NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D6.10.3 Updated NeOn Toolkit plugins

Deliverable Co-ordinator: **Andreas Harth**

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

Other Authors: **Alessandro Adamou (CNR), Sofia Angeletou (OU), Noam Bercovici (UKO-LD), Enrico Daga (CNR), Mathieu D'Aquin (OU), Klaas Dellschaft (UKO-LD), Jerome Euzenat (INRIA), Miguel Angel Garcia (UPM), Qiu Ji (UKARL), Diana Maynard (USFD), Victor Mendez (ISOCO), Elena Montiel (UPM), Enrico Motta (OU), Oscar Munoz-Garcia (UPM), Bostjan Pajntar (JSI), Freddy Priyatna (UPM), Walter Waterfeld (SAG), Faoud Zablith (OU)**

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

Document Identifier:	NEON/2009/D6.10.3/v1.0	Date due:	January 31, 2010
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	January 31, 2010
Project start date	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is 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 11 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.ump.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 SL–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 – 665 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 Kolenz-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. Marino 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 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:

- Caterina Caracciolo, FAO
- Michael Erdmann, ONTO
- Aldo Gangemi, CNR
- Juan Heguiabehere, FAO
- Felix Kiechle, UKARL
- Holger Lewen, UKARL
- Nadejda Nikitina, UKARL
- Simon Schenk, UKO-LD

Change Log

Version	Date	Amended by	Changes
0.1	21-11-2009	Andreas Harth	Initial version
0.2	18-01-2010	Individual plugin developers	Updated plugin descriptions
1.0	29-01-2010	Andreas Harth	Addressed reviewer comments

Executive Summary

The NeOn Toolkit is an extensible Ontology Engineering Environment and serves as the reference implementation of the NeOn architecture. The Toolkit integrates functionalities common to current 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 its latest incarnation (version 2.3) the NeOn Toolkit core relies on the OWL API as the underlying data model.

In deliverable D6.10.1[ea07b] we have described the first set of plugins that have been implemented by NeOn partners for the NeOn Toolkit. In deliverable D6.10.2[ea08] we have provided an update of the available partner plugins and have described plugins that had been developed either completely new or had been considerably extended. Both deliverables were based on the now obsolete KAON2 API as data model.

The plugins covered in this deliverable are all based on the OWL API. This deliverable contains a description of the final set of plugins available for the NeOn Toolkit version 2.3. As part of the new release we have coordinated two Code Sprints (meetings where plugin developers collectively work on code), streamlined the plugin build and code distribution process as well as the quality assurance procedure based on reviews conducted by the partners.

Contents

Chapter 1

Introduction

The NeOn Toolkit is an extensible Ontology Engineering Environment and serves as the reference implementation of the NeOn architecture.

Here we give a brief overview on the NeOn architecture. For an in-depth discussion on the architecture we refer the interested reader to previous deliverables [ea07b] [ea08].

1.1 Architecture and plugins

The general architecture of NeOn is structured into three layers:

- The infrastructure services layer contains the basic services required by most ontology applications.
- The engineering components layer contains the main ontology engineering functionality realised on the infrastructure services.
- The user interface components layer comprises front-ends for both engineering components and infrastructure services.

The NeOn architecture relies on the architectural concepts of the Eclipse platform[1]. The Eclipse IDE (integrated development environment) provides both user interface components and 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 modularised applications (plugins), a workbench model and the base for graphical components.

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 directly in the Eclipse platform but via 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 realised in plugins. This stresses the modular character of Eclipse, which follows the philosophy that "everything is a plugin". A plugin itself can be extended by other plugins.

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).

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

[1] `http://www.eclipse.org/`

initial version in [ea07a] - is organised along a set of ontology life-cycle activities. As part of WP10, we provide the training activities to disseminate a working knowledge of the Toolkit according to the NeOn methodology. Finally, the case study workpackages use the Toolkit and plugins within the case study prototypes.

In addition, a couple of plugins (gOntt and Kali-Ma) manage plugin configurations and allow for combining plugin use to form more complex workflows, targeted at project management (gOntt) and design patterns (Kali-Ma).

Chapter 2

Plugin development process

The plugin development process in general is a very open and decentralised one: anybody is free to develop and publish their plugin, and to make the plugin available under the conditions they prefer. Yet to ensure a coherent development and coordination among developers within the NeOn project, we have set up a basic process for plugin development that includes guidelines addressing aspects such as source code management, licensing and quality assurance.

2.1 Source code management

The default assumption for plugins developed within the NeOn consortium is that the plugins are available under open source licenses. Plugin developers are free to use the source code management system (SCM) of their choice. The default SCM for NeOn plugins is hosted at Google Code[1].

2.2 Licensing

The NeOn Toolkit in its open source version is distributed under the Eclipse Public License (EPL). In order to avoid conflicts in bundling the plugin with the NeOn Toolkit, we therefore recommend the EPL as the default license to plugin developers. The EPL is an open source software license used by the Eclipse Foundation. We deem the EPL as appropriate, as every plugin to the NeOn toolkit by its nature is also 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 copyright 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.

However, some plugins are licensed under the GPL as they contain code published under GPL, and due to the viral nature of the GPL, the derivative works have to be licensed under the GPL as well. To this end, a user has to use the update mechanism and separately confirm the GPL before downloading and installing the plugin.

In addition, there are plugins which are developed by SAG which are closed source but made available free-of-charge to NeOn partners.

2.3 Download site: plugins as Eclipse Features

To facilitate the deployment of new plugins, we make use of the Eclipse Update mechanism which allows for deploying and updating new features. Features are a concept of Eclipse to represent units of useful sets of functionalities. The role of features is to allow providers to make collections of plugins that logically go

[1] `http://code.google.com/p/neon-plugins/`

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.

For the NeOn Toolkit, we created a dedicated NeOn Update site referenced in the NeOn Toolkit core. 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.

2.4 Plugin wiki

We maintain a plugin wiki[2] as a central entry point for information about available plugins.

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 brief description of the functionality of the plugin.

2.5 Plugin documentation

In addition to the plugin wiki, all plugins provide a user documentation integrated into the Eclipse help infrastructure, which allows for documenting plugin functionality in a standardised manner.

2.6 Code sprints

To facilitate the migration of plugins to the new data model based on the OWL-API (v3), we have conducted two Code Sprints at Karlsruhe where plugin developers from partners met and worked on plugins. Members from Ontoprise were present to immediately address questions and comments from the developer community.

The first Code Sprint was held from August 19-20, 2009 at UKARL with a focus on plugin migration. Developers from ISOCO, ONTO, OU, SAG, UKARL and UPM attended. The key outcomes were:

- Plugins migrated: OWLDoc (UPM), Watson (OU), OWL OntoVis (ISOCO), Reasoning (UKARL)
- Migration started: Label Translator (UPM), ODEMapster (UPM), Module Extraction (OU), I2Ont (ISOCO), Radon (UKARL), SAG
- Improvements and fixes for NeOn Core (ONTO)
- Subversion Code repositories for NeOn core[3] and several plugins[4] (ONTO, UKARL)
- Update site for OWL API plugins
- Watson/gOntt integration
- Several new and updated documentation pages

The second Code Sprint was held at ONTO and UKARL from December 7-10, 2009 with a focus on core functionality. Developers from ISOCO, ONTO, OU, and UKARL attended. The key outcomes were:

- Improvements and fixes for NeOn Core (ONTO, OU, ISOCO)
- Identification of bugs and recording in Bugzilla (all)
- Work on reasoning and SPARQL plugins (UKARL)

[2] http://neon-toolkit.org/
[3] https://neon-toolkit.svn.sourceforge.net/svnroot/neon-toolkit
[4] http://code.google.com/p/neon-plugins/

2.7 Quality assurance and bug management

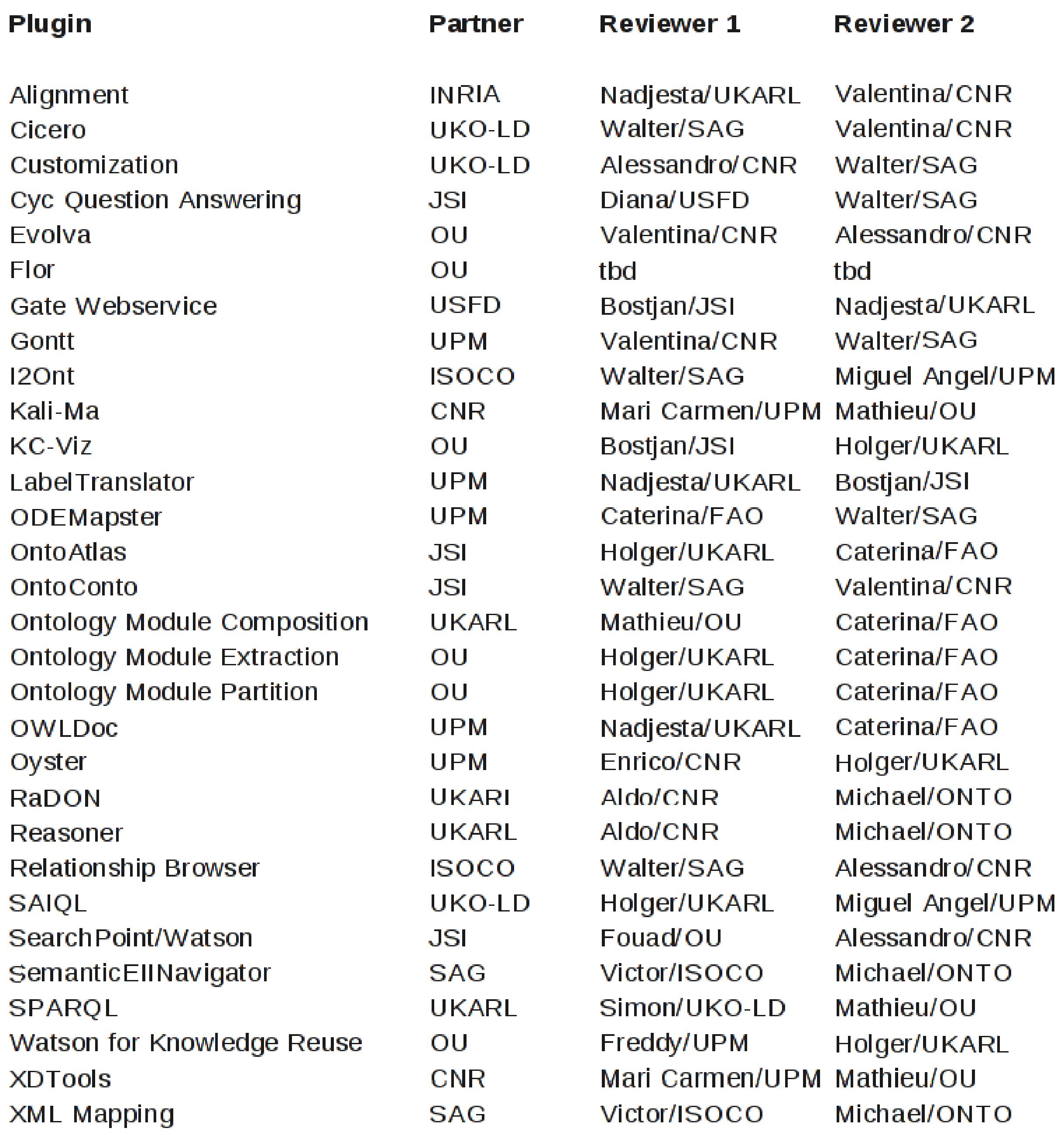
In order to ensure a professional appearance, we have established a review process that every plugin needs to meet before being officially published.

Before a plugin is officially published and made available to the NeOn community, every plugin is formally reviewed by at least two persons from within the NeOn consortium. Figure 2.1 lists the assigned set of reviewers. The review is done using a review form, which covers the following aspects:

- Installation: is the plugin findable/installable?
- Functionality: does the plugin work as expected? Please submit bugs you might encounter at Bugzilla and add here the bug number.
- Usability: is the plugin functionality easily accessible? Does the user interaction make sense? What suggestions do you have for improving usability?
- Documentation: is the plugin documentation available using the Eclipse Help function? Is the documentation sufficient to use the plugin?
- Integration: does the plugin integrate with gOntt and Kali-ma?
- Other comments/suggestions?

To facilitate communication between plugin users and developers we employ a bug management system based on Bugzilla[5].

[5] http://www.neon-toolkit.org/bugzilla/



Plugin	Partner	Reviewer 1	Reviewer 2
Alignment	INRIA	Nadjesta/UKARL	Valentina/CNR
Cicero	UKO-LD	Walter/SAG	Valentina/CNR
Customization	UKO-LD	Alessandro/CNR	Walter/SAG
Cyc Question Answering	JSI	Diana/USFD	Walter/SAG
Evolva	OU	Valentina/CNR	Alessandro/CNR
Flor	OU	tbd	tbd
Gate Webservice	USFD	Bostjan/JSI	Nadjesta/UKARL
Gontt	UPM	Valentina/CNR	Walter/SAG
I2Ont	ISOCO	Walter/SAG	Miguel Angel/UPM
Kali-Ma	CNR	Mari Carmen/UPM	Mathieu/OU
KC-Viz	OU	Bostjan/JSI	Holger/UKARL
LabelTranslator	UPM	Nadjesta/UKARL	Bostjan/JSI
ODEMapster	UPM	Caterina/FAO	Walter/SAG
OntoAtlas	JSI	Holger/UKARL	Caterina/FAO
OntoConto	JSI	Walter/SAG	Valentina/CNR
Ontology Module Composition	UKARL	Mathieu/OU	Caterina/FAO
Ontology Module Extraction	OU	Holger/UKARL	Caterina/FAO
Ontology Module Partition	OU	Holger/UKARL	Caterina/FAO
OWLDoc	UPM	Nadjesta/UKARL	Caterina/FAO
Oyster	UPM	Enrico/CNR	Holger/UKARL
RaDON	UKARL	Aldo/CNR	Michael/ONTO
Reasoner	UKARL	Aldo/CNR	Michael/ONTO
Relationship Browser	ISOCO	Walter/SAG	Alessandro/CNR
SAIQL	UKO-LD	Holger/UKARL	Miguel Angel/UPM
SearchPoint/Watson	JSI	Fouad/OU	Alessandro/CNR
SemanticEIINavigator	SAG	Victor/ISOCO	Michael/ONTO
SPARQL	UKARL	Simon/UKO-LD	Mathieu/OU
Watson for Knowledge Reuse	OU	Freddy/UPM	Holger/UKARL
XDTools	CNR	Mari Carmen/UPM	Mathieu/OU
XML Mapping	SAG	Victor/ISOCO	Michael/ONTO

Figure 2.1: List of plugin reviewers

Chapter 3

NeOn Toolkit Plugins

Here we provide descriptions of the individual plugins that are part of version 2.3 of the NeOn Toolkit. A current list of plugins can be accessed via the update mechanism in the NeOn Toolkit and are listed on the NeOn Toolkit wiki[1].

3.1 Alignment

The Alignment Plugin allows one to automatically compute and manage ontology alignments. More precisely, the Alignment 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

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

3.2 Cicero

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

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

[1]`http://neon-toolkit.org/wiki/Neon_Plugins`

- First, there are activities during which argumentation data is actively created, e.g. by discussions between the participants. In this case, the argumentation framework has the role of structuring the discussion process, helping in systematically exploring possible solutions and capturing the pro and contra arguments. Argumentation support is then a means of having more efficient discussion and decision taking processes. The most important activities of an ontology engineering project during which discussion data may be actively created are the ontology specification, ontology conceptualisation, ontology formalisation and ontology implementation phases.

- Second, there are activities where previously recorded discussions are used for understanding the design rationale of elements in the ontology network. Besides the previously mentioned activities, this may be especially useful during reusing or re-engineering ontological resources, or during the alignment with other ontologies.

The recorded discussions help in keeping track of the design rationale all the way through the ontology engineering process, and keeping the design rationale up to date by amending it with additional arguments. In general, supporting the argumentation process is important in each situation where either several users collaboratively decide an issue or where a user by himself creates an ontology element that should be later used as input for the activity to be developed by another user. In the latter case, the collaboration is facilitated by enhanced and more complete ontology documentation.

3.3 Customization

The main purpose of the ontology customization plugin is to propose to an ontology engineer using the NeOn Toolkit a simple way to define a customised ontology view corresponding to his need. The plugin perspective integrates the ontology relation browser from Isoco by default. While the user browses the ontology he can select classes or properties which might be relevant for him. Then the user can easily define a customised view of the ontology from those classes or properties by using the template mechanism.
Altogether, the plugin supports the following functionalities:

- A filtering mechanism using the annotation properties and/or the datatype properties;

- A customised ontology view builder which proposes to the user a friendly user interface to help him to compose the view definition;

- A query engine for the query language for OWL ontology SAIQL, the plugin gives the possibility to execute a SAIQL query directly.

A natural time to use the plugin is when the ontology engineer is preparing an ontology for a particular application. He may have decided to view this ontology in a certain way, either by filtering via language label, or, by extracting an ontology related to a specify set of axioms.

3.4 Cyc Question Answering

the Cyc plugin enables answering questions given in natural language based on a collection of documents. The user provides a collection of documents from the domain of interest that will be used for answering questions. We have used ASFA abstracts as a collection of documents and ASFA thesaurus in addition to Cyc and WordNet to provide semantic information in the form of synonyms and generalisations of the terms that occur in the documents. AnswerArt technology is used for answering questions provided in natural language. The plugin provides answers to the questions using Cyc and provides contextualised visual representation of the results.

The main goal of the Cyc plugin is to provide question answering functionality in a contextualized way. In general we derive answers from a repository of textual documents in the form of triples. The triples get semantically enhanced using Cyc Knowledge Server and other domain ontologies. This also provides contextualisation of knowledge extraction, since different ontologies enhance different triples. The AnswerArt module provides functionality of question answering, transforming natural language questions into triples queries, visualising and summarising the results.

3.5 Evolva

The Evolva plugin is an ontology evolution tool, which evolves and extends ontologies by identifying new ontological entities from external data sources, and produces a new version of this ontology with the added changes. A key feature of Evolva is that it relies on background knowledge to link and evaluate entities to be added to the ontology. The user starts by selecting the ontology to evolve. Then a list of classes is visualised, with the ability for the user to select which concepts should be included in the evolution. This is helpful in case the user wants to evolve only part of a big ontology. The user then specifies the domain data source (e.g. text corpus) that Evolva processes and identifies new domain concepts, with the ability to manually select or avoid those to potentially integrate in the ontology. Evolva exploits online ontologies and WordNet to identify links between new concepts and existing concepts in the ontology. Such links are displayed to the user in the form of statements, with the corresponding complete path derived from the source of background knowledge. The user then chooses the statements that needs to be added to the ontology, and finally decides if the changes should be applied directly on the ontology, or create a new detached version. The new ontology automatically appears in the NeOn Toolkit ontology editor, ready to be used.

In terms of the ontology engineering lifecycle, Evolva fits at the level of evolving and updating the ontology to keep it aligned with the represented domain. Ontology evolution being a tedious and time consuming task, our plugin aims to decrease the load and aid ontology maintainers responsible of such a task. After having built a basic ontology, the ontology engineer can use Evolva to identify and integrate new concepts that arise in the domain during the ontology lifecycle, without having to go through existing data that describe the domain in the form of more traditional data representations such as text documents.

3.6 Flor

The FLOR plugin adds to the NeOn Toolkit the functionality of folksonomy enrichment which is part of the Non Ontological Resource Re-engineering activity.

The plugin implements the FLOR algorithm which creates a semantic description for an input folksonomy tagspace. This is automatically created by using formal knowledge from different sources. The FLOR plugin allows the user to load a tagspace, select the knowledge sources among a number of ontologies and produces a semantic structure containing the tags senses and their relations. In addition the FLOR plugin provides visualisation and browsing functionalities.

3.7 Gate Webservice

The GATE web services plugin makes use of various GATE Services which perform annotation on text. The client supplies text and the service responds with semantic annotation relative to the specific ontology. The plugin contains 4 different applications:

- SARDINE (Species Annotation and Recognition and Indexing of Named Entities)
- SPRAT (Semantic Pattern Recognition and Annotation Tool)
- ANNIE (Ontology Generation Based on Named Entity Recognition)

- TermRaider (Automatic Terminology Extraction Tool)

In the case of the first 3 applications, the tool produces an ontology according to the important concepts, instances and relations found in the text. In the case of TermRaider, the tool produces a set of relevant terms from the text.

The plugin is designed for any NeOn Toolkit user who wishes to acquire ontological information (concepts, instances etc.) automatically from textual sources (e.g. web pages), or to acquire terms from text which can offer a starting or reference point for a human ontology engineer to develop their ontology, or which can be used as input to other automatic systems such as ontology refinement. The results of the applications may be integrated with existing ontologies by means of the change logging mechanism, or by means of other plugins such as ontology merging.

3.8 gOntt

gOntt plugin provides two main functionalities: one for scheduling ontology projects and one for helping in the execution of ontology projects.

In the case of the first functionality, gOntt provides support to ontology developers (a) to decide which ontology lifecycle model is the most appropriate for building their ontologies (e.g., waterfall, iterative-incremental), