NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D6.10.1 Realization of core engineering components for the NeOn Toolkit

Deliverable Co-ordinator: **Peter Haase**

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

Other Authors: Milan Agatonovic, Carlos Buil-Aranda, Mathieu d'Aquin, Mauricio Espinoza, Michael Gesmann, Qiu Ji, Chan Leduc, Klaas Dellschaft, Raul Palma, Jacopo Penazzi, Johanna Völker, Yimin Wang

This deliverable describes the first 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/2008/D6.10.1/v1.0	Date due:	February 29, 2008
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	February 29, 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:

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

Work package participants

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

UKARL

ONTO

UPM

OU

CNR

USFD

UKOB

CNR

INRIA

iSOCO

SAG

Change Log

Version	Date	Amended by	Changes
0.1	03-12-2007	Peter Haase	Creation
0.2	08-01-2008	Peter Haase	Updated list of plugins
0.3	31-01-2008	Peter Haase	Introduction, conclusion
0.4	07-02-2008	Peter Haase	Updated list of plugins
1.0	19-03-2008	Peter Haase	Addressing reviewers comments
1.1	25-03-2008	Peter Haase	Final QA

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 this deliverable we first describe the plugin development process for plugins that are developed by partners within the consortium. This includes aspects of source code management, distribution of plugins, quality assurance, and licensing.

As the main contribution, we then describe the first set of plugins that have been implemented by NeOn partners for the NeOn Toolkit. These plugins implement and integrate functionalities that have been developed in the technical workpackages and support a wide range of ontology lifecycle activities according to the NeOn methodology. Many of the plugins are already used in the case study prototypes. Further, this initial set of plugins plays an important role as seed plugins to leverage the adoption of the NeOn Toolkit and the development of new plugins by the community.

Table of Contents

List of figures

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.

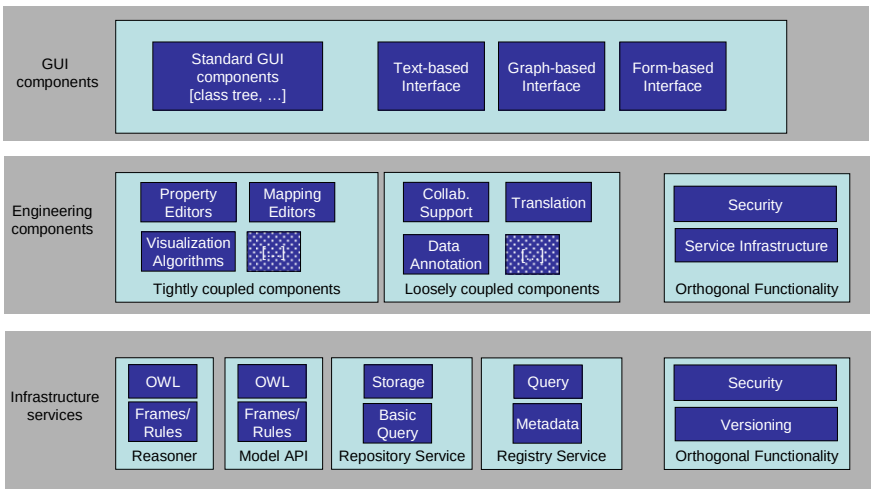


Figure 1: NeOn Architecture

Eclipse as integration platform

The NeOn architecture relies on the architectural concepts of the Eclipse platform[1]. 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".

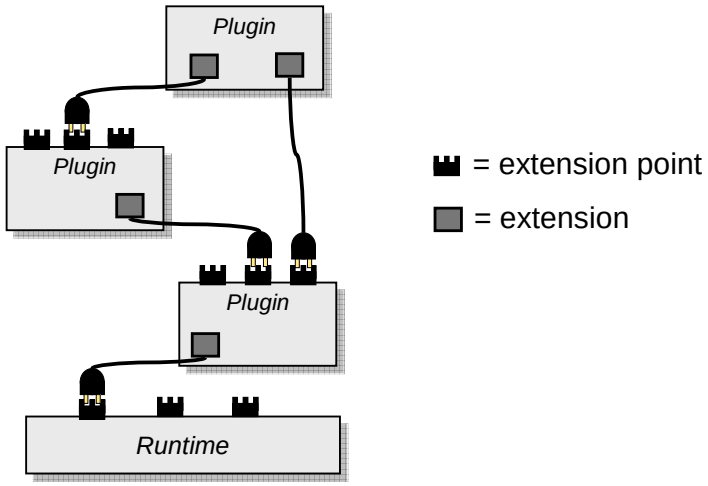


Figure 2: Plugin Concept of Eclipse

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/

1.2 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 (c.f. [NeOnD741] for WP7, and [NeOnD821] for WP8) 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.

In the future, we will publish best practices as guidelines for plugin developers in the NeOn community.

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 80 projects are hosted at Ontoware, around 10 of them being plugins for the NeOn toolkit.

2.2 Licensing

In principle, every plugin provider is entitled to distribute his or her plugin under the license he thinks is most appropriate. Yet, to avoid an unnecessarily cluttered situation, we recommend the Eclipse Public License (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 copyleft 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 a user documentation. The plugin wiki is integrated into the NeOn Toolkit portal (c.f. Figure 3: NeOn Plugin Wiki).

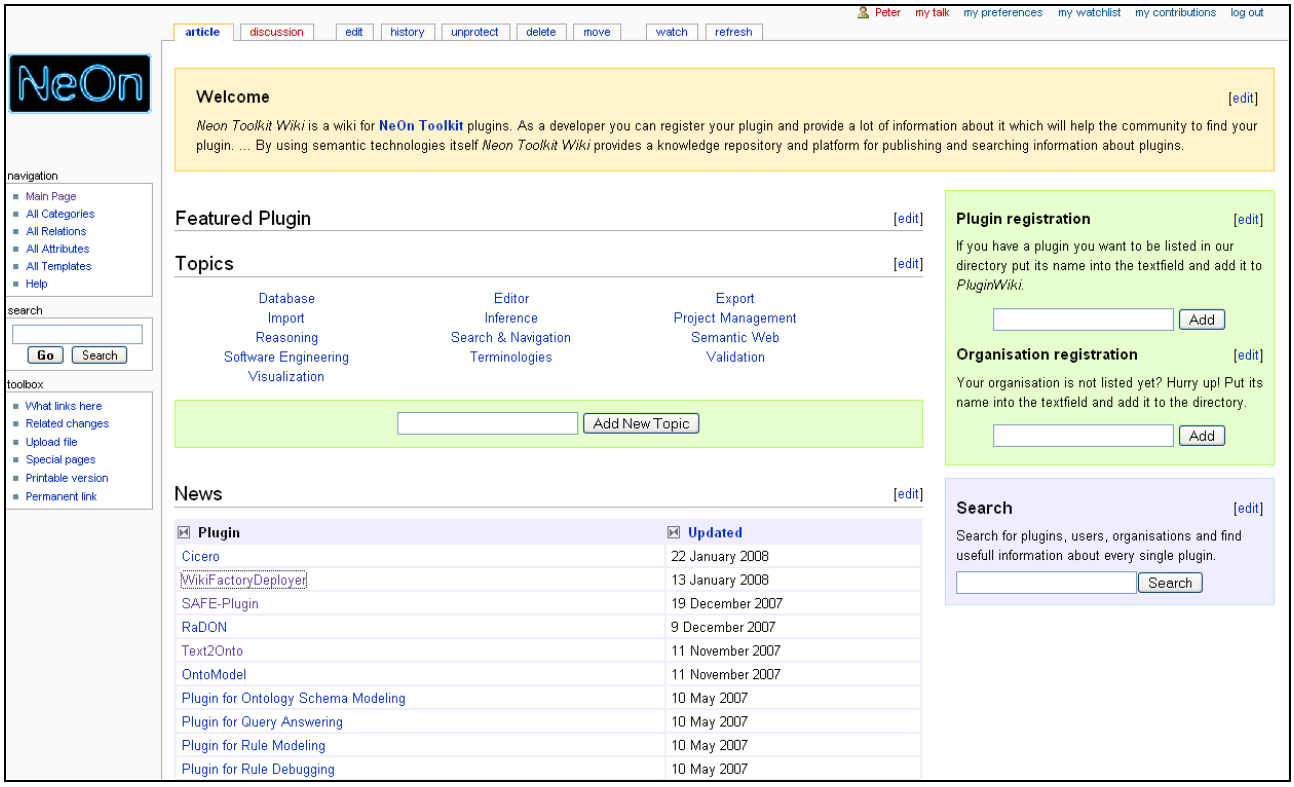


Figure 3: NeOn Plugin Wiki

In addition, we also maintain an internal wiki (restricted to the NeOn consortium) for plugins that are not officially published yet. This internal wiki 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 Quality Assurance and Bug Management

In principle, the responsibility for quality assurance lies with the developer of the plugin. Yet, in order to ensure a professional appearance, we have established basic measures that every plugin needs to meet before it is officially published: Every plugin is tested by at least two users from within the NeOn consortium with respect to:

- deployment procedure
- functionality and ease of use according to user documentation
- completeness and correctness of the description in the plugin wiki

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 3: NeOn Toolkit Bugzilla).

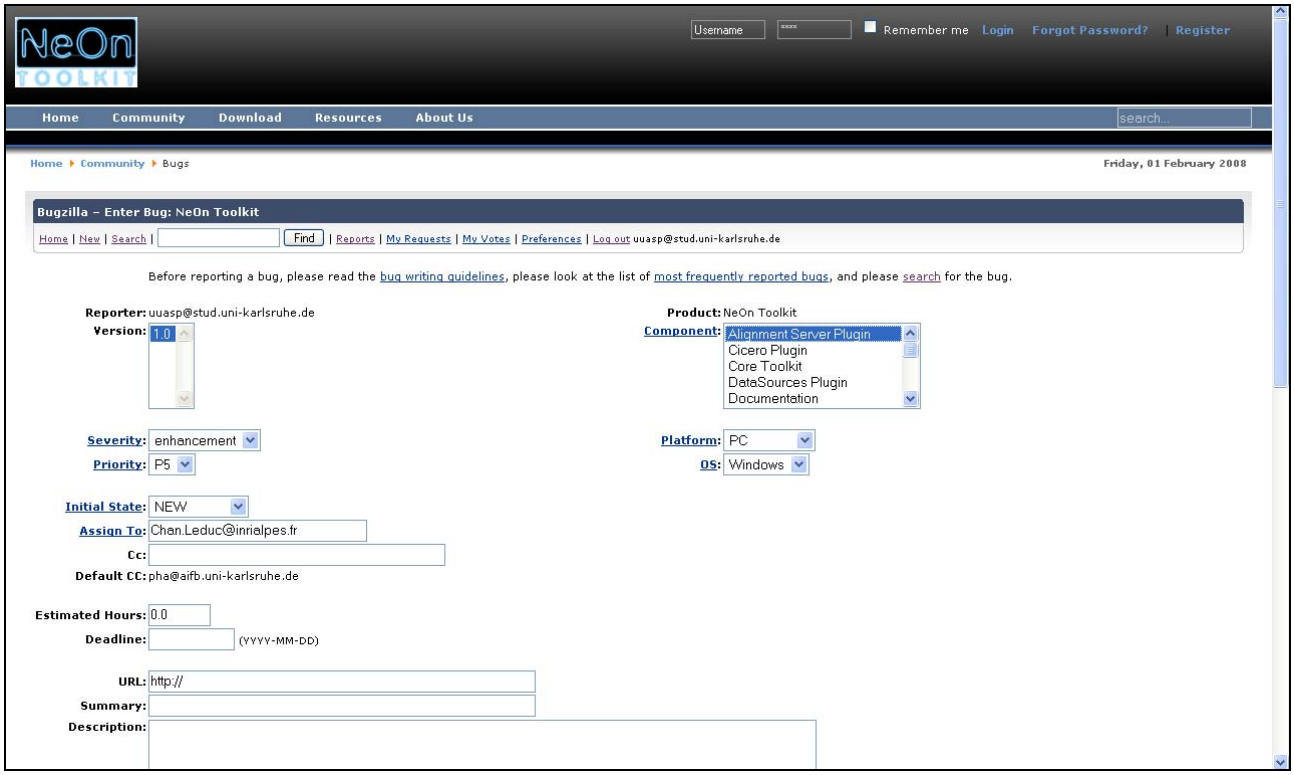


Figure 3: NeOn Toolkit Bugzilla

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

2.5 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 mechanism[2], a mechanism that allows deploying and updating new features.

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

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, 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 4: 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.

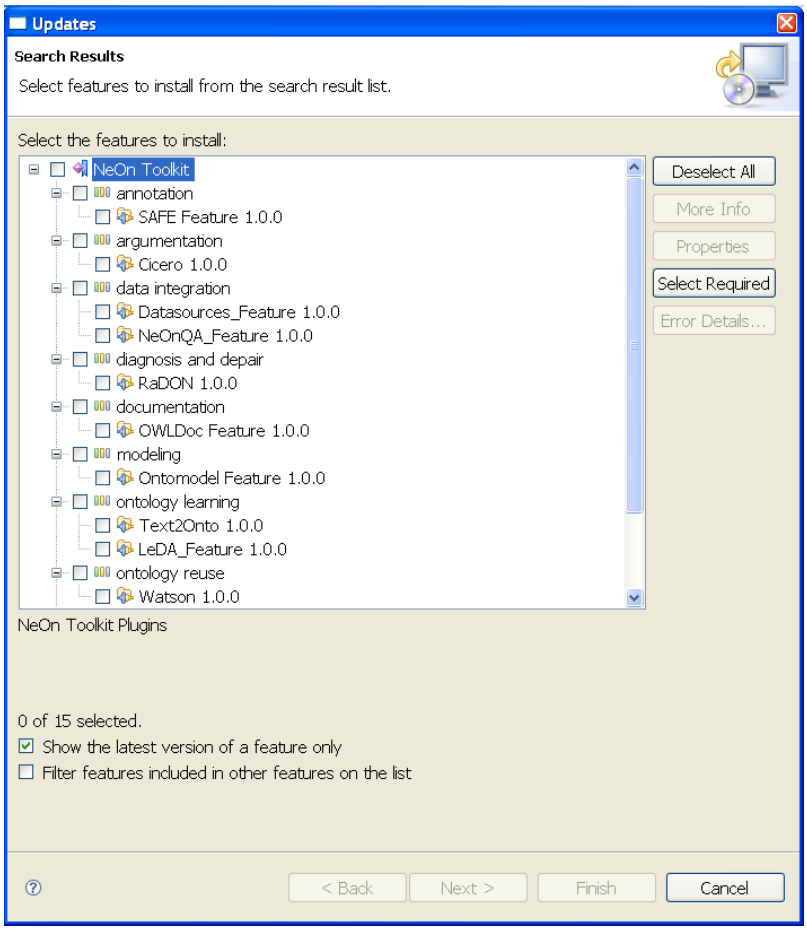


Figure 4: NeOn Toolkit Update Site

3 Plugin Descriptions

In this chapter, we provide descriptions of the individual plugins. After a general overview, we provide for each plugin a detailed functional description, a 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.

Plugin Name	Lifecycle activity	Developed by	Organization
AlignmentServer	Ontology Alignment / Aligning Ontology Mapping Ontology Matching	Chan Leduc	INRIA
Cicero	Ontology Documentation	Klaas Dellschaft Hendrik Engelbrecht José Monte	UKO-LD
DataSources	Non Ontological Resource Reuse	Michael Gesmann	Software AG
EcoreImport	Non Ontological Resource Reengineering Ontology Learning Non Ontological Resource Reuse	Peter Haase	AIFB
LabelTranslator	Ontology Localization	Mauricio Espinoza	UPM
LeDA	Ontology Learning	Johanna Völker	AIFB
Neonqa	Question Answering	YWang	UKARL
OWLDoc	Ontology Documentation	Raonne Barbosa-Vargas Oscar Munoz-Garcia Oscar Corcho	UPM
OntoModel	Ontology Conceptualization Ontology Enrichment Ontology Extension Ontology Formalization Ontology Modification Reestructuring Ontology Specialization Ontology Update	Tian Bai Saartje Brockmans Peter Haase	AIFB
OntologyRelatonshipVis	Ontology Visualization	Carlos Buil-Aranda	ISOCO
Oyster	Ontology Searching Ontology Selection	Raul Palma	UPM AIFB
RaDON	Ontology Diagnosis Ontology Repair	Qiu Ji	AIFB
SAFE	Ontology Population	Hamish Cunningham	USFD
Text2Onto	Ontology Learning	Johanna Völker	AIFB
Watson for Knowledge Reuse	Ontology Reuse Ontology Searching Ontology Selection	Mathieu d'Aquin Fouad Zablith	OU
WikiFactoryDeployer	Ontology Documentation	Jacopo Penazzi	CNR

XML Mapping	Ontology Population Non Ontological Resource Reengineering Ontology Learning Non Ontological Resource Reuse	Michael Gesmann	Software AG

3.2 Alignment Server

3.2.1 Functional Description

The Alignment Server Plugin allows automatically computing and managing ontology alignments. More precisely, the Alignment Server Plugin offers the following functionalities:

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

3.2.2 User Documentation

To activate the NeOn Alignment Plugin from the NeOnToolkit Menu, click on the "Align" menu or the button on Toolbar, a view "Alignment Server" for the plugin will be opened. The figure below shows the available functions of the plugin.

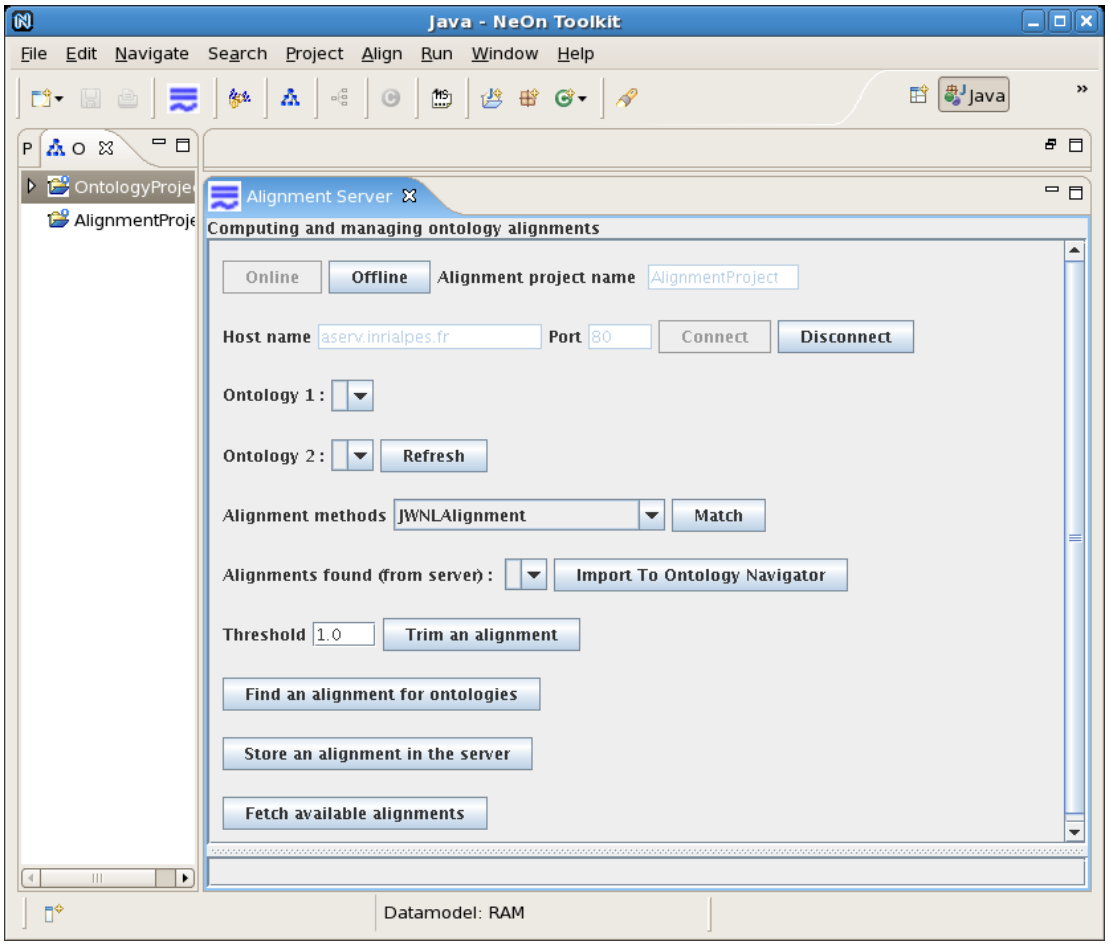


Figure 5: Alignment Server Plugin

The NeOn Alignment 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 the server while the online mode offers additionally mechanisms to store and reuse alignments. In the offline mode, which is activated by clicking on button "Offline", the NeOn Alignment 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 imported to Ontology Navigator as OWL ontologies. In the online mode, which is activated by clicking on button "Online", the NeOn Alignment Plugin provides all functions from the Alignment Server. Resulting alignments are stored on the server and exported to Ontology Navigator as OWL ontologies. This allows NeOn Toolkit users, with help of the Ontology Editor, to use, share or edit alignments.

