks (e.g. "Concept" for concept extraction), the user can select one or more methods for each type of ontology element she wants to extract from the corpus.

Corpus view

In the bottom left corner, the user will find a corpus view, which allows her to set up a corpus that is a collection of text documents from which the ontology will be generated. The doc view (see hidden tab on the right) is used to display previews of selected documents. Text2Onto is able to analyse documents in plain text, PDF (Windows only) and HTML format. However, a manual conversion into purely textual format is highly recommended for efficiency reasons.

POM view

The POM view on the right shows the results of the most recently initiated ontology learning process. The view contains several tabs -- one for each type of ontology element that was extracted from the corpus -- showing a tabular listing of individual results. By clicking on the column headers the user can sort the ontology elements according to their associated labels or confidence values.

Preferences

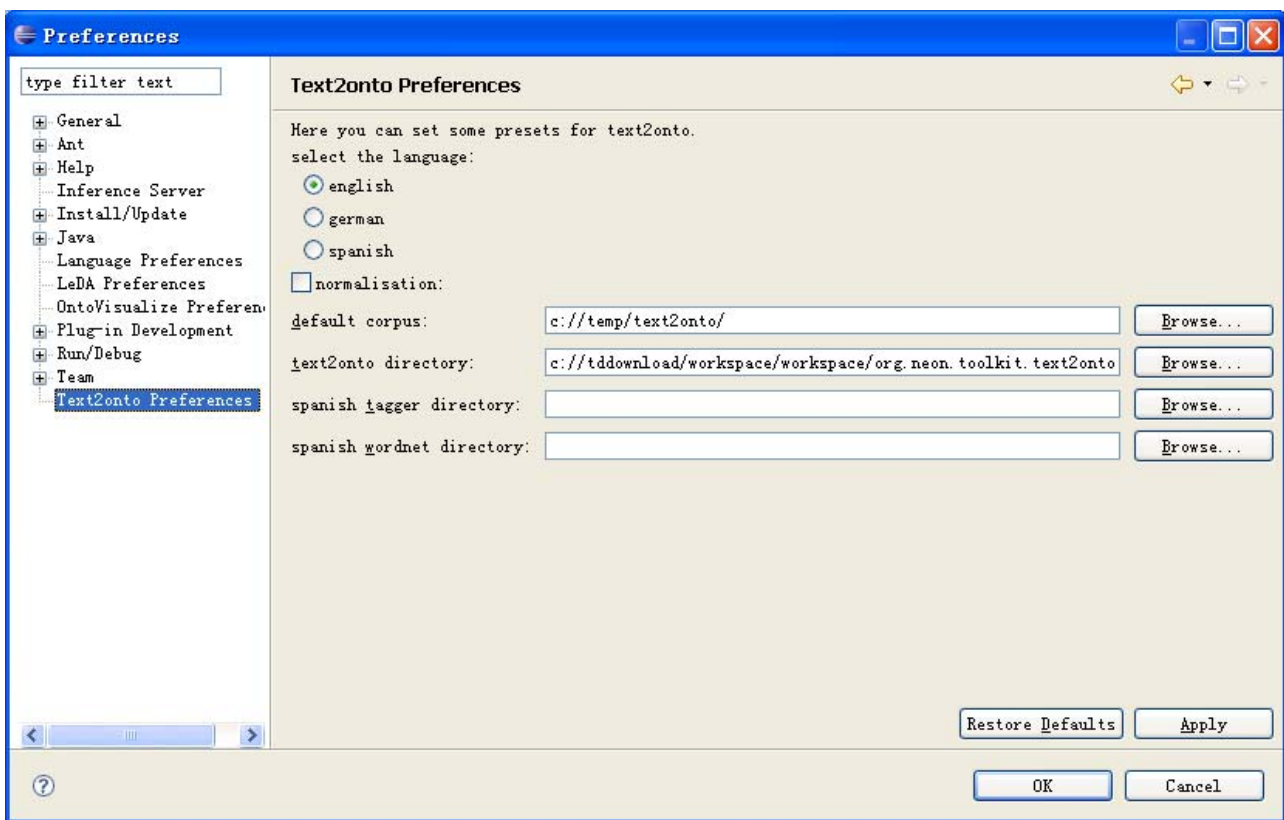


Figure 88: Text2Onto Preferences

The preference page, which is accessible from the main menu of on the top of the Text2Onto perspective ("Window" -> "Preferences..." -> "Text2Onto Preferences") replaces the original configuration file of Text2Onto's API. It allows for setting the following parameters:

- **Language:** The language of the documents to be analysed. Text2Onto provides full support for learning ontologies from English and Spanish corpora as well as partial support for ontology extraction from German texts. For details with respect to the Spanish version of Text2Onto please refer to SEKT D3.3.3.
- **Normalization:** If this parameter is selected Text2Onto will normalize all confidence values to an interval of 0.0 to 1.0.

- **Default corpus:** The default directory for populating the ontology learning corpus.
- **Spanish tagger directory:** The part-of-speech tagger to be used for the analysis of Spanish documents. In the current version of Text2Onto this parameter is expected to point to the TreeTagger (<http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/>) installation directory.
- **Spanish WordNet directory:** In case the language is set to Spanish, this path should refer to a licensed version of Spanish WordNet (<http://www.lsi.upc.edu/~nlp/web/index.php>).

3.17.3. Integration into the NeOn Toolkit

Eclipse Extension Points

- [org.eclipse.ui.perspectives](#)
- [org.eclipse.ui.views](#)
- [org.eclipse.ui.actionSets](#)
- [org.eclipse.core.runtime.preferences](#)
- [org.eclipse.ui.preferencePages](#)

3.17.4. Intended Usage in the Case Studies

3.18. Watson

The Text2Onto plugin is to be used in WP7 for ontology learning experiments.

The Watson plugin for knowledge reuse allows the developer of an ontology to query Watson to find existing descriptions of selected entities in the edited ontology, and to reuse (i.e. integrate) these descriptions into the currently edited ontology. It is an easy and integrated way to reuse knowledge that has been published on the (Semantic) Web.

The Watson plugin is a tool that aims to facilitate large scale knowledge reuse by extending an ontology editor with the features of the Watson Semantic Web search engine. With this plugin, it is possible to discover, inspect and reuse ontology statements originating from various online ontologies directly in the ontology engineering environment. This document quickly describes the usage of the Watson plugin to build or extend an ontology within the NeOn toolkit ontology editor.

The two demo videos available at http://watson.kmi.open.ac.uk/editor_plugins.html provide useful additions to this guide.

3.18.1. User Documentation

Installing the Watson Plugin: As for most of the NeOn Toolkit plugins, the Watson plugin can be installed using the automatic software update feature of the Toolkit (Menu *Help, Software Updates, Find and Install*). You can then "Search for new features to install" using the NeOn toolkit update site. The Watson Plugin can be found under the "Knowledge Reuse" category.

Alternatively, if you encounter problems with the above procedure, you can also install the Watson plugin by copying the corresponding jar files (downloadable from http://watson.kmi.open.ac.uk/editor_plugins.html) in the *Plugin* directory of your NeOn Toolkit installation.

Basics: The NeOn toolkit is able to consider ontologies in two different formats, F-Logic and OWL. A version of the Watson Plugin exists for both languages, but only the OWL version is maintained.

The Watson plugin works in the context of an existing ontology. First, a new OWL project has to be created ("New Project"), and a new ontology should be added to the project ("New Ontology" or import an existing one). Essentially, the Watson plugin is a feature integrated within the NeOn toolkit that can be asked for information about a particular entity (statements) from any other ontology of the Semantic Web. It can be triggered through the "right-click" menu of a particular entity.

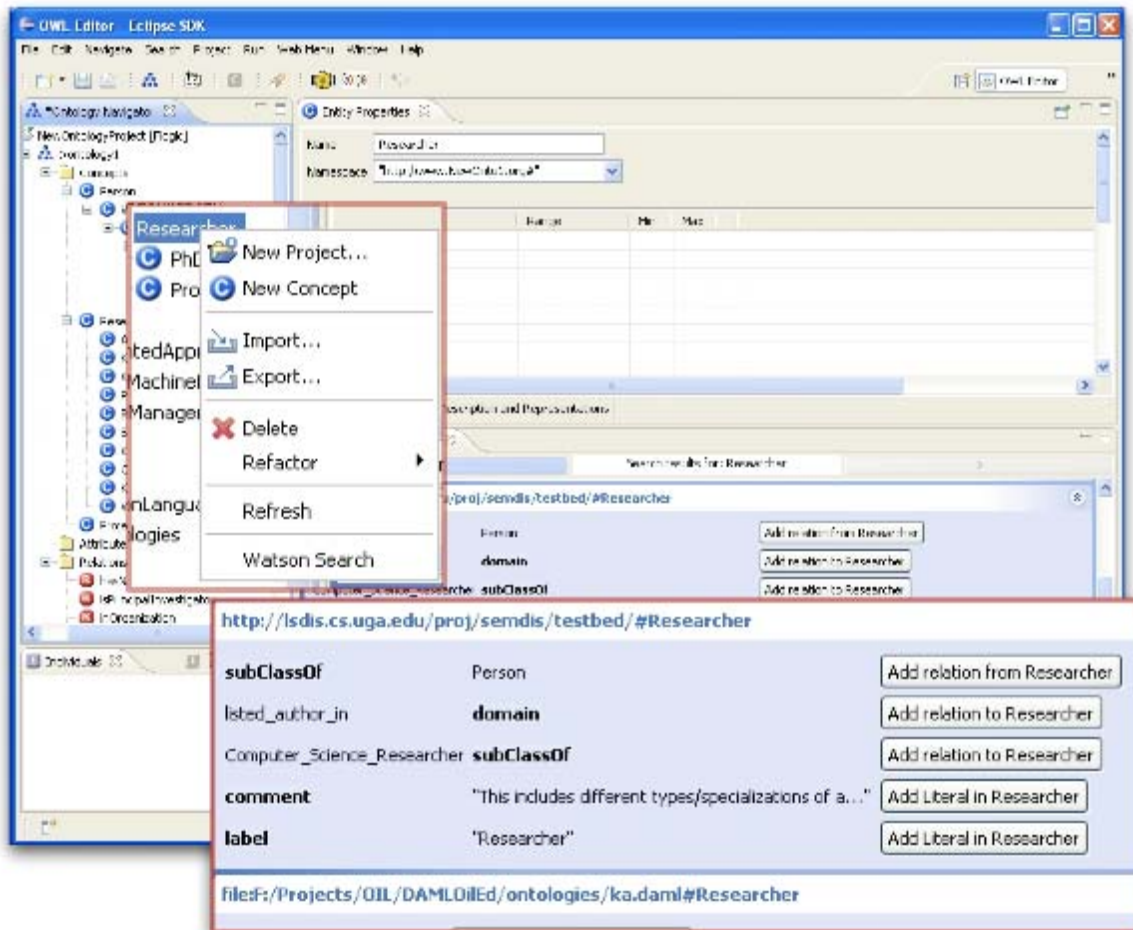


Figure 89: Watson Search

For example, in the figure above, a class *Researcher* has been created. Clicking on the "Watson Search" item of its right-click menu will trigger a search for any statement on the semantic web concerning a class named *researcher*. The result of the search for a particular entity is displayed in a separate view. The list of entities that have been found is shown, with for each entity, the statements they are associated with. In the example of the figure, several classes *Researcher* have been found in various semantic web ontologies. Among these classes, one is a subclass of *Person*, has for subclass *Computer_Science_Researcher*, has for label *Researcher*, is the domain of the property *listed_author_in*, etc. Each of the statements retrieved thanks to Watson can be imported into the ontology using the button next to it. The statement will then be attached to the original entity (the one that triggered the search) in the currently built ontology. In our example, clicking on the button "Add relation from Researcher" next to the statement "**subClassOf** Person" will: 1- create the class *Person* in the current ontology if it does not already exist and 2- make *Researcher* a

subclass of *Person*. In the same way, importing the statement "listed_author_in **domain**" will create the relation *listed_author_in* with *Researcher* as domain.

3.18.2. Integration into the NeOn Toolkit

The Watson plugin can be considered as both a *GUI component* (provides a view) and an *engineering component* (provides ways to create ontologies by reuse) for the NeOn toolkit, and which relies on the *infrastructure component* provided by Watson. It makes use of two main extension points from the eclipse GUI components, one for creating the item on the right-click menu of entities (*org.eclipse.ui.popupMenus*) and one for creating the view to display the results (*org.eclipse.ui.views*). It heavily relies on the NeOn Toolkit datamodel component, as well as on the Watson client API (see http://watson.kmi.open.ac.uk/-WS_and_API.html).

3.18.3. Intended Usage in the Case Studies

The Watson plugin is useful in any scenario where an ontology is being created or extended, and where the reuse of other existing ontologies is desirable.

4. Conclusions and Future Work

In this deliverable we have described the second set of plugins developed for the NeOn Toolkit by the partners of the NeOn project. Existing plugins have been updated to be compatible with the latest version of the NeOn Toolkit and extended with new functionalities. New plugins have been developed to address previously unaddressed ontology lifecycle activities. Further, the new plugins have been developed with an improved Quality Assurance process and include user documentation integrated into the NeOn Toolkit using the Eclipse help infrastructure.

Based on observations in the case studies, there is still need for functionalities that are currently not yet provided. Therefore, on the one hand we will develop updated and improved versions of the existing plugins. On the other hand, a number of new plugins are already planned. These include for in particular plugins to support the methodology (such as the gOntt plugin), collaboration, and runtime-oriented plugins, e.g. to provide end-user oriented search. The deliverable describing the available plugins will be updated again in the final year of the project.

5. References

- NeOnD531 Mari Carmen Suárez-Figueroa et al: D5.3.1 NeOn Development Process and Ontology Life Cycle, NeOn project deliverable, August 2007.
- NeOnD532 Mari Carmen Suárez-Figueroa et al: D5.3.2 Revision and Extension of the NeOn Development Process and Ontology Life Cycle, NeOn project deliverable, December 2008.
- NeOnD621 Walter Waterfeld et al: D6.2.1: Specification of NeOn reference architecture & NeOn APIs, NeOn project deliverable, February 2007.
- NeOnD613 Jose Manuel Gómez-Pérez et al.: D6.3.2 Evaluation of NeOn Components against Requirements, NeOn project deliverable, January 2008.
- NeOnD6101 Peter Haase et al: D6.10.1: D6.10.1 Realization of core engineering components for the NeOn toolkit, NeOn project deliverable, February 2008.
- NeOnD742 Yimin Wang, et al.: D 7.4.2 Prototype System for Managing the Fishery Ontologies Lifecycle, NeOn project deliverable, February 2008.
- NeOnD821 Jose Manuel Gómez-Pérez: D 8.2.1 Software architecture for the NeOn pharmaceutical case studies, NeOn project deliverable, February 2007.