Stepwise example.

Show the Ontology Navigator view.

From the menu bar of the NeOnToolkit, perform "Window -> Show view -> OntoStudio-> Ontology Navigator", and create a project, for example "OntologyProject", in this view.

Open working OWL ontologies.

From the menu bar of the NeOnToolkit, open working OWL ontologies in the project "OntologyProject" created above by "File -> Import... -> OntoStudio -> FileSystem Import Wizard".

Activate the Neon Alignment Plugin.

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

Define an alignment project.

From the view for the plug-in, give a name in the field "Alignment Project Name". This name will be used for an Ontology Navigator project which includes all alignments created by the NeOn Alignment plug-in later on.

Activate the on-line mode.

From the view for the plugin, activate the on-line mode by clicking on the button "Online" and the button "Connect" becomes enable.

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, we can see the buttons "Refresh", "Match", "Import to Ontology Navigator", "Trim", "Find an alignment...", "Store an alignment..." and "Fetch available alignments..." become enable. In particular, a list of available alignment methods is visible at "Alignment methods". In addition, an Ontology Navigator project, whose name was defined above, is automatically created for alignments.

Matching two ontologies.

First we must fetch the working 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 "Alignment methods". Click on "Match" to match these two ontologies with the method chosen. The resulting alignment will be added to the list "Alignments found". See the figure below.

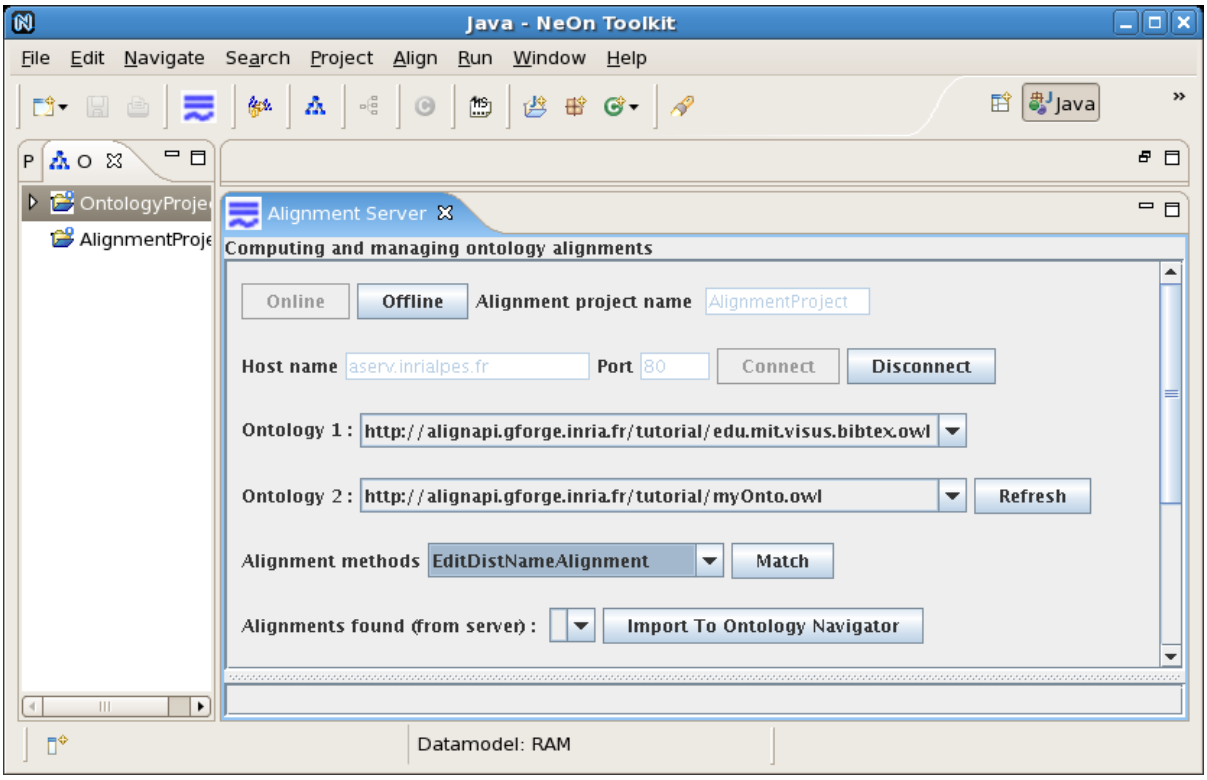


Figure 6: Alignment Server: Matching two ontologies

Importing an alignment.

Choose an alignment from the list "Alignments found" and click on "Import to Ontology Navigator". The alignment chosen will be imported as an OWL ontology to the alignment project which was created in Ontology Navigator. Users can navigate the OWL alignment imported in Ontology Navigator. See the figure below.

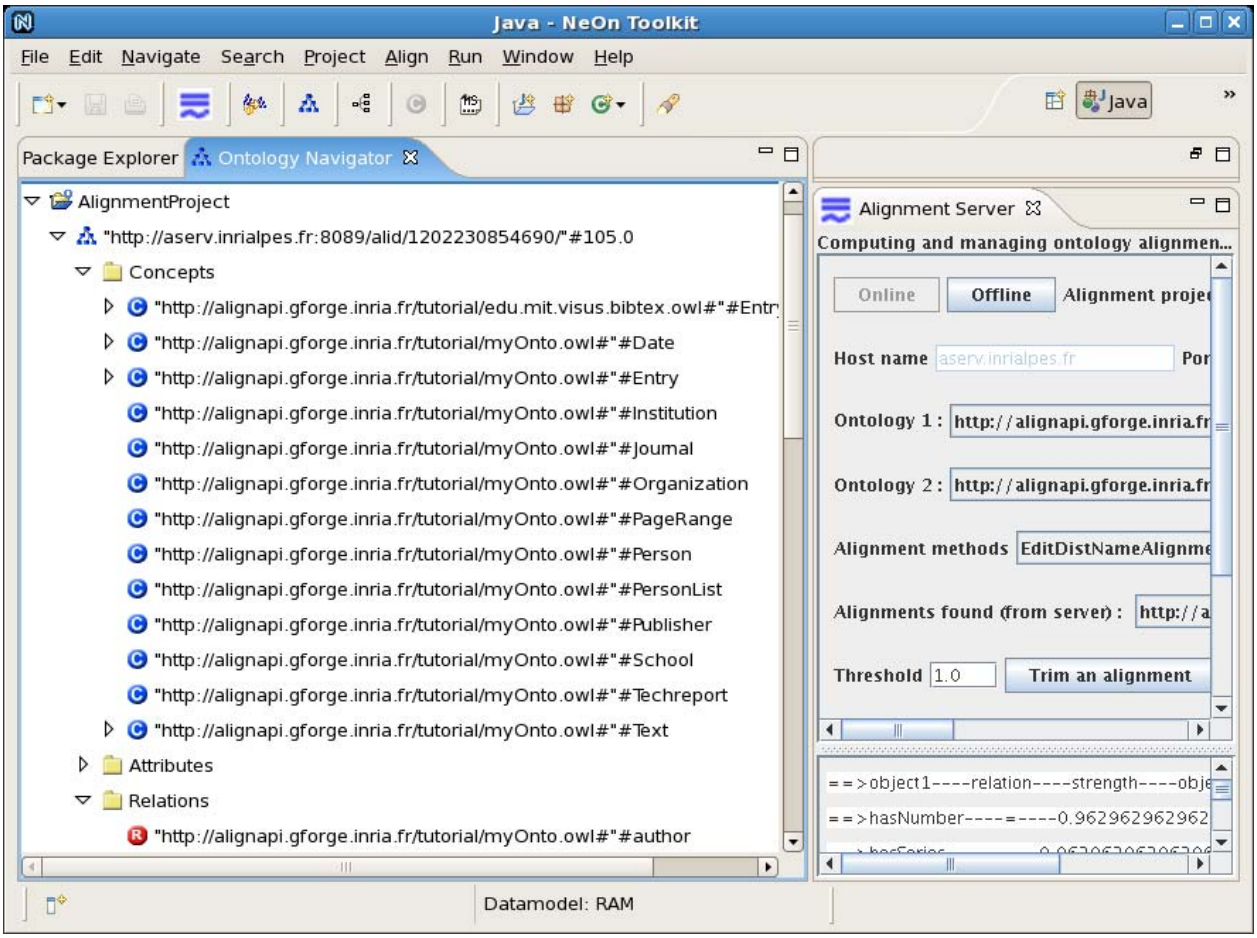


Figure 7: Alignment Server: Importing an alignment

Fetch available alignments.

This function allows users to obtain a list of all alignments available on the Alignment Server. See the figure below.

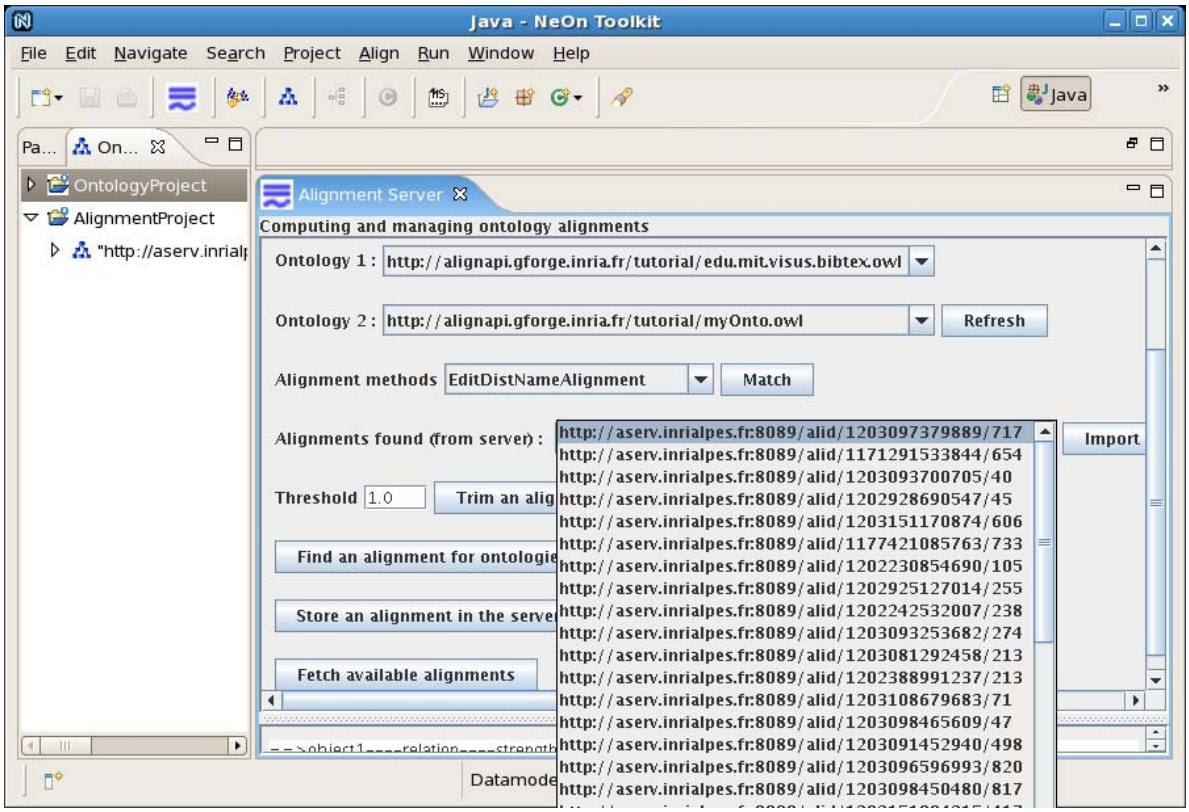


Figure 8: Alignment Server: Fetching Alignments

3.2.3 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

3.2.4 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.

3.3 Cicero

3.3.1 Functional Description

The main purpose of the Cicero plugin for the NeOn toolkit will be to establish the provenance links between discussions and the corresponding ontology elements and ontology changes. Furthermore, it will provide search mechanisms for finding discussions that are relevant for specific ontology elements. For the final plugin that is due in M30, it is planned to support the following functionality:

- Create issues from within the NeOn toolkit.
- Establish provenance links between issues created in the toolkit and the corresponding ontology elements.
- Allow for easily establishing provenance links between ontology changes made in the toolkit and solution proposals in Cicero.
- Allow for searching and filtering discussions:
    - Full text search in issues, solution proposals and arguments.
    - Filtering results for specific authors and dates.
    - Filtering results for the status of the discussion (e. g. running or decided).
    - Filtering results for discussions with a provenance link to specific ontology elements.

However, in a first version that will be released in M24, the plugin will allow for doing the following:

- Annotate an ontology with the URL of the corresponding project in an installation of Cicero.
- Open the project in Cicero in the internal browser window of the NeOn toolkit.

3.3.2 User Documentation

As shown in the picture below, the Cicero-plugin provides a new (sub-)property page called **Argumentation Settings** for an (F-Logic) ontology element.

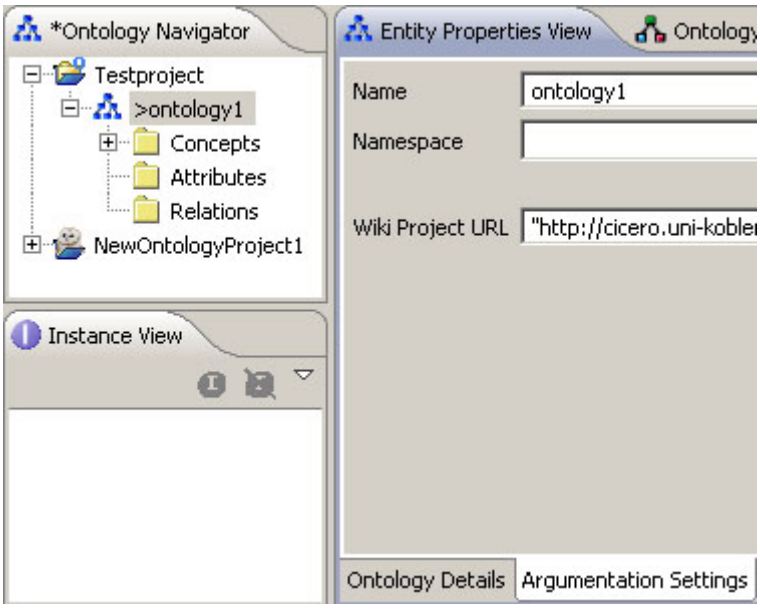


Figure 9: Cicero – Argumentation Settings

This interface mainly provides a text field **Wiki Project URL**. The user can here enter a (valid) URL of the project in a Cicero-Wiki-installation in which the currently selected ontology is discussed. The value entered in the text field can then be persistently saved in the datamodel.

By right-clicking on a concept in an (F-Logic) ontology, the respective popup menu will appear which will be extended by the entry **Show Argumentation**. In the current implementation of the Cicero-plugin the URL entered in the **Wiki Project URL**-text field of the **Argumentation Settings**-property page will be opened in a (internal) web-browser.

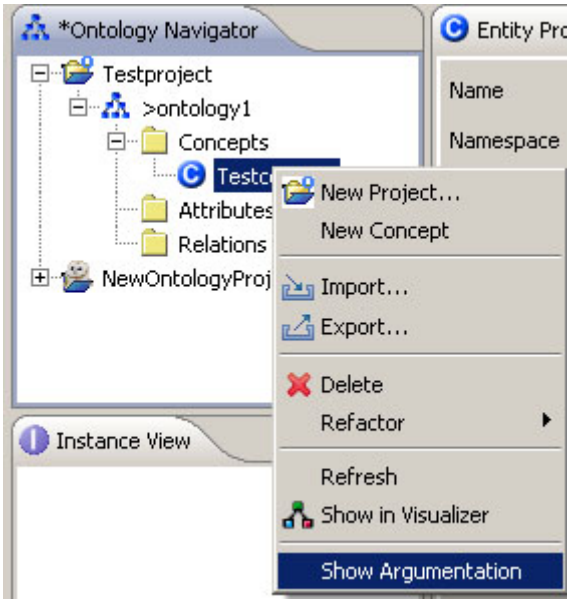


Figure 10: Cicero – Showing argumentation details

3.3.3 Integration into the NeOn Toolkit

The following extension points are used within the plugin:

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

The plugin itself consists of the following packages:

- **org.neontoolkit.cicero**
    - contains the **Activator.java**, which controls the plug-in life cycle.
- **org.neontoolkit.cicero.actions**
    - contains the **ConceptTreeElementArgumentationAction.java**, which extends the popup-menu of a concept tree element in the ontology navigator tree with the entry "Show argumentation".
- **org.neontoolkit.properties**
    - contains the **OntologyPropertyContributor.java**, which provides a further property page for a module tree element. There the user can adjust the respective argumentation settings.
- **org.neontoolkit.cicero.statements**
    - contains the **ArgumentationStatement.java**, which allows to add/remove the argumentation settings to/from the data model.

Note that the current version of the plugin (0.1) only supports **F-Logic ontologies**.

3.3.4 Intended Usage in the Case Studies

Currently, there are ongoing talks about using Cicero and the plugin in WP7 in the context of AGROVOC maintenance as well as in WP8 for supporting the definition of invoice models.

3.4 Datasources Plugin

3.4.1 Functional Description

This plugin extends the navigation tree by a node named "Datasources". For ontologies that had been generated from data sources like e.g. SQL databases, XML schema files, or web services, this features displays the data sources' parameters like database name, user, password, location of the imported XML schema file, wsdl file of imported web service. Also, users can change those parameters that are specific for a runtime environment and might change when deploying an application from one environment to another one, e.g. from testing to production.

This plugin depends on the representation of data source parameters in ontologies. It is only usable for those ontologies, which had been created as FLogic ontologies via the import wizard. Some of the data sources mentioned above are only available in the extended Toolkit.

3.4.2 User Documentation

The tree node will be shown automatically, when opening the tree. Clicking on the tree node opens a list of child nodes that represent the data sources. Of course this list may be empty. Clicking on one of the displayed nodes shows detail information in the EntityPropertiesView.

3.4.3 Integration into the NeOn Toolkit

This feature extends the TreeProvider and offers a new EntityProperties view.

- Relies or depends on the following plugins
    - com.ontoprise.ontostudio.datamodel
    - com.ontoprise.ontostudio.gui
    - org.eclipse.core.resources
    - org.eclipse.core.runtime
    - org.eclipse.ui
    - org.eclipse.ui.views
    - org.neontoolkit.gui
- uses Neon-Toolkit / OntoStudio extension points
    - org.neontoolkit.gui.entityProperties
    - org.neontoolkit.gui.extendableTreeProvider

3.4.4 Intended Usage in the Case Studies

The plugin may be used for the runtime applications in the case studies that make use of the FLogic integration capabilities of the NeOn Toolkit.

3.5 EcoreImport

3.5.1 Functional Description

The plugin allows to import EMF models (filetype ecore). EMF (Eclipse Metamodeling Framework) is the Eclipse implementation of MOF, the Metaobject Facility. The Import is based on a mapping between the constructs of EMF and the OWL metamodel.

Being able to import EMF models into ontologies builds several bridges into the area of Model Driven Development. Many tools support Ecore as a standard export format for models. For example, UML models can be exported as Ecore by most state-of-the-art modelling tools. Additionally, there exist model transformation tools that allow to map other models (e.g. in XML Schema, Java) into Ecore, which then in turn can be imported into an ontology.

3.5.2 User Documentation

The usage is very simple, as the plugin is a standard Import wizard.

- Select the ontology project into which the model should be imported
- Select "Import ... Metamodels ... Ecore Model"
- Select the ecore file to be imported

3.5.3 Integration into the NeOn Toolkit

The plugin is realized as an Import Plugin by extending the Eclipse extension point org.eclipse.ui.importWizards. The ImportWizard is implemented by extending the AbstractImportWizard of the NeOn Toolkit.

The plugin has dependencies from the following NeOn Toolkit plugins:

- com.ontoprise.ontostudio.datamodel
- com.ontoprise.ontostudio.gui
- com.ontoprise.ontostudio.io

Additionally, the plugin has dependencies from the following Eclipse plugins:

- org.eclipse.ui
- org.eclipse.core.runtime
- org.eclipse.core.resources
- org.eclipse.ui.ide
- org.eclipse.emf.
- org.eclipse.emf.common
- org.eclipse.emf.ecore
- org.eclipse.emf.ecore.xmi

3.5.4 Intended Usage in the Case Studies

The plugin may be relevant in WP8, where a variety of existing models, e.g. in the form of UML models or XML Schema, need to be reused. Using existing tools (such as the EMF plugin) the models can be converted to ecore and then imported as an ontology.

3.6 LabelTranslator

3.6.1 Functional Description

LabelTranslator takes as input an ontology whose labels are described in a source natural language (e.g.. English) and obtains the most probable translation of each ontology label in a target natural language (eg. Spanish). LabelTranslator extends the NeOn Toolkit for supporting the translation of ontological labels using relevant information obtained from different lexical resources.

Main Charactecristic:

- LabelTranslator support the translation of ontological labels. In this sense, a label can represent a class name, a property name, or relation name.
- Linguistic information to be considered (i.e. that LabelTranslator will manage) will be:
    - Label
    - Gloss or Definition
    - Source of knowledge
- LabelTranslator will look for the relevant information in the lexical resources that have been implemented.
    - EWN databases
    - Web Resources
        - GoogleTranslate
        - Wiktionary
        - IATE
        - Babelfish
        - FreeTranslation

3.6.2 User Documentation

The Labeltranslator.zip file contains the following:

- org.neon.toolkit.labeltranslator_1.0.2.jar.
    - This file constitute LabelTranslator plug-in.
- The software deliverable contains three EWN databases needed for proper operation of the LabelTranslator.

- ewn_en.db
- ewn_es.db
- ewn_de.db

How to install:

1. Unzip the downloaded zip-file.

2. Copy org.neon.toolkit.labeltranslator_1.0.2.jar into the plugins folder of your NeOn ToolKit installation, e.g. at C:\NeOn Toolkit 1.1\plugins

3. Copy the three ewn databases into the database folder of your NeOn ToolKit installation, e.g. at C:\NeOn Toolkit 1.1\database

4. Start NeOn ToolKit version 1.1.

5. In the Window menu choose "Preferences... > Language Preferences" and changes the order of the languages, which will be used by LabelTranslator plug-in.

- NOTE: Remember that the current version of LabelTranslator only manages three languages (English, Spanish, and German). These languages must be the first three languages of the list of preferences languages. (seeFigure 11: Languages Preferences used by LabelTranslator plug-in). For default, LabelTranslator supposes that the first language of the list of preferences is the source language and the second language is the target language.

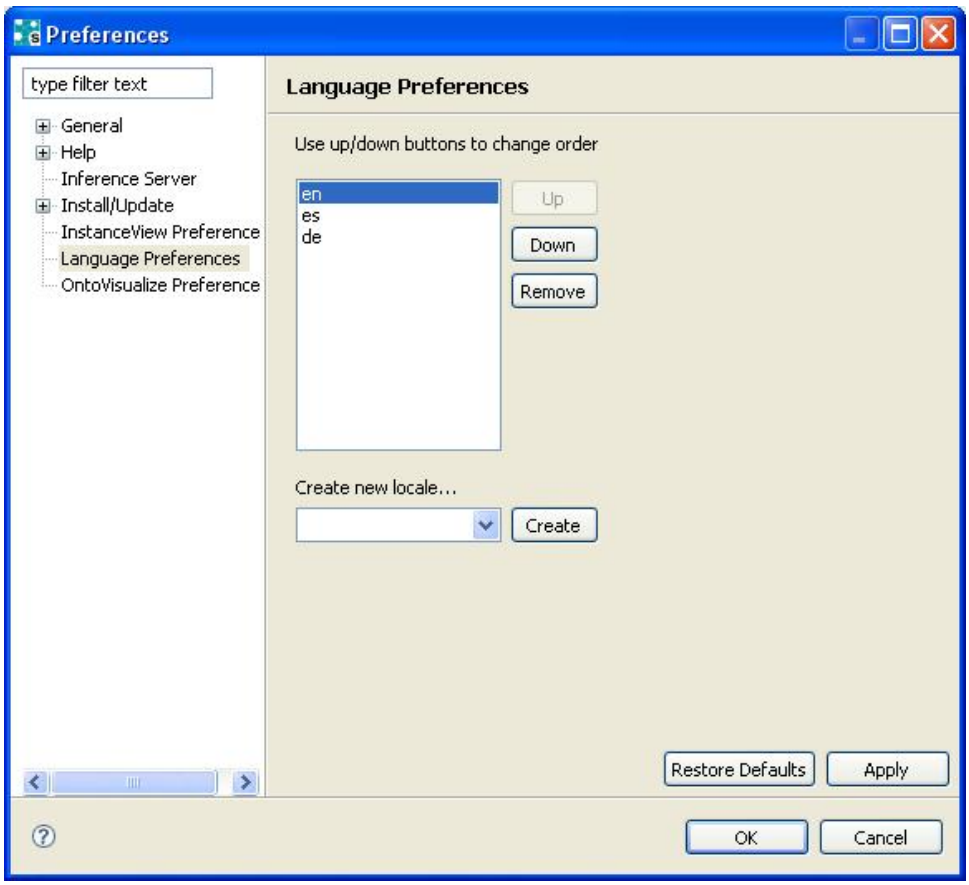


Figure 11: Languages Preferences used by LabelTranslator plug-in

Once activated, LabelTranslator uses some views of the Neon ToolKit to load the ontology and store the multilingual results respectively. In Figure 3, we show a screenshot of both the Ontology Navigator and the Entity Properties view with some linguistic information updated from the results obtained by LabelTranslator.

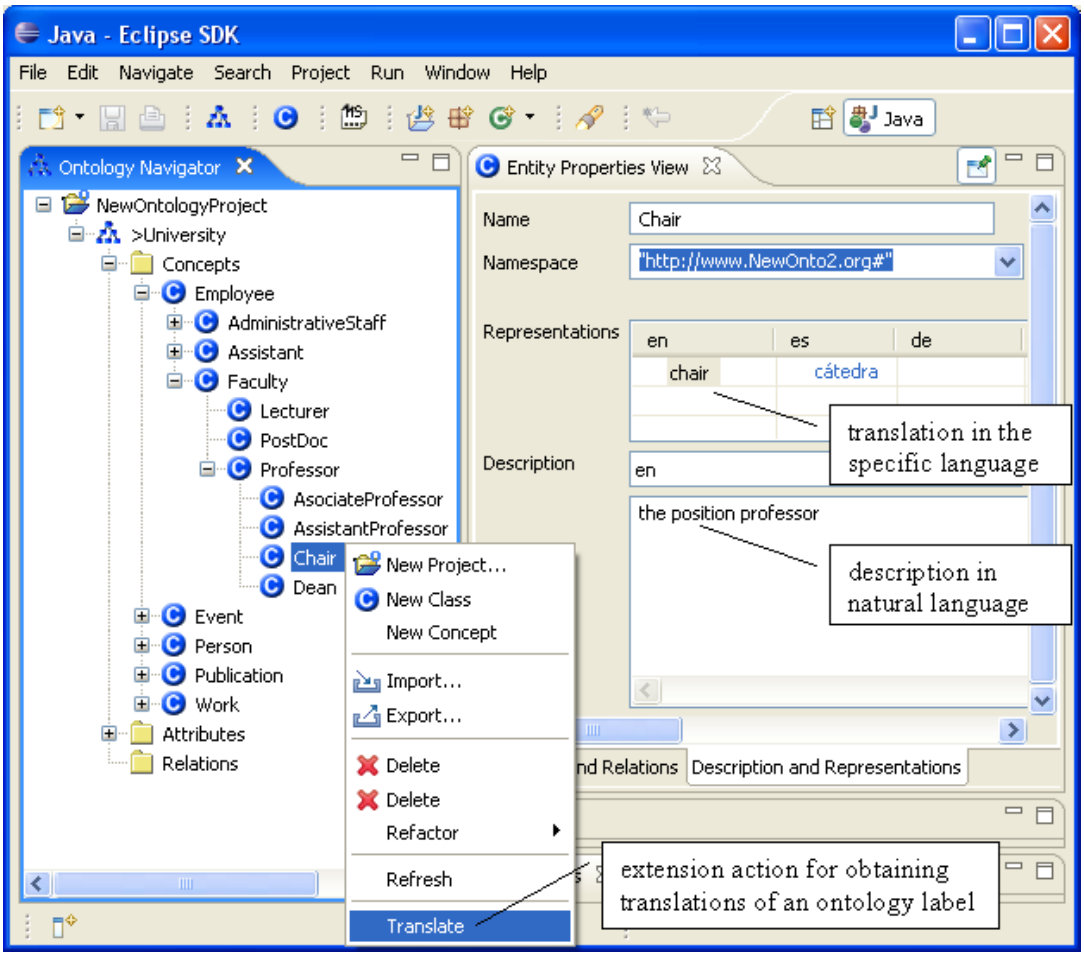


Figure 12: Screenshot of the main views of NeOn Toolkit used by LabelTranslator plugin

In the Ontology Navigator, right click on the frame (concept, attribute or relationship) and choose "Translate". Then, an user interface shows the ranked translations generated by our plug-in (see Figure 4). The system proposes the most relevant translation to be selected but the user can change this default selection.

Figure 13: Ranked translations of the concept "chair" generated by LabelTranslator plugin

Finally in the Entity Properties View tab, on the right of Figure 3 above, the plugin fills in runtime the fields *representation* and *descriptions* (that show linguistic information), according to the translations decided by the user. These fields represent the link between the conceptual knowledge and the discovered linguistic information.

3.6.3 Integration into the NeOn Toolkit

In the following we describe the high level architecture that features the 1st prototype. The first layer encapsulates the graphical user interfaces that permit the interaction with the user. The GUIs implemented in this layer allow a semi-automatic translation of a specific ontology label. The LabelTranslatorService implements the business functionality of the multilingual model (in the second layer). This service encapsulates some functionalities such as: translation of ontological labels, ranking of the senses of a translated label, etc. Finally, the third layer provides the repository in order to store the linguistic information. The current NeOn Toolkit views are used for storing the multilingual information.

For the first prototype of the NeOn toolkit, LabelTranslator plugin provides a further extension of the current actions of the contextual menu of the Ontology Navigator view. This action allows to the user translate a ontology label from a source language (Eg. English) to target language (Eg, Spanish). Another view used as user interface for the LabelTranslator plugin is the Entity Properties View, which shows property pages for the elements in the user interface. In this case the plugin does not add extensions; however, some fields and tables (that show linguistic information) are filled in runtime, according to the modalities decided by the Label Translator core

- Eclipse Extension Points
    - com.ontoprise.ontostudio.gui.navigator.concept.ConceptTreeElement
    - com.ontoprise.ontostudio.gui.navigator.attribute.AttributeTreeElement
    - com.ontoprise.ontostudio.gui.navigator.relation.RelationTreeElement

3.6.4 Intended Usage in the Case Studies

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

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

- As an intelligent translation assistance tool that help to 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 but the user can change this default selection. This possible user intervention is justified because the context could not be enough to translate an ontology label according to the user interpretation.

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.

3.7 LeDA

3.7.1 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 useability 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

3.7.2 User Documentation

LeDA View

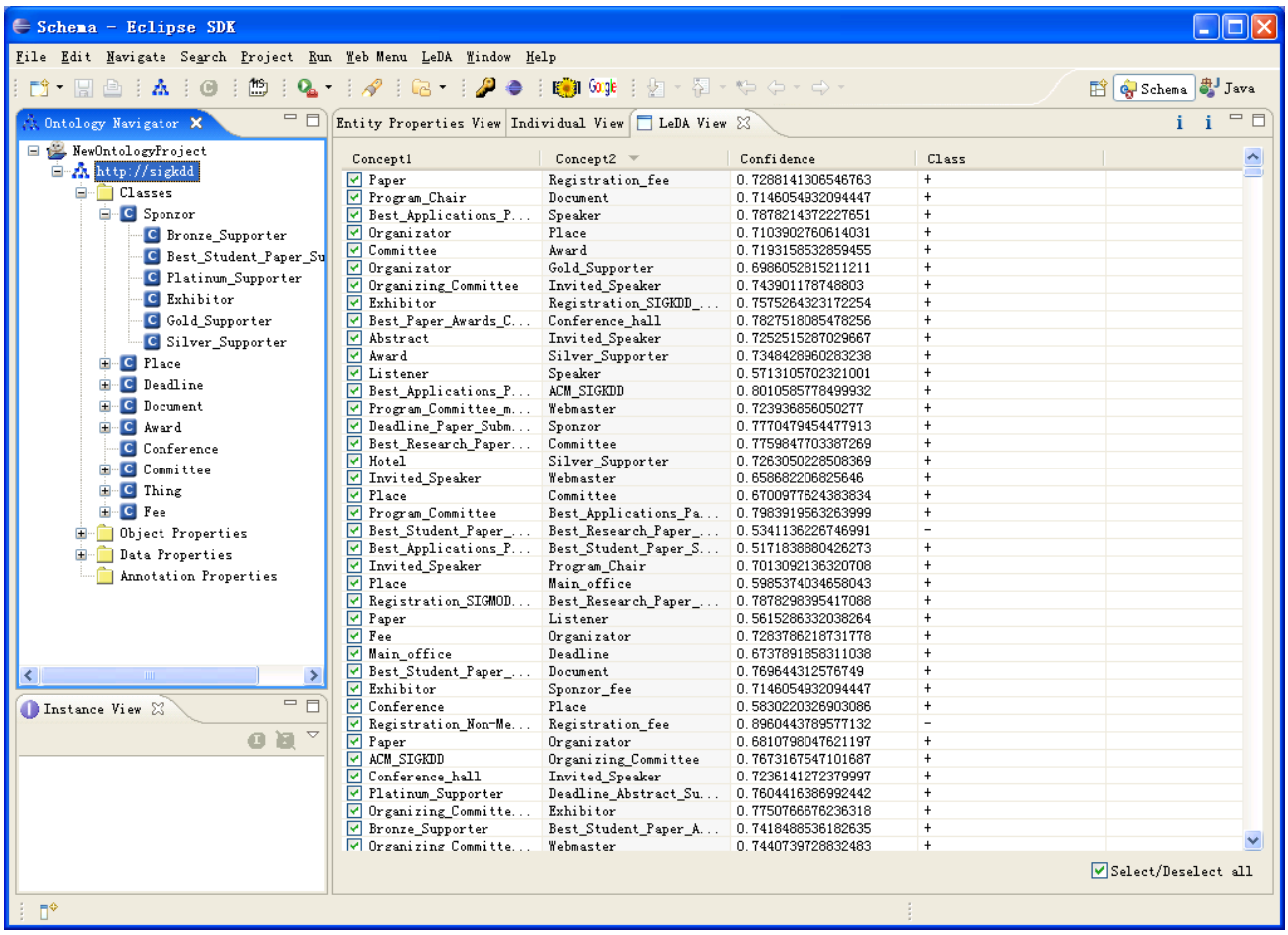


Figure 14: LeDA User Interface

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

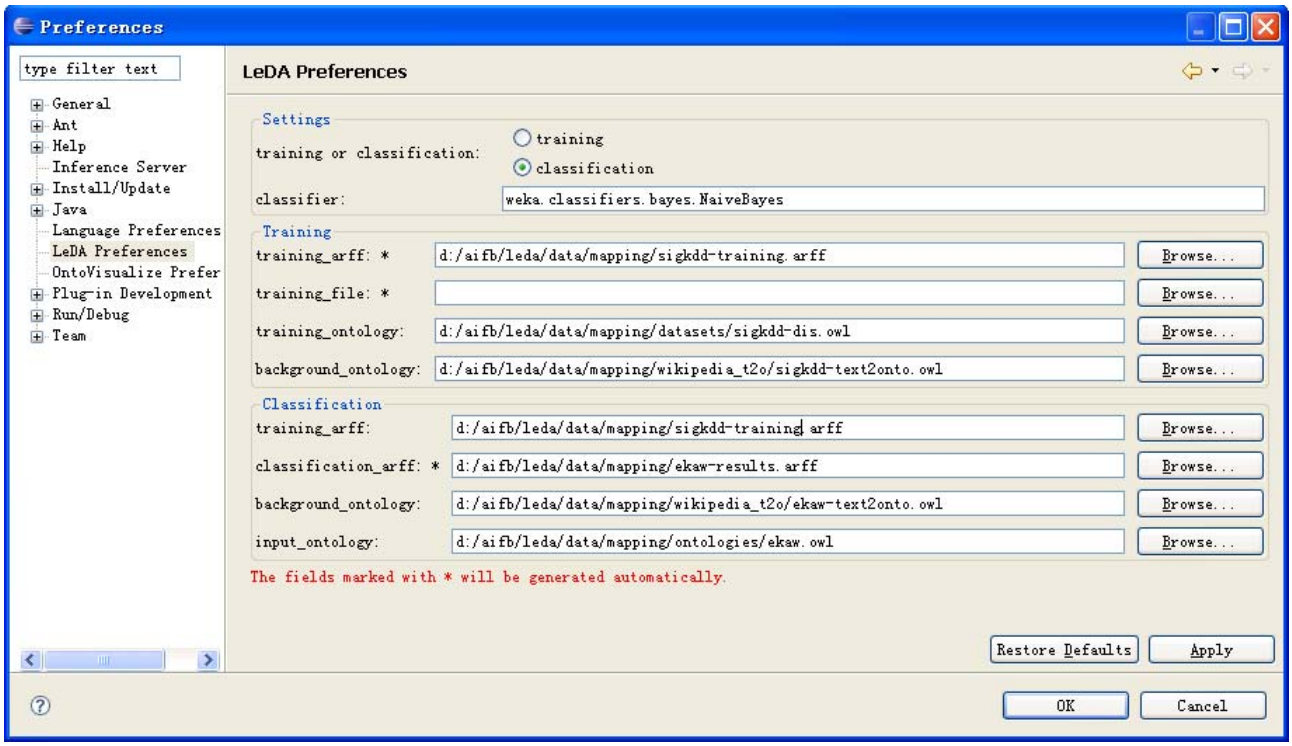


Figure 15: LeDA Preferences

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.

3.7.3 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

3.7.4 Intended Usage in the Case Studies

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

3.8 NeOnQA

3.8.1 Functional Description

NeOnQA introduces a novel approach to integrated distributed databases using networked ontologies. As our central contribution, we implement this approach by developing real life applications called NeOnQA to help integrating and querying distributed databases.

- We establish the mapping between ontology and database schema and also supports creating mappings between ontologies. The ontologies and databases can be distributed. We create ontology-database schema mapping by lifting database schema into an ontology that can be processed by reasoners, such as KAON2.
- We integrate the ontologies and mappings that are distributed and interconnected. NeOnQA uses Oyster ere as metadata registry to identify the remote ontology resource and propagate local resource.

3.8.2 User Documentation

Now we present two use cases to get started. In the first scenario, we intend to create a database schema mapping for the database FIGIS residing locally. Please note: Remote databases can be also accessed by specifying its IP address with the database name). The sample procedure can be as follows:

- Step 1: We should first connect to the database by specifying the database's address, its username and password. If the connection successfully established, the database schema would be automatically extracted as a tree-like structure as shown below.

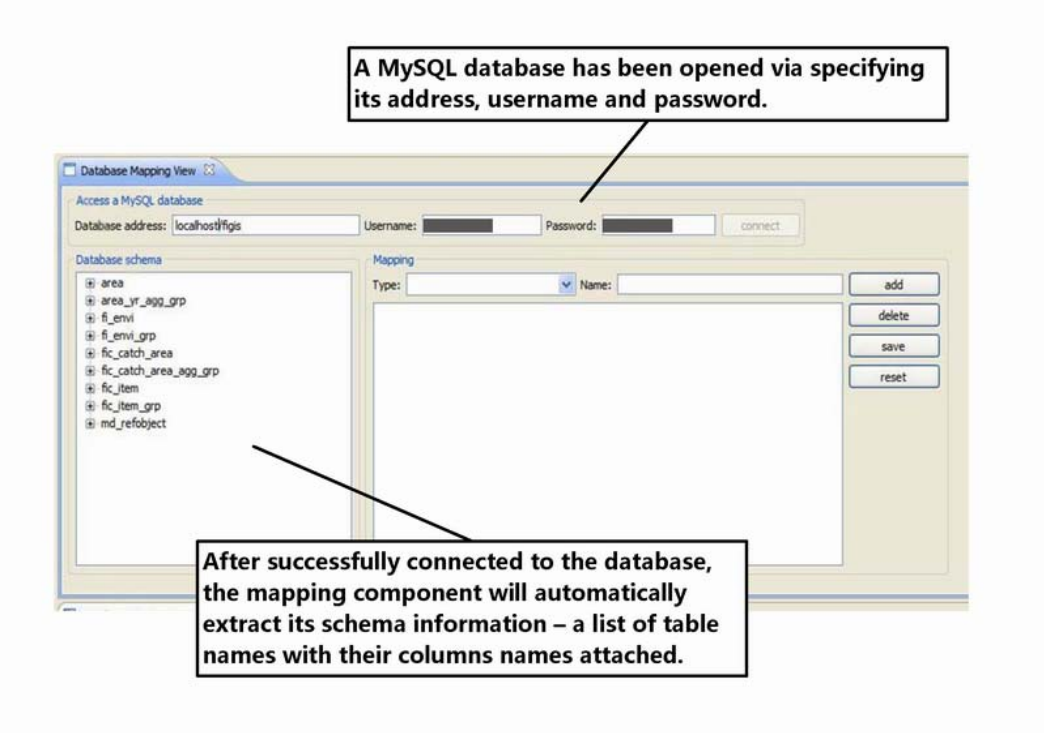


Figure 16: NeOnQA – Extracting Schema Information

- Step 2: Now we can manually select a certain column to create an OWLClass-mapping, or two columns in the same table to create an ObjecProperty, DataProperty or AnnotationProperty. The creation of mapping entries is usually according to the semantics involving in the database schema. (Figure below)

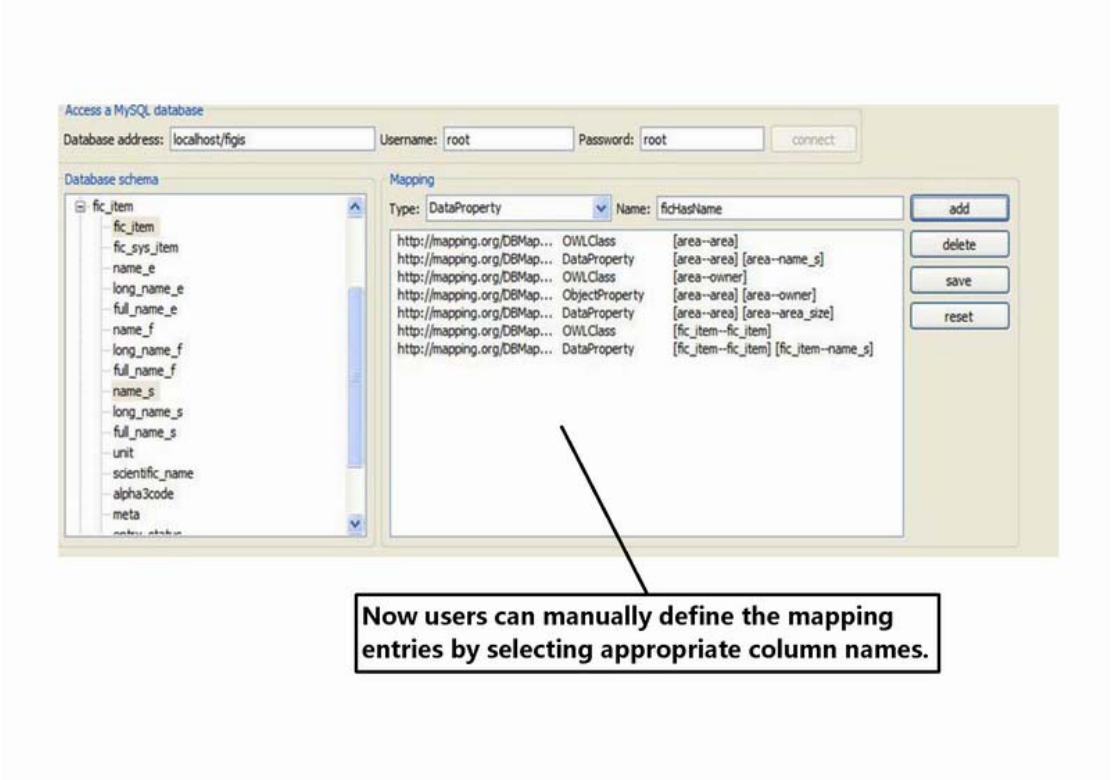
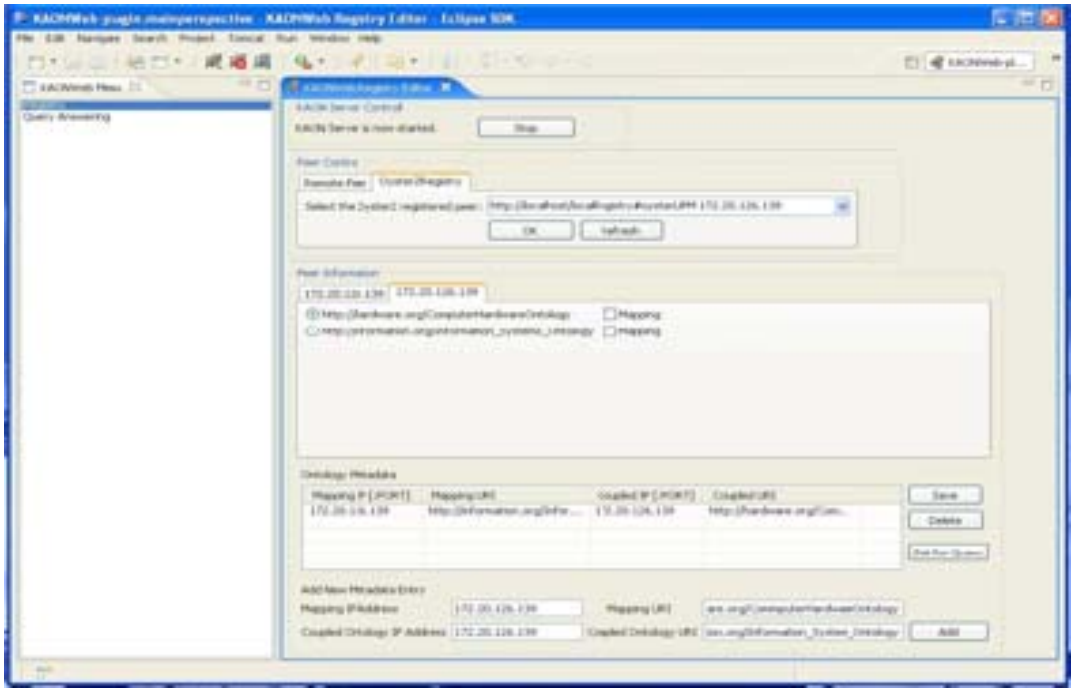


Figure 17: NeOnQA - Creating database mappings

- Step 3: Do not forget to click the “save” button to persist this database schema mapping after all the necessary mapping entries has been created.

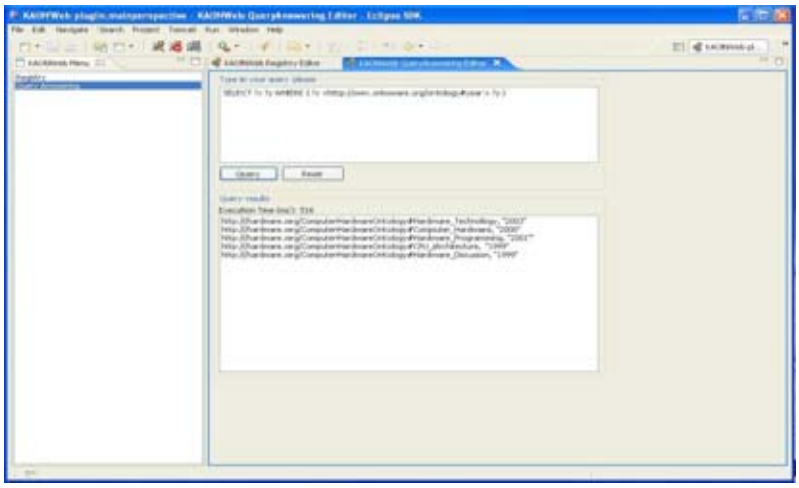
The figure below shows the view “Registry”, including functionalities like:

- Turn on/off the server(including the KAON2 server mode and the services for network communication)
- Manage local ontology and modify its metadata
- Access the ontology and its metadata on remote peers, either through direct IP address input or through Oyster2 registered peer information.
- Set one of the managed ontology as the target for query (deliver the virtual ontologies to several peers for concurrent reasoning later)



The second view “Query Answering” includes functionalities like:

- Query answering with SPARQL
- Show the time interval for query answering



3.8.3 Integration into the NeOn Toolkit

- Eclipse Extension Points
  - com.ontoprise.ontostudio.gui
- NeOn Toolkit extension Points
  - Datamodel

- extendableTreeProvider

Users are able to start a distributed query answering task by clicking a button in the tool bar in the main GUI. The new Eclipse perspective will appear in the GUI as component for distributed query answering.

Users are also able to start a distributed query answering task by right-clicking an ontology in the tree-like ontology navigator.

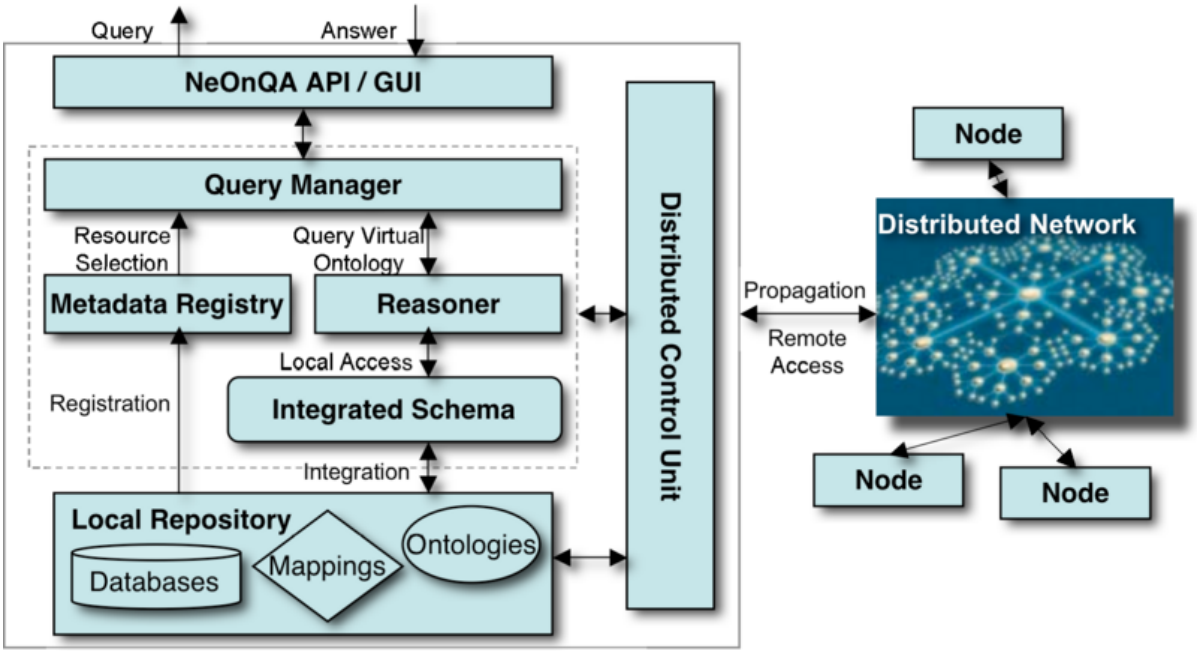


Figure 18: Architecture of NeOnQA

The Figure above depicts the architecture of one NeOnQA node that interacts with other NeOnQA nodes in the distributed network. Next, we introduce the components in this architecture in detail.

- The local repository consists of databases, mappings and ontologies. In this deliverable, we consider ontologies as OWL DL ontologies and mappings as database-ontology mappings, or ontology-ontology mappings. The mappings are responsible for linking heterogeneous resources. The local repository can be shared in the distributed network.

- The integrated schema provides an integrated view towards the local repository. We apply logic integration to the database schema and ontology TBox, formulating an integrated schema using mappings. It therefore doesn't matter whether the data are stored in database or ontology ABox as we only need to query against the integrated schema. We can also access and integrate remote databases and ontologies by using distributed control unit introduced later. Here we mainly focus on discussing the approaches for integrating databases as our previous work have addressed the problem of distributed ontology ABox integration and query answering.

- The query manager is the component responsible for answering queries over extracted database schemata in the distributed network. Here we consider queries as conjunctive queries over OWL DL ontologies that are mapped with database schemata. The query process can be divided into two steps: Resource selection and query answering.

    - The purpose of the resource selection is to search resources in the distributed network that are relevant to answer a particular query using the metadata registry. The Metadata Registry maintains metadata about resources available (i.e. nodes,

ontologies, and mappings), which may be accessible either locally or remotely in the distributed network. We extend the selection approach introduced in our previous work by enabling the identification of mappings in the metadata of distributed nodes.

- In the step of query answering, the query is evaluated against the integrated schema by the reasoner. In NeOnQA, we rely on KAON2 as the underlying reasoning engine, as KAON2 does not retrieve the remote data but only accesses the remote schema.

- The distributed control unit is an important component that is mostly different from the P2P network sub-layer introduced in. On the on hand, we use web service to propagate the resource in local repository. On the other hand, the communications with remote nodes for retrieving schema or sending results are established by direct JAVA socket connection due to overheads occurred in our real life experiments in using web service for data transfer.

- Users can issue queries by using NeOnQA GUI and API, get results of queries, identify local and remote resource, and manage the local system. We use SPARQL as conjunctive query language in NeOnQA.

3.8.4 Intended Usage in the Case Studies

The NeOnQA plugin will be used in WP7 for query answering over semantic data. Following the screenshots shows above, we describe an example here in detail:

- First, we load a FIGIS database fragment and check the schema of this database.

- Second, we create mapping between ontology and database by recognizing entities in the schema and creating ontology entities.

- Third, we can save the lifted database schema as an FIGIS ontology.

- Forth, we can query this FIGIS ontology directly using SPARQL, the NeOnQA system will use KAON2 engine to query the database.

3.9 OntoModel

3.9.1 Functional Description

OntoModel is a UML-based editor for OWL ontologies, rules and ontology mappings. It makes use of the extension mechanism of UML to define profiles for the depiction of the constructs of the ontology languages.

3.9.2 User Documentation

OntoModel is invoked via the context menu on an OWL ontology (“Show Diagram”) as shown in Figure 19: OntoModel - Showing an Ontology in a UML diagram. This action will open the OntoModel perspective and show the ontology in a UML-based diagram as shown in

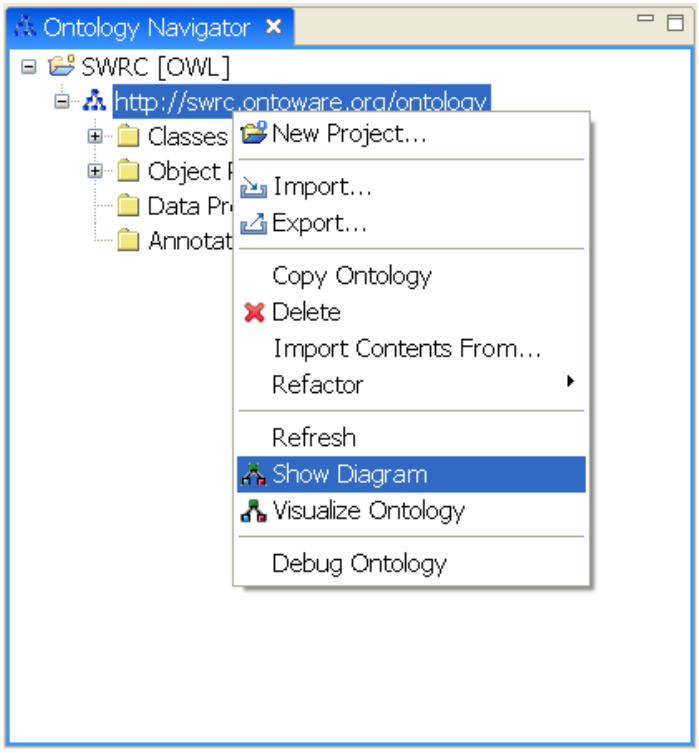


Figure 19: OntoModel - Showing an Ontology in a UML diagram

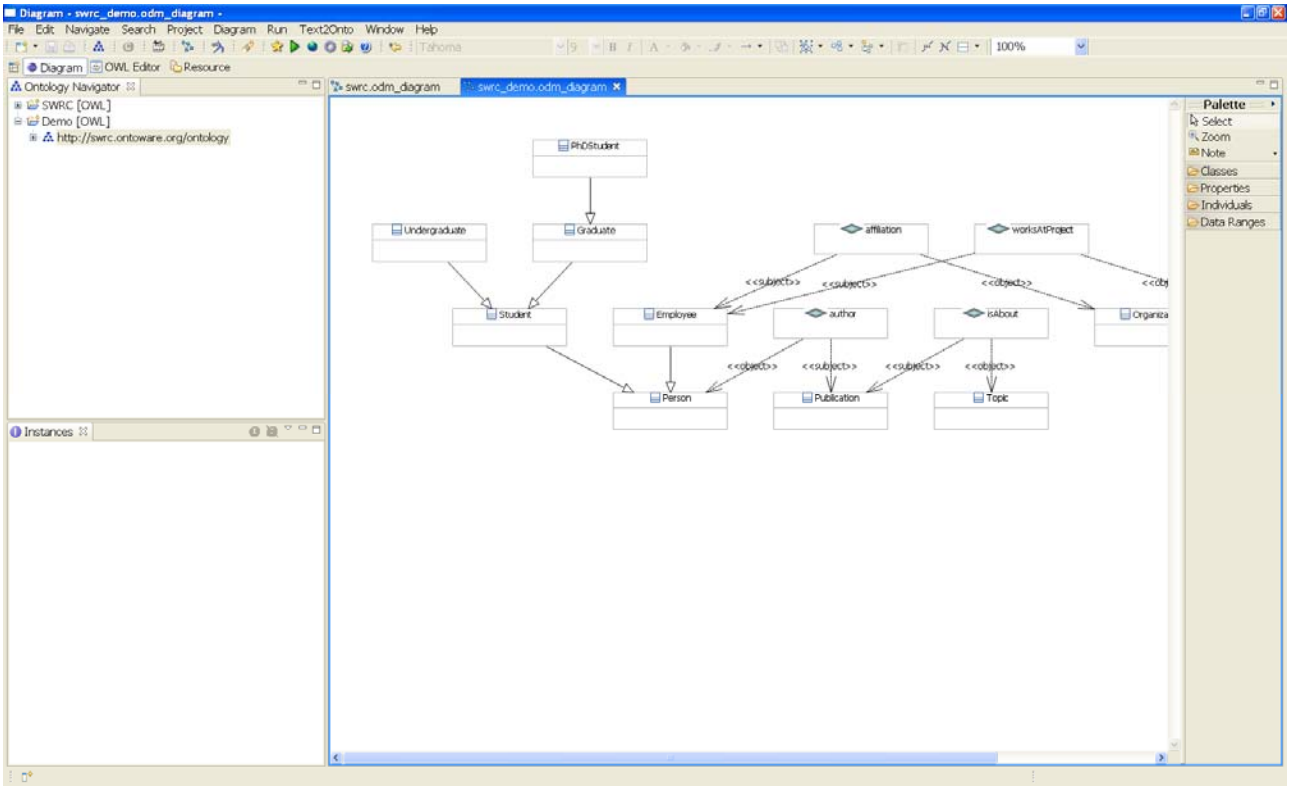
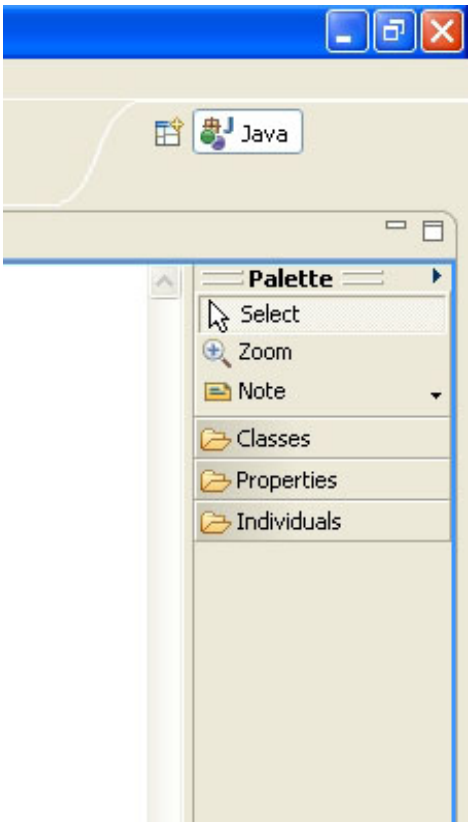
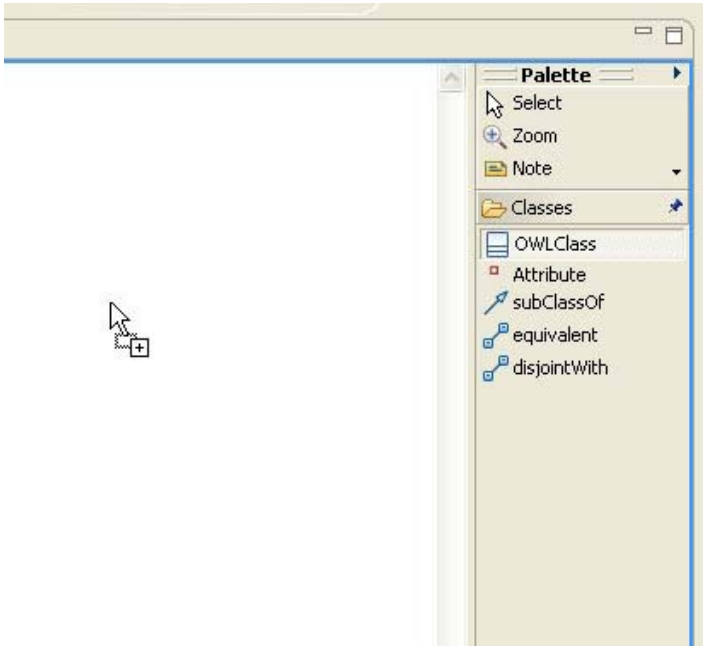


Figure 20: OntoModel Screenshot

The Constructs Palette - Although the tool supports all available OWL 1.1 elements, the palette on the right hand side of the diagram only contains the most common constructs for classes, properties and individuals.



Modelling a new construct - After selecting a construct in the palette, a click in the diagram field builds a new construct. Constructs that are connections between other constructs, are modelled by dragging it from one to the other construct.



Moving constructs - Constructs can be moved by dragging it.

Deleting constructs - Constructs can be deleted by selecting the option „Delete from Model“ after right-clicking on it.

Arranging diagrams - Selected elements can be arranged automatically using the „Arrange All“ icon once or several times. The line style of arrows and lines can be selected using the „line style“ icon.

3.9.3 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.

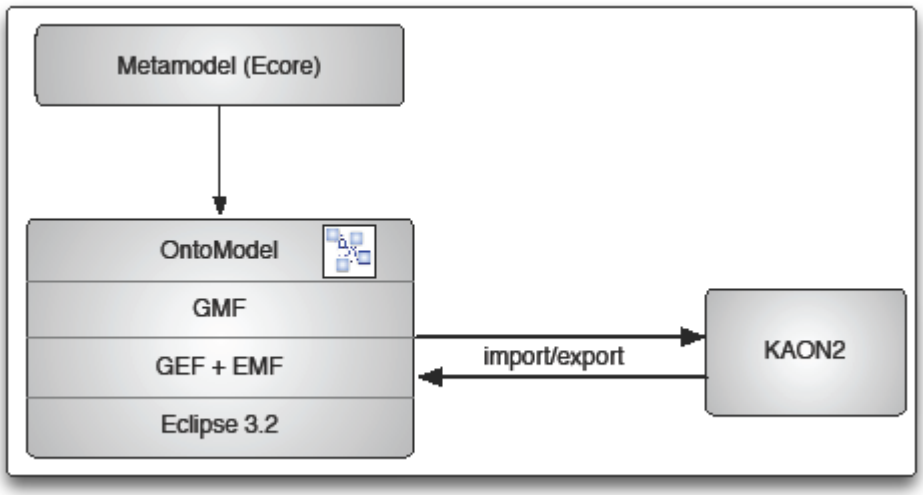


Figure 21: Ontomodel Plugin Architecture

Figure 21: Ontomodel Plugin Architecture shows how OntoModel builds on Eclipse and some of its available plug-ins: it builds on the Graphical Modelling Framework (GMF[3]), which in turn builds on the Graphical Editing Framework (GEF[4]) and the Eclipse Modelling Framework (EMF[5]).

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,

[3] http://www.eclipse.org/gmf/

[4] http://www.eclipse.org/gef/

[5] http://www.eclipse.org/modeling/emf/

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.

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.

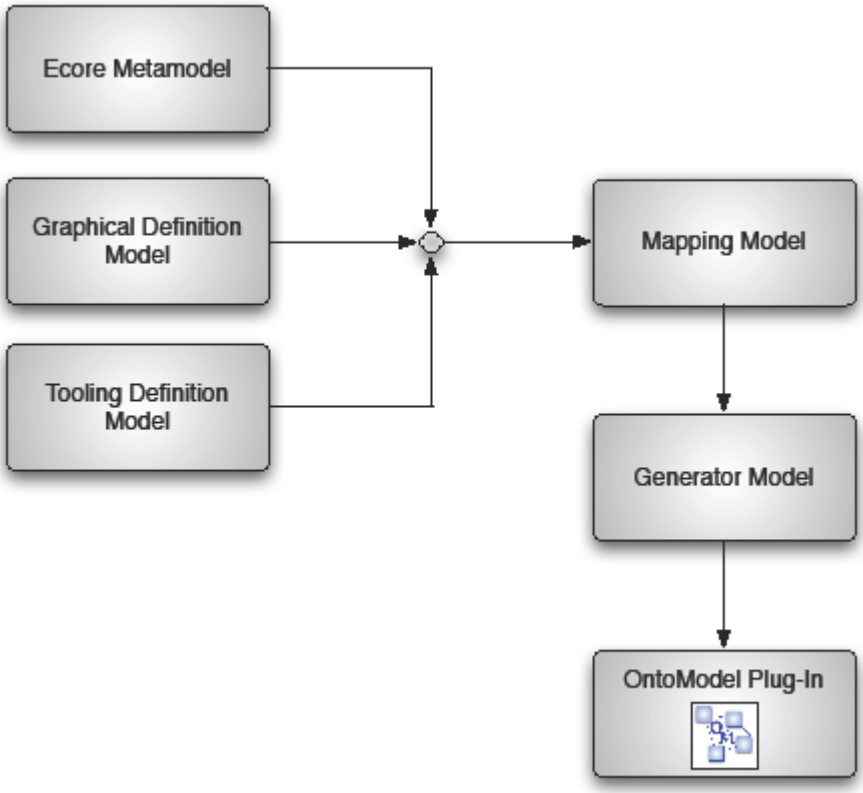


Figure 22: Components and models used during the GMF-based development of OntoModel

Figure 22: Components and models used during the GMF-based development of OntoModel illustrates the main components and models used during the GMF-based development of OntoModel.

The graphical definition model is the core concept of GMF. It contains the information related to the graphical elements that are used in our ontology models. However, they do not have any direct connection to our metamodel. The graphical definition model is generated semi-automatically from our Ecore metamodel. (Note that, although our editor is UML-based, GEF also provides other graphical elements.)

Similarly, the tooling definition model is generated semi-automatically from the metamodel to design the palette and other periphery (menus, toolbars, etc).

A goal of GMF is to allow the graphical definition to be reused for several domains. This is achieved by using a separate mapping model to link the graphical and tooling definitions to the selected domain metamodel. This semi-automatically generated model is a key model to GMF development and is used as input to a transformation step producing the generator model.

The generator model generates the OntoModel plug-in which is then refined with dialogs and other application-specific user interface aspects. The plug-in bridges the graphical notation and the domain metamodel when a user is working with a diagram, and also provides for the persistence and synchronization of both. An important aspect of GMF-based application development is that a service-based approach to EMF and GEF functionality is provided which can be leveraged by non-generated applications.

3.9.4 Intended Usage in the Case Studies

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

With OntoModel, we addressed the following uses cases of this case study:

- Build, adapt and visualize the common Pharmainnova ontology as well as the partners' ontologies.
- Define mappings between the common and each partner's ontology.

As the typical end user in this scenario is not experienced in modelling 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 modelling 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.

3.10 Ontology Relationship Visualizer

3.10.1 Functional Description

This plugin adds a new visualization paradigm to the NeOn toolkit. The plugin provides to the user a navigation driven by the relations contained in the classes of the ontology. The motivation for this type of navigation is the need of the pharmaceutical invoicing use case to present to the end users a complex ontology like the invoice reference ontology described in D8.3.1. The goal of the plugin is to simplify the elements that a user has to visualize in order to annotate an invoice into a complex ontology.

In this plugin we abstract the visualization of ontologies part from the annotation and invoice visualization parts. The plugin visualizes OWL ontologies and the user has to indicate to the plugin which is the starting class s/he wants to visualize. Once this class is selected the visualization starts. The user is able to view the domain class of the relation and its range class. The user can navigate until there is no other relation in the last class visualized. The user is also able to view the properties of each class but s/he is not able to edit them.

3.10.2 User Documentation

The user only needs to right click one class in the tree visualization of the ontology and select the “Relation Visualization” option.

3.10.3 Integration into the NeOn Toolkit

The OntoVisualize plugin is realized as a new view. It makes use of the following extension points:

* org.eclipse.ui. perspectives

* org.eclipse.ui.popupMenus

* org.eclipse.ui.views

* org.eclipse.ui.viewActions

3.10.4 Intended Usage in the Case Studies

The plugin is part of the invoicing prototype. This type of visualization is used by the invoicing prototype along with other functionalities for annotating the invoices that are sent to a company. This visualization is used for presenting to the users an ontology in the most simple way possible.

3.11 OWLDoc

3.11.1 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.

3.11.2 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.

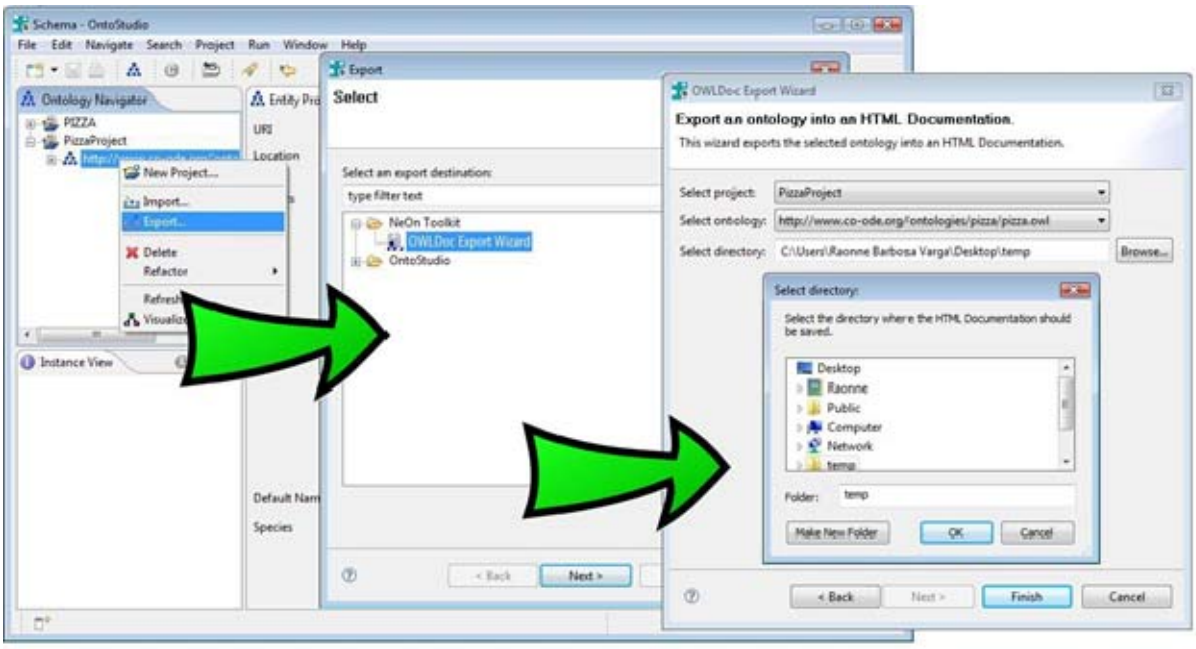


Figure 23: OWLDoc Export

3.11.3 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

3.11.4 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.

3.12 Oyster

3.12.1 Functional Description

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. An *ontology registry* is a system that provides services for storage, cataloguing, discovery, management, and retrieval of ontology (and related entities) metadata definitions. It provides the means to support advanced semantic searches of ontologies based on the their characteristics. To achieve this goal, Oyster implements the proposal for metadata standard OMV (http://omv.ontoware.org) as the way to describe ontologies and related entities.

Oyster offers a user driven approach where each peer has its own local registry of ontology metadata and also has access to the information of others registries, thus creating a virtual decentralized ontology metadata registry. The goal is a decentralized knowledge sharing environment using Semantic Web technologies that allows developers to easily share ontologies.

The Oyster system (freely available under http://ontoware.org/projects/oyster2/) was designed using a service-oriented approach, and it provides a well defined API. Accessing the registry functionalities can be done using directly the API within any application, invoking the web service provided or using the included java-based GUI as a client for the distributed registry. In Oyster, ontologies are used extensively in order to provide its main functions (register metadata, formulating queries, routing queries and processing answers):

- *Creating and importing metadata:* Oyster2 enables users to create metadata about ontologies manually, as well as to import ontology files and to automatically extract the ontology metadata available, letting the user to fill in missing values. The ontology metadata entries are aligned and formally represented according to two ontologies: (1) the OMV ontology, (2) a topic hierarchy (i.e. the DMOZ topic hierarchy), which describes specific categories of subjects to define the domain of the ontology.

- *Formulating queries:* A user can search for ontologies using simple keyword searches, or using more advanced, semantic searches. Here, queries are formulated in terms of these two ontologies. This means queries can refer to fields like name, acronym, ontology language, etc. or queries may refer to specific topic terms.

- *Routing queries:* A user may query a single specific peer (e.g. their own computer, because they can have many ontologies stored locally and finding the right one for a specific task can be time consuming, or users may want to query another peer in particular because this peer is a known big provider of information), or a specific set of peers (e.g. all the member of a specific organization), or the entire network of peers (e.g. when the user has no idea where to search), in which case queries are routed automatically in the network.

- *Processing results:* Finally, results matching a query are presented in a result list. The answer of a query might be very large, and contain many duplicates due to the distributed nature and potentially large size of the P2P network. Such duplicates might not be exactly copies because the semi structured nature of the metadata, so the ontologies are used again to measure the semantic similarity between different answers and to remove apparent duplicates.

The high-level design of the architecture of a single Oyster node is shown in the figure below.

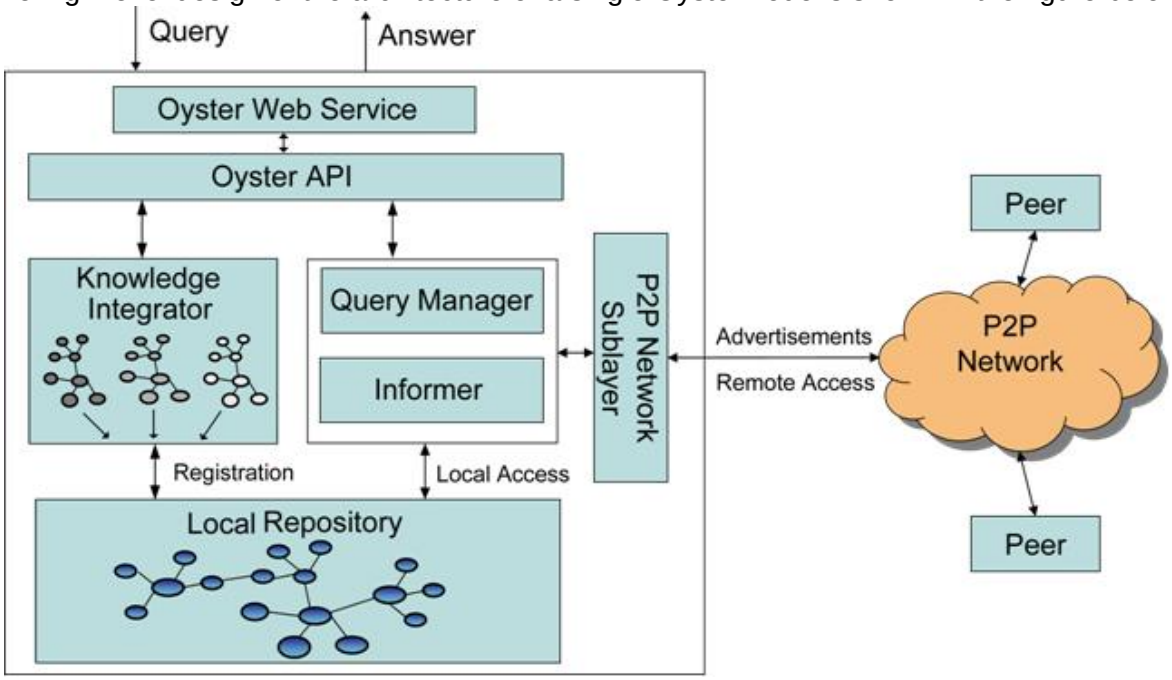


Figure 24: Oyster Architecture

In the following, we briefly discuss the individual components of the system architecture. For a complete description we refer the reader to NeOn deliverable D1.4.1.

- The *Local Repository* of a node contains the metadata about ontologies and related entities (i.e. OMV instances) that it provides to the network. It supports query formulation and processing and provides the information for peer selection. In Oyster, the Local Repository is based on KAON2 and it supports SPARQL as its query language.
- The *Knowledge Integrator* component is responsible for the extraction and integration of knowledge sources (e.g. ontologies) into the Local Repository. This component is also in charge of how duplicate query results are detected and merged.
- The *Query Manager* is the component responsible for the coordination of the process of distributing queries. It receives queries from the user interface, API or from other peers. Either way it tries to answer the query or distribute it further according to the content of the query.
- The *Informer* component is in charge of proactively advertise the available knowledge of a Peer in the distributed network and to discover peers along with their expertise (i.e. the expertise contains a set of topics (i.e. ontology domains) that the peer is an expert in).
- The *Peer-to-Peer network sub-layer* is the component responsible for the network communication between peers. In Oyster, we rely on an RMI based implementation; however, other communication protocols would be possible as well.
- The *Oyster API* provides a well defined interface in Java with a set of methods that expose all the registry functionalities (available under http://ontoware.org/projects/oyster2/).
- The *Oyster Web Service* encapsulates the Oyster API and provides a free realisation of the NeOn registry service compliant with the eBXml standard (available under http://ontoware.org/projects/oyster2/).

3.12.2 User Documentation

Oyster Import Wizard

As we explained above, Oyster is a distributed registry where each node in the Oyster network has its own local registry. Currently to start the local node, we need to start a KAON2 server instance outside the NeOn toolkit (see section *how to use it*), but in the future the local node will start within the NeOn toolkit.

Oyster Import Wizard release (Oyster2ImportWizard_vx.xx) contains:

- org.neon_toolkit.oyster.plugin.iwizard.zip
  - Contains the NeOn Plugin
- Oyster2APIvx.xx.zip
  - Contains the necessary files for Oyster
    - new Store (configuration file)
    - O2ServerFiles (directory)
      - Contains all the necessary files: ontology files, logo, etc.
    - Oyster2 (directory)
      - Contains kaon2.jar and run.bat

How to install it

- Extract Oyster2APIvx.xx.zip (e.g. c:\Oyster2APIvx.xx) Note: The default is c:\Oyster2APIvx.xx.
- Copy file "new store" to the working directory of the NeOn Toolkit (e.g. where neontoolkit.exe is located)
- Set the name of your peer i.e. in file "new store" change property "localPeerName" which has by default the value "myLocalPeer"
- Execute run.bat to start a KAON2 instance.
- Extract org.neon_toolkit.oyster.plugin.iwizard.zip into NeOn_Toolkit folder. It will copy the jars files into the plugin subdirectory

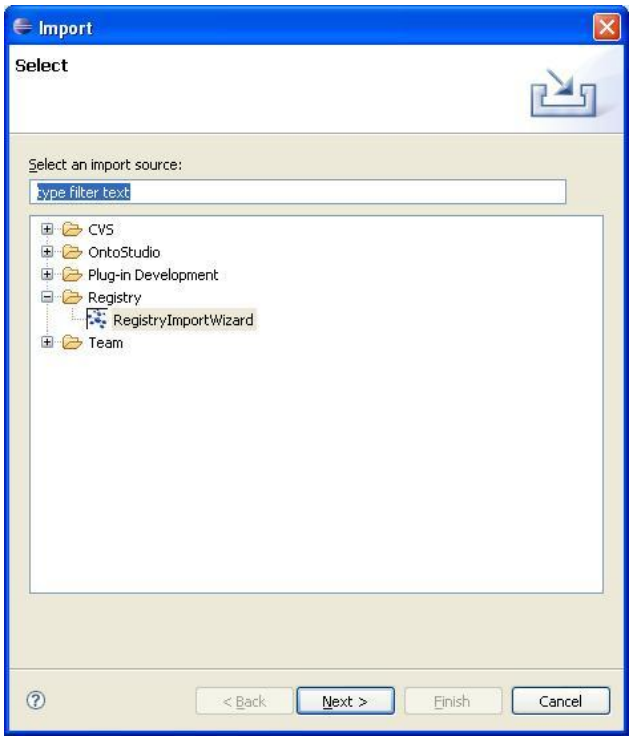
To change the location of the files required files, you can edit the new store configuration file and the run.bat

For additional information please refer to http://ontoware.org/projects/oyster2

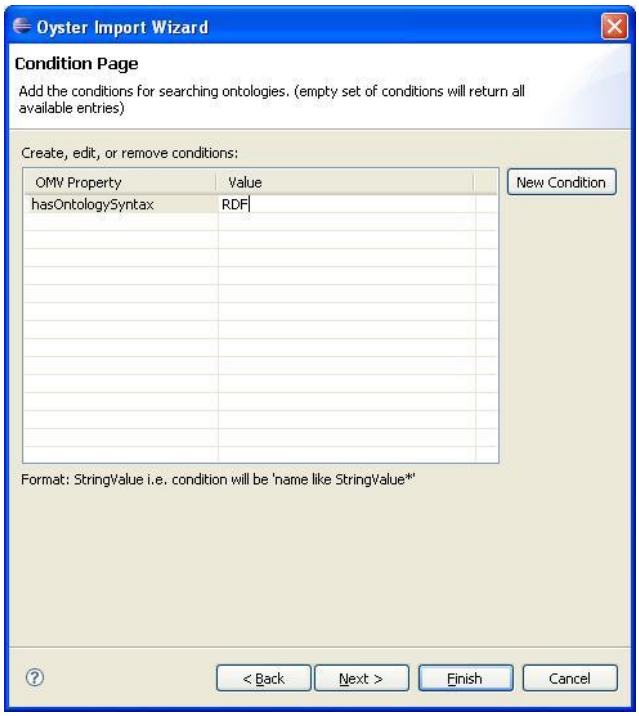
How to use it

The current version of the import wizard uses the Oyster API. The following figures show how the import wizard works.

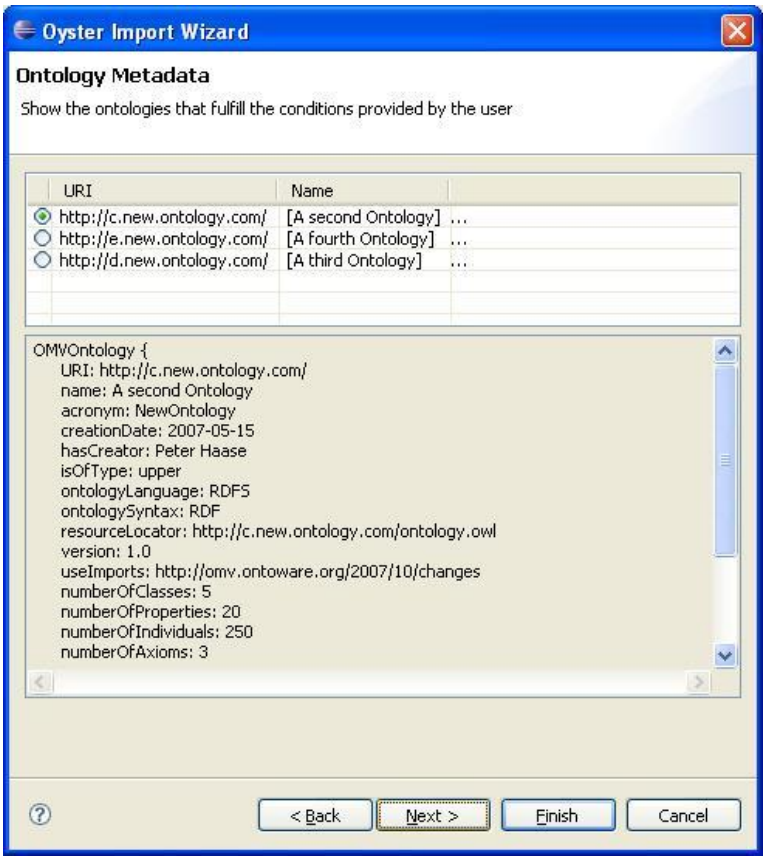
- Launch the import wizard



- Specify the conditions (i.e. OMV properties) that should be fulfilled by the ontologies we are searching



- Select an ontology from the result list



- Select the NeOn project in which will be imported the selected ontology. Note that the selected entry is the OMV metadata (not the ontology itself). However to import the ontology into the workspace we use the resourceLocator information that is the location where the actual ontology can be found

3.12.3 Integration into the 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, as described above, 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, as part of the future work, Oyster will provide some engineering components that will rely on the registry services (e.g. Versioning). Finally, Oyster will also provide GUI components (i.e. plugins) that implement user interfaces to the registry services. In particular two GUI plugins will be implemented as described below.

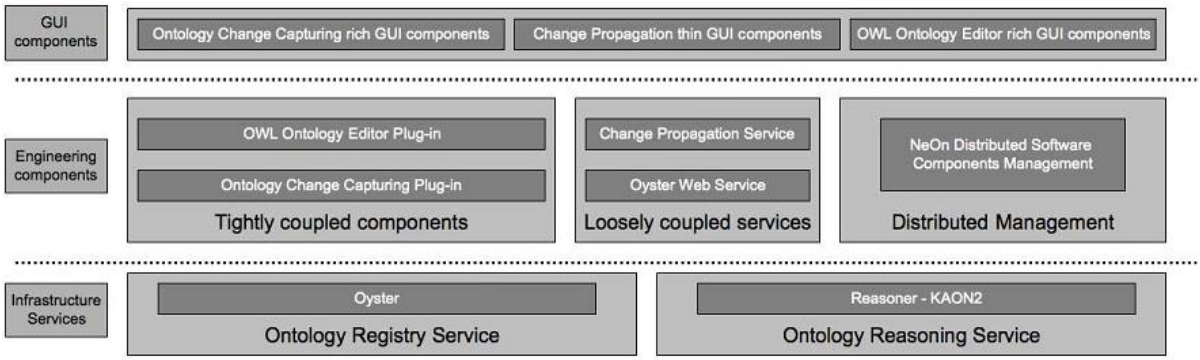


Figure 25: Oyster in the NeOn Architecture

Oyster Import Wizard

For the first prototype of the NeOn toolkit, Oyster provides an import wizard plugin that allows users to search for ontologies based on the available metadata.

- Eclipse Extension Points
  - importWizard
  - org.eclipse.ui.perspectives
- Relies or depends on the following plugins
  - Com.ontoprise.ontostudio.datamodel
  - Com.ontoprise.ontostudio.gui
  - Com.ontoprise.ontostudio.io
  - Datamodel 5.1
  - DatamodelBase 5.1
  - Util 5.1

Oyster Perspective

For the second protype of the NeOn toolkit, Oyster will provide a perspective which implements the complete GUI to he registry.

3.12.4 Intended Usage in the Case Studies

Oyster will be used in WP7 as ontology registry.

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 the ontology keeping track of all changes in a controlled and coherent manner. The registry 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 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 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 ontology related entities.

3.13 RaDON

3.13.1 Functional Description

The purpose of the RaDON plugin is to deal with inconsistency and incoherence occurring in networked ontologies. Specifically, RaDON provides the following functionalities:

- **Compute all Minimal Unsatisfiability-Preserving Subsets (MUPS)**: This functionality corresponds to the button of "Compute MUPS" which can be activated if the ontology is incoherent;
- **Compute the Minimal Incoherence-Preserving Subsets (MIPS)**: This corresponds to the button of "Compute MIPS" which is activated if the ontology is incoherent;
- **Compute all Minimal Inconsistent Subsets (MIS)**: It corresponds to the button of "Compute MIS" which is activated if the ontology is inconsistent;
- **Repair automatically**: This corresponds to the button of "Repair Automatically". This functionality is used basing on the found MUPS or MIPS or MIS. If MUPS/MIPS/MIS are computed, the section of MUPS/MIPS/MIS will be expanded automatically. And 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;
- **Repair manually**: This corresponds to the button of "Repair Manually" which is located in each section like the button of "Repair Automatically". If this button is activated, a new dialog will be shown with the information of MIPS or MIS. User could choose the axioms to be removed.

3.13.2 User Documentation

The user interface for debugging and repairing is accessible as a view called "Debug and Repair Ontology View".

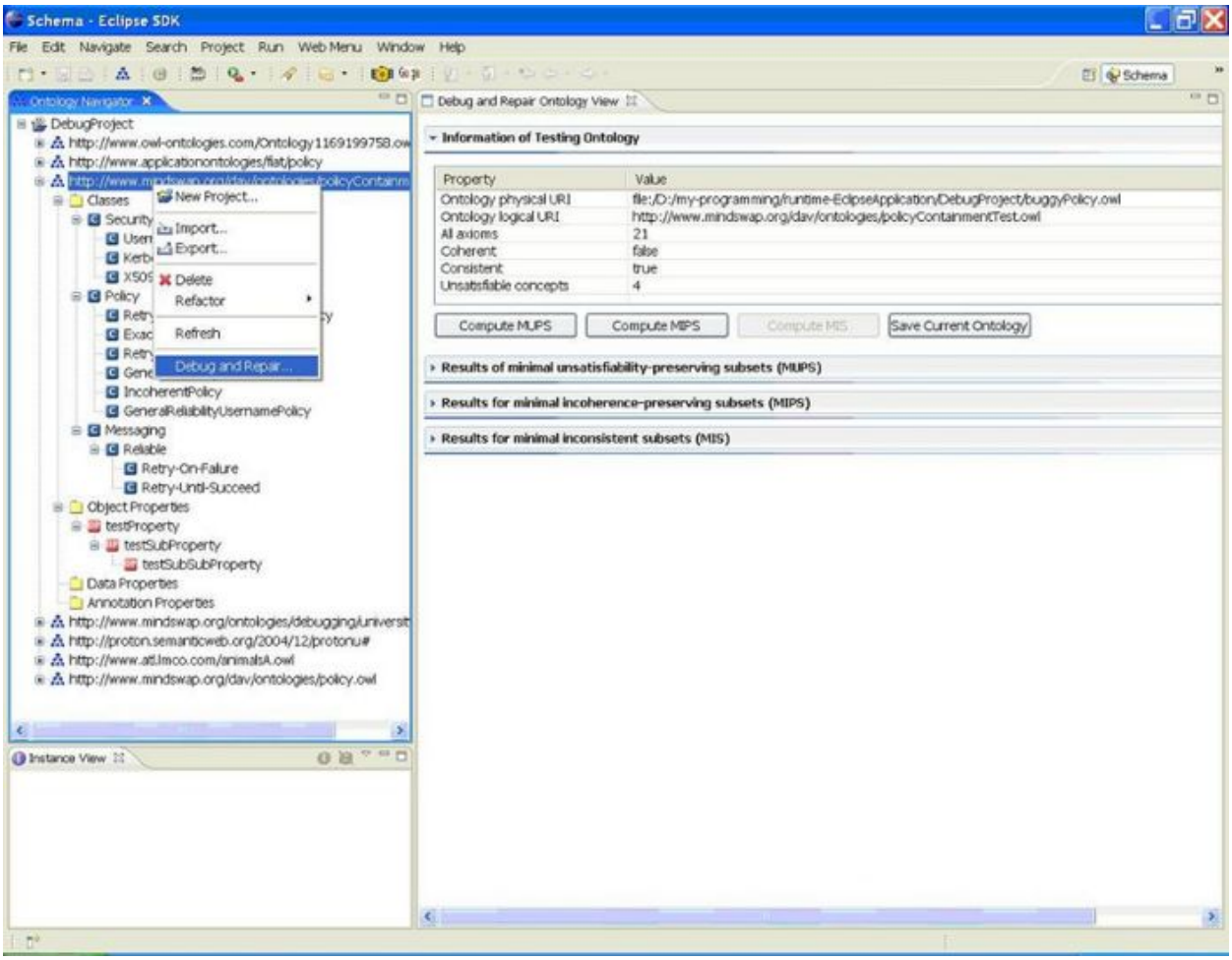


Figure 26: RaDON Plugin

To debug or repair different ontologies, right-click an ontology in the view of OntologyNavigator in NeOn Toolkit and select the item of "Debug and Repair...". (as shown in Figure 26: RaDON Plugin). Then our "Debug and Repair Ontology View" is invoked. At the same time, the logical and physical URIs of this chosen ontology and the size of all axioms are shown. Besides, we also show some information about whether this ontology is inconsistent or incoherent and how many unsatisfiable concepts.

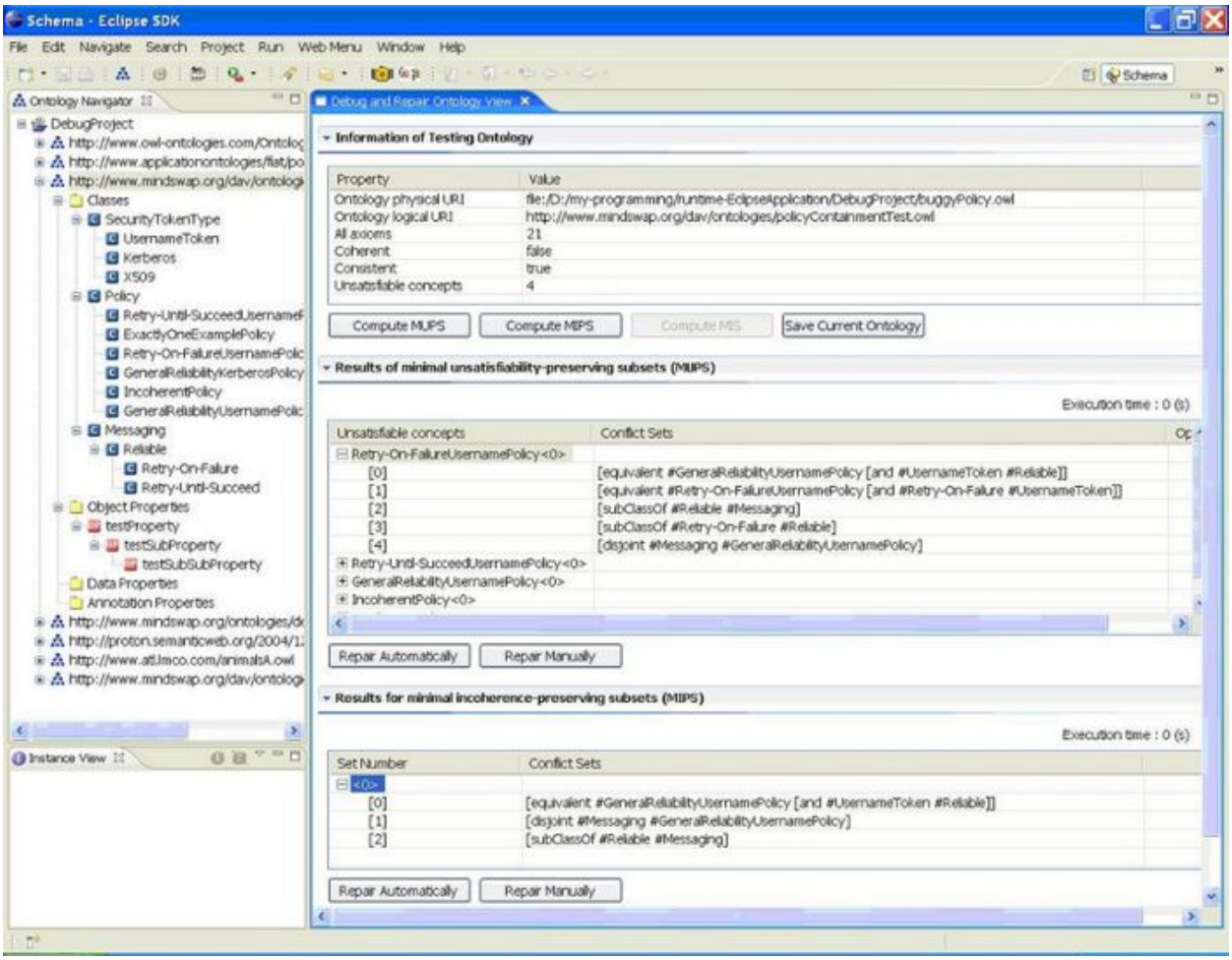


Figure 27: Computing MUPS / MIPS / MIS

If the test ontology is incoherent, user can obtain all MUPS and / or MIPS by pressing the buttons "Compute MUPS" and / or "Compute MIPS" respectively. The corresponding sections will be expanded accordingly. Similarly to compute MIS.

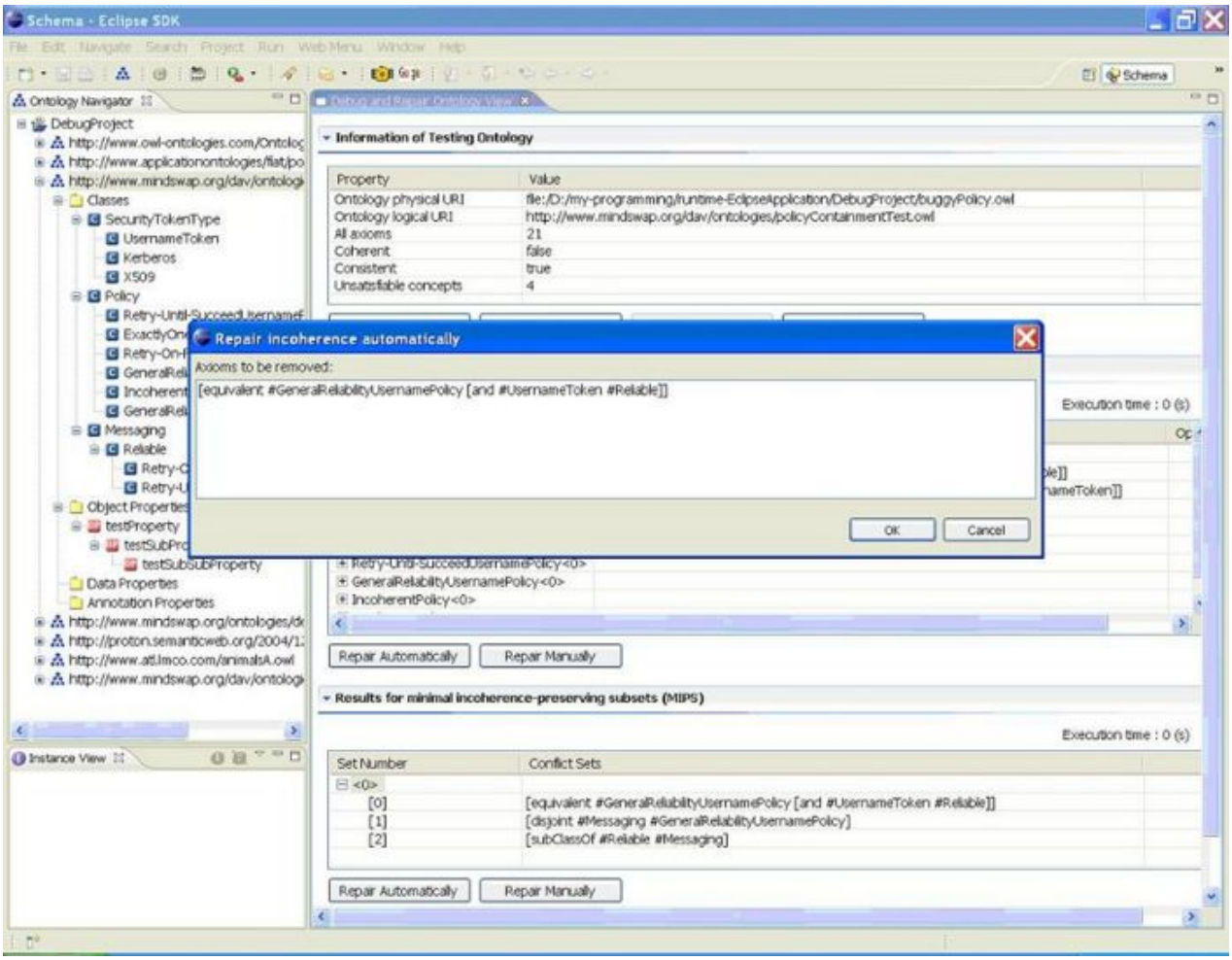


Figure 28: Automatic Repair of Ontologies

According to the obtained MUPS / MIPS / MIS, user can repair the ontology automatically or manually by using the corresponding button provided in each section. The proposed axioms to be removed to keep the coherence or consistency of the ontology are shown in a new dialog. In this new dialog, the proposed axioms will be removed from the test ontology is the button of "OK" is pressed. Otherwise, no action is performed for the button of "Cancel".

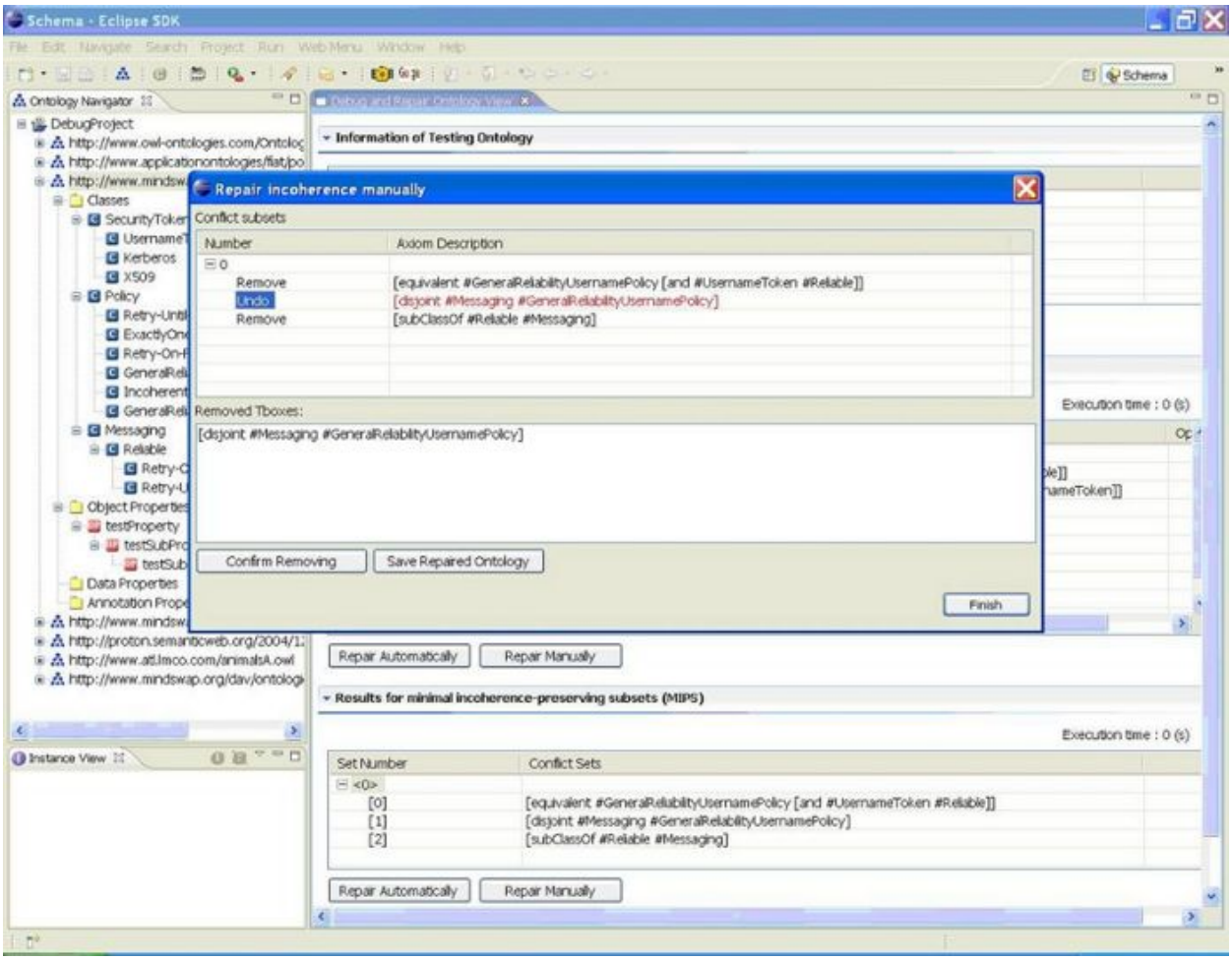


Figure 29: Manual Repair of Ontologies

If the button of "Repair Manually" in a section is pressed, a new dialog will be displayed to the user with MIPS if the button located in the section of MUPS or MIPS or with MIS if the button located in the section of MIS. In this new dialog, user could choose the axioms to be removed by clicking the label of "Remove" before an axiom. And the selected axioms are shown in the area of "Removed Axioms". User could withdraw his/her decision by clicking "Undo" label. After selecting the axioms to be removed, user can confirm his/her decision by clicking the button of "Confirm Removing" which will remove the selected axioms from the test ontology.

3.13.3 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.datamodel

3.13.4 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.

3.14 SAFE

3.14.1 Functional Description

SAFE (Semantic Annotation Factory Environment)is a software suite and a methodology for the implementation and support of annotation factories. An annotation factory is a process implemented by software engineers to deploy information extraction, one of the core technologies which makes semantic textual annotation feasible. The SAFE plugin is a web service enabling a user to perform semantic annotation on a set of documents and to view the results as an ontology.

3.14.2 User Documentation

Installation:

- download the JAR file
- put it in the plugins directory of your Neon Toolkit

Use:

- Click the SAFE button in the Neon Toolkit.
- Select the directory containing the texts to annotate
- Select the directory to place the output
- Click Run
- The resulting ontologies can be viewed in the Toolkit in the usual way.

3.14.3 Integration into the NeOn Toolkit

The figure below depicts how the SAFE-plugin interacts with the SAFE server (an external web server) as a loosely coupled component.

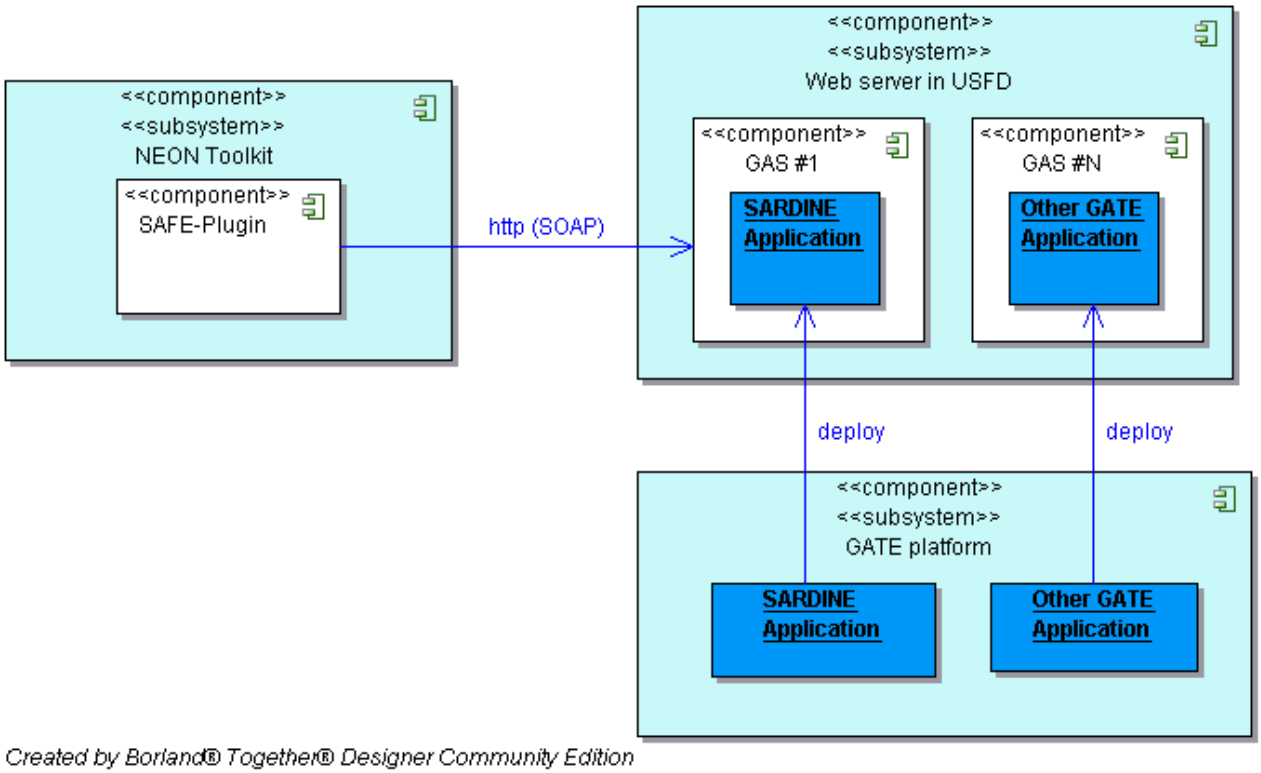


Figure 30: Architecture of the SAFE plugin

The plugin itself makes use of the following Eclipse Extension Points

org.eclipse.ui.actionSets

org.eclipse.ui.preferencePages

3.14.4 Intended Usage in the Case Studies

The SAFE plugin will be used in WP7 for ontology population and relation discovery. SAFE is a general purpose plugin that can invoke any predefined semantic annotation application using GATE, as a web service. Each instance of SAFE needs a set of texts to work on, and sometimes other task-specific resources such as ontologies. For the purpose of WP7, we have developed a SAFE application instance called SARDINE (Species Annotation, Recognition anD Indexing of Named Entities). It requires a corpus of texts (e.g. FAO's FIGIS factsheet corpus), and runs a semantic annotation application over the text, It enables the following:

- recognition of existing fish names from the FAO Species ontology
- identification of new potential fish to be added to the ontology
- identification of relations between both new and existing fish in the ontology, such as synonyms and hyponyms.

The result is a new populated ontology in OWL format, which contains the concepts from the Species ontology recognised in the text, term candidates suggested as new concepts to be added to the ontology, and relations between the two sets.

3.15 Text2Onto

3.15.1 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.

3.15.2 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.

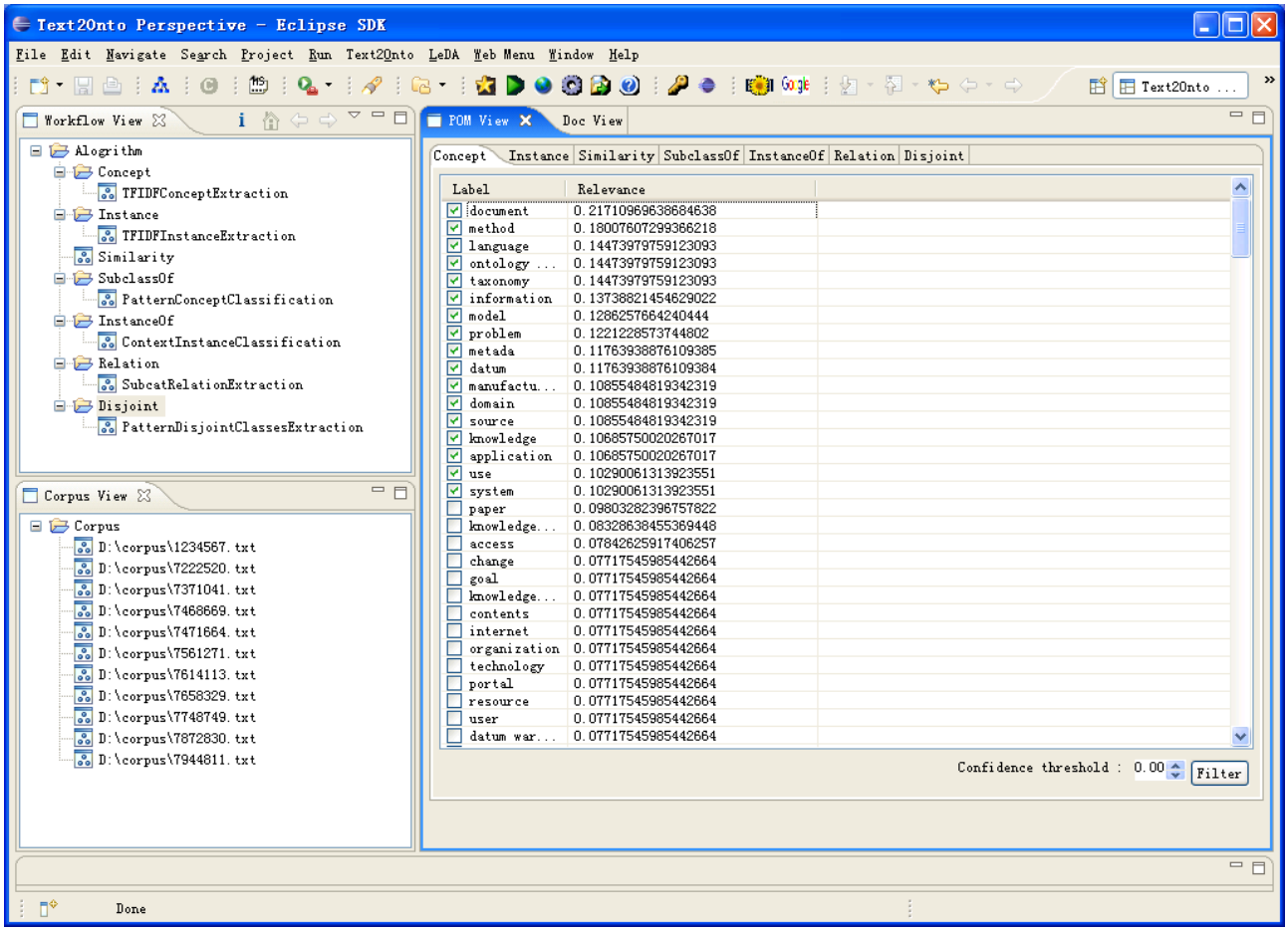


Figure 31: Text2Onto User Interface

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

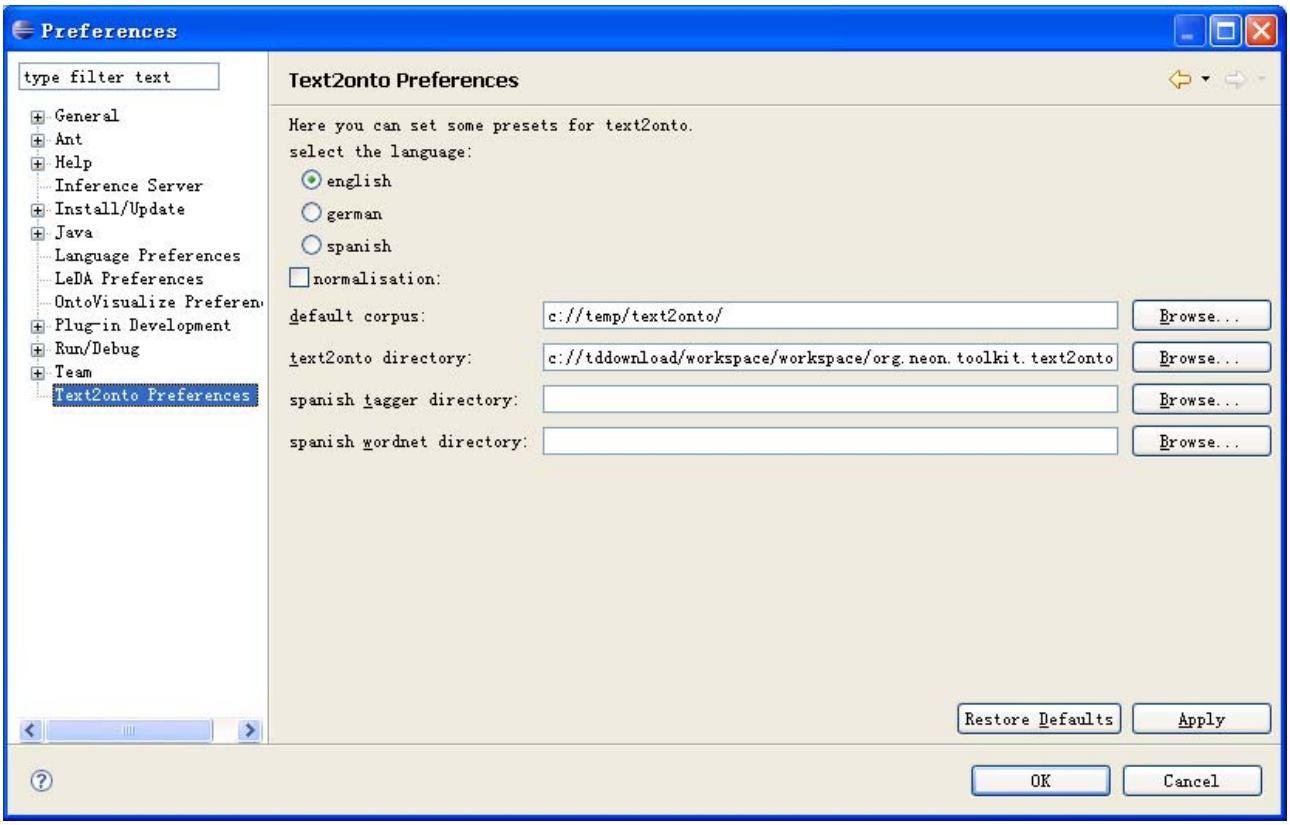


Figure 32: Text2Onto Preferences

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).

3.15.3 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

3.15.4 Intended Usage in the Case Studies

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

3.16 Watson for Knowledge Reuse

3.16.1 Functional Description

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.

3.16.2 User Documentation

To use the Watson Knowledge Reuse plugin, right click on the concept to inspect in the NeOn toolkit Navigator, and choose "Watson Search". A list of existing descriptions of the considered concept will appear in a a new view. Each statement in this list can be integrated to the currently considered ontology by using the corresponding button.

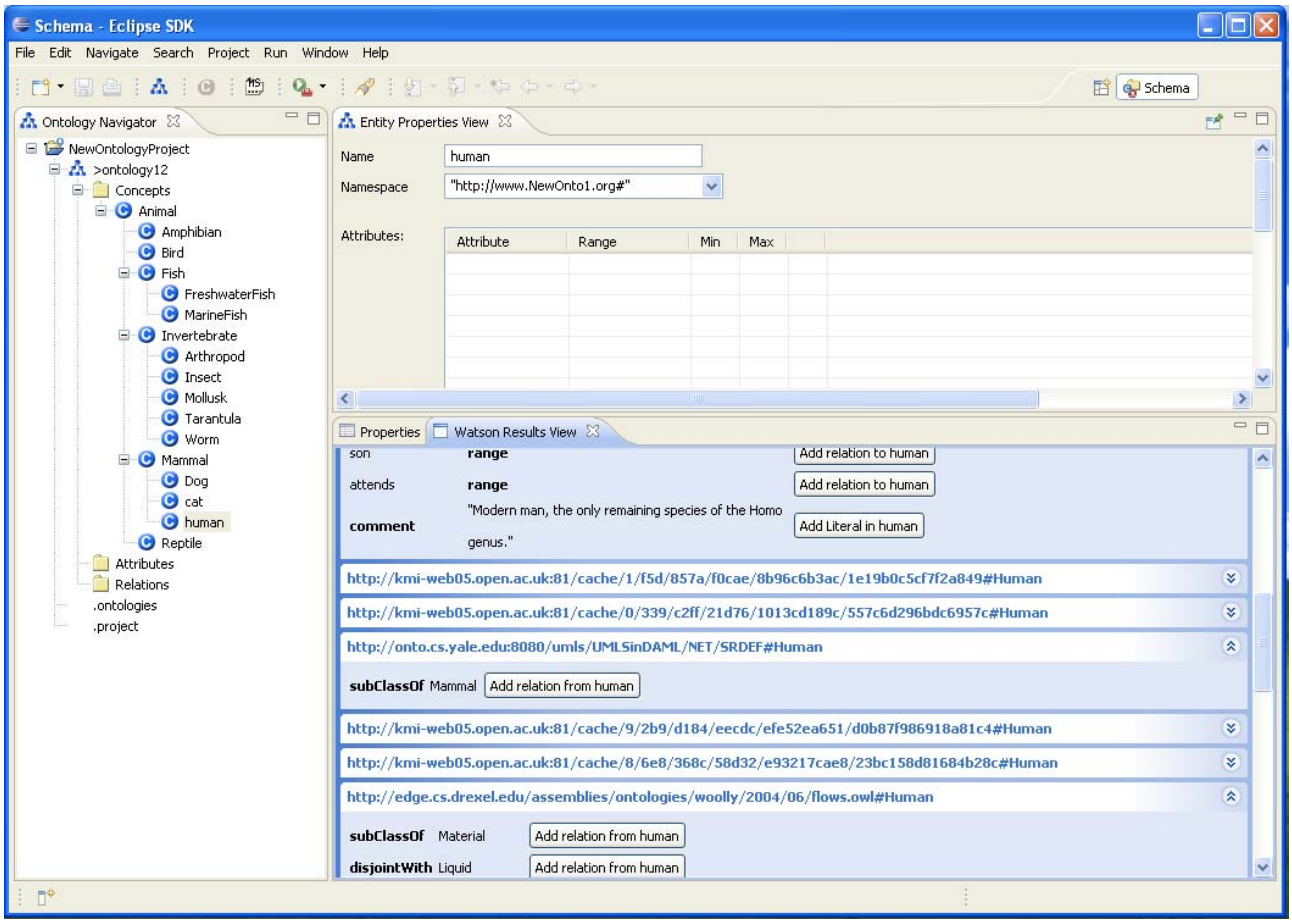


Figure 33: Screenshot of the Watson Plugin

Descriptions of Watson itself can be found on the Watson Website.

3.16.3 Integration into the NeOn Toolkit

Watson provides a new Eclipse view in the NeOn Toolkit. It relies or depends on the following plugins

- org.eclipse.core.resources
- org.eclipse.core.runtime
- org.eclipse.ui
- com.ontoprise.ontostudio.datamodel
- com.ontoprise.ontostudio.gui

Eclipse Extension Points

- org.eclipse.ui.views

- org.eclipse.ui.popupMenus

3.16.4 Intended Usage in the Case Studies

The Watson plugin is useful in any use case where an ontology is built or extended, and where the knowledge contained in existing ontologies that are available online can be reused. The advantage of using the Watson plugin is that it facilitates the selection of this reusable knowledge and its integration within the edited ontology. Also, by reusing elements of ontologies, the Watson plugin creates links between these ontologies, therefore encouraging and facilitating the creation of a network of ontologies. We expect this kind of feature to be useful in both WP7 and WP8.

3.17 WikiFactory Deployer

3.17.1 Functional Description

WikiFactoryDeployer provides the automatic generation of semantic wikis (based on Semantic MediaWiki platform) from ontologies.

3.17.2 User Documentation

Installation

Before to proceed with installation process make sure your system meets the following requirements:

- **Mediawiki (version 1.11)**
- **MySQL database (version >= 5.0.24)**
- **Web Server with PHP support (PHP version >= 5.1.5)**

To install the WikiFactory Deployer plugin just extract the WikiFactoryDeployer_1.0.0.zip archive into the plugin directory of your NeOn Toolkit installation path (eg. *C:\Programs\NeOn\plugins*).

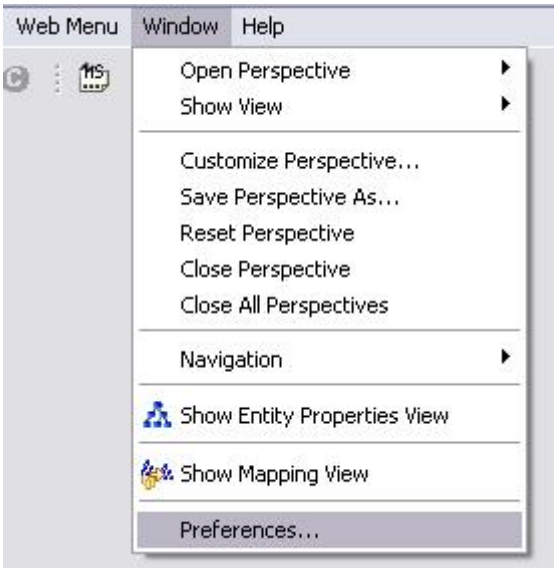
Configuration & Use

At the next start of the NeOn Toolkit you should have a new menu item and a new icon on the toolbar.

WikiFactory Deployer plugin needs some configuration information to work properly.

To access its preference page do as follows.

- Access the *Window* menu and select the *Preferences...* item



- select the “WikiFactory Deployer” item

A configuration page will be prompt.

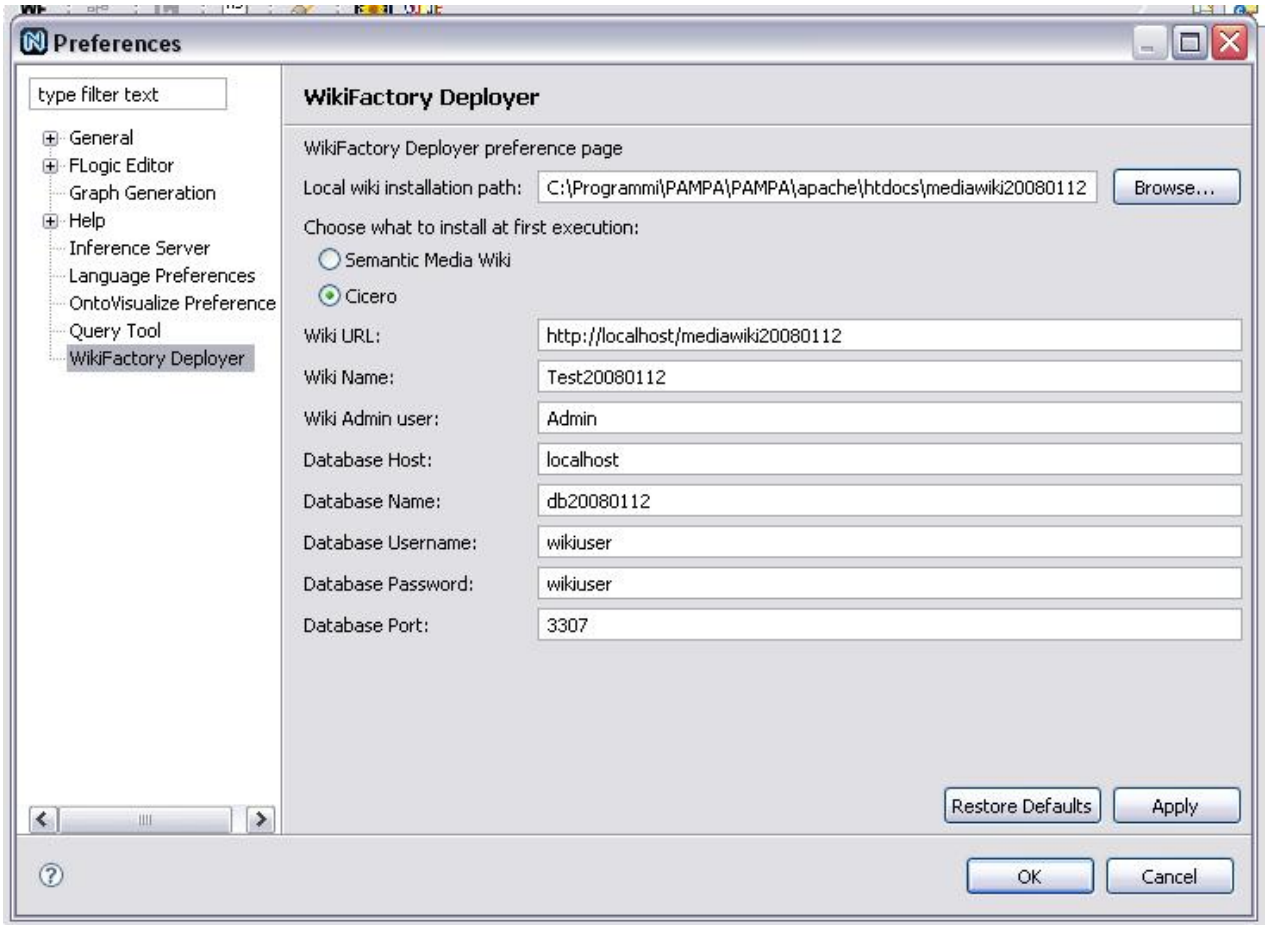


Figure 34: WikiFactoryDeployer Preferences

Here you have to provide the following information.

Local wiki installation path
It refers to the physical installation path of your Mediawiki wiki, and for example, if you use the Apache Web Server, it will looks like C:\Programs\Apache\htdocs\MyWiki

Semantic Media Wiki / Cicero Installation
With this radio button you can choose what to install at first time you run the WikiFactory Deployer plugin. First option will install the Semantic Media Wiki extension, while the second will install both Semantic Media Wiki and Cicero extensions. If you choose to install the Cicero extension, a Cicero admin account will be created for you using the Wiki Admin account.

Wiki URL
Here you have to provide the URL of your wiki

Wiki Name
Fill with the name you provided during the Mediawiki installation.

Wiki Admin User
You have to fill this textbox with the username of your Mediawiki Wiki Admin account

Database Host
If you are using a local installation of Mediawiki then probably also your database will be local. In this case just fill with the field with “localhost”, otherwise fill it with the IP of your database server.

Database Name
Fill with the database name provided during the installation of your Mediawiki wiki

Database Username
Fill with the database username provided during the installation of your Mediawiki wiki. Notice that the user you specify must have the rights to alter database tables.

Database Password
Again, fill with the password of the database user you provided during the Mediawiki installation.

Database Port
Fill with the port number of your database. A wrong configuration will deny the plugin to connect to the database.
When you are finished, click the *Apply* button.

How to deploy your first ontology

To deploy your ontology do the following:
1. Check your wiki is online (check then that your web server is running)
2. click on the WF toolbar icon or *WikiFactory* --> *Deployer* menu
3. a file dialog will be prompt: just select the OWL file you want to deploy and click *Open*.
4. an input dialog will ask you to provide the ontology default namespace.
5. Just wait that the deployment process completes.
6. When the deploy will be finished the main page of your wiki will be opened into the system default browser.

F.A.Q.

Q) What happens if I perform a new deploy to my Wiki?

A) Each time you do a new deploy the previous is deleted (means Wiki contents are deleted)

Q) What if I need to repeat the Semantic MediaWiki / Cicero installation process after the first execution?

A) To repeat it do as follows:

1) go to the WikiFactoryDeployer_1.0.0 plugin installation path and access the setup folder
2) delete the INSTALL.DAT file
3) go to your Mediawiki installation path and open the LocalSettings.php.
4) go to the end of the file and remove the following lines

```
include_once('extensions/SemanticMediaWiki/includes/SMW_Settings.php');
enableSemantics('localhost');
```

Remove also the following if Cicero extension is installed

```
include_once('extensions/DILIGENTArgumentationTool/DAT_DILIGENTArgumentationTool.php');
```

5) start the NeOn Toolkit, set your preferences from the plugin preference page and deploy your ontology.

3.17.3 Integration into the NeOn Toolkit

The plugin provides an easy wizard to deploy an ontology to a semantic wiki. The wizard is available from a new menu item ("*WikiFactory*") on the NeOn Toolkit toolbar.

To make WikiFactory Deployer works properly, users can define their settings through a preference page available from the default *Window->Preferences...* NeOn toolkit menu.

3.17.4 Intended Usage in the Case Studies

None yet.

3.18 XML Data

3.18.1 Functional Description

The plugin allows users of the Neon Toolkit

- to translate XML schema constructs into ontology classes, attributes and properties.
  During import of XML schemas the user can choose between the two following modes:
  - the order of XML elements within a parent element will not be preserved (simpler case with potential loss of information)
  - the order of XML elements within a parent element will be preserved (more complex case, although order is preserved in the ontology, it might not be visualized on the basic Studio, because this requires extended functionality)
- to access data that is stored in XML documents
  - statically during import process
  - dynamically from a rule using the XML documents URL (only supported in the extended version where rules can be executed)

The following constraints need to be considered:

- Mapping
  - XML schema distinguishes between attribute, element and type namesets. Within a set all names are unique, but names are not unique across these sets. For the ontology it is necessary to have different names for attribute/elements/types with the same name. Therefore, the mapping process transforms names as described below.
  - Mapping attributes
    - XML attributes will be mapped to datatype properties.
    - XML attribute names receive a '@' character as prefix in the generated datatype property name.
    - Facets of XML type definitions will not be mapped.
    - FLogic does not support the full set of XML schema types, thus the appropriate more general FLogic type will be used.

    - Mapping elements
        - XML simple typed elements will be mapped to datatype properties.
        - Globally defined elements that need to be mapped will be mapped to classes.
    - Mapping types
        - Complex types will be mapped to classes.
        - Generated class names will have "_type" postfix for mapped type names.
        - For anonymous XML types, a unique name is generated that contains the name that had been generated for the surrounding type.

- Each XML schema addresses exactly one target namespace. When making use of other namespaces, these can be imported. The XML importer creates separate ontologies for each of these target namespaces. The target namespace serves as ontology name. However, namespace imports can only be resolved when import statements in the XML schema specify the schemaLocation attribute.
- With many globally defined and referenced elements and types the resulting ontology may become extremely large. Therefore, the import wizard asks for a starting point, i.e. a globally defined element. XML documents to be imported need to have a root element of the select start element.

3.18.2 User Documentation

Basically, the XML import works like any other ontology import. Choose "Import..." from File menu or context menu. From dialog choose "OntoStudio" folder and there select the "XML import" node.

Next dialog asks for the XML schema file to be imported and offers a checkbox where user can select that ordering of XML nodes shall be represented. Please note, that representation of node orders can be expressed in the ontology, but cannot yet be fully visualized in studio because it makes use of parameterized properties. Users can proceed when clicking on the "Next" button.

Next dialog step asks for the project, where the generated ontologies are assigned to. Further, it asks for a root element, where the generation process shall start.

At this stage, the generation can be started when clicking on the "Finish" button. Optionally users can click on the "Next" button. In that case they can select an XML document to be imported.

3.18.3 Integration into the NeOn Toolkit

This feature provides an ontology import wizard. It does it in a way that is consistent with the NeOnToolkit Studio's import wizard. In addition it allows for import of an XML document immediately during the import process.

Technically speaking, the XML Import wizard first creates one or more FLogic ontology files and then uses internally the Studio's file import wizard.

- Relies or depends on the following plugins
    - com.ontoprise.ontostudio.datamodel

  - com.ontoprise.ontostudio.gui
  - com.ontoprise.ontostudio.io
  - org.eclipse.core.resources
  - org.eclipse.core.runtime
  - org.eclipse.ui
  - org.neontoolkit.gui
- Uses Eclipse Extension Points
  - org.eclipse.ui.importWizards

3.18.4 Intended Usage in the Case Studies

XML Schemata are found as legacy data sources to be integrated in all case studies. For these cases, the plugin may be of use.

4 Conclusions and Future Work

In this deliverable we have described the first set of plugins developed for the NeOn Toolkit by the partners of the NeOn project. The plugins already address a wide range of ontology lifecycle activities. As such, we hope that this set of plugins serves as a seed for the uptake of the NeOn toolkit and for the development of new plugins by the NeOn consortium and the NeOn community.

Certainly, with the now available plugins we are not at the end of the road yet. Based on observations in the case studies, we see the 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 example plugins to support the collaborative workflow for developing ontologies in the FAO case study. The deliverable describing the available plugins will be updated on an annual basis, with D6.10.2 at M36 and D6.10.3 at M48.

In the coming months, we will monitor the response to the plugins, both from within and outside the NeOn consortium. Based on the feedback, we may plan the development of additional plugins.

Furthermore, we will critically assess how well our quality assurance process is functioning. In particular, we will monitor feedback from users, discussions on the mailing lists and in the forum. A critical point will be the publishing of new versions of the NeOn Toolkit, which may lead to incompatibilities with existing plugins and thus require updated versions of the plugins. If needed, the QA procedure will be extended for the future development of plugins.

5 References

NeOnD531 Mari Carmen Suárez-Figueroa et al: D5.3.1 NeOn Development Process and Ontology Life Cycle, NeOn project deliverable, August 2007.

NeOnD621 Walter Waterfeld et al: D6.2.1: Specification of NeOn reference architecture & NeOn APIs, NeOn project deliverable, February 2007.

NeOnD741 Óscar Muñoz-García: D 7.4.1 Software architecture for managing the fishery ontologies lifecycle, NeOn project deliverable, August 2007.

NeOnD821 Jose Manuel Gómez-Pérez: D 8.2.1 Software architecture for the NeOn pharmaceutical case studies, NeOn project deliverable, February 2007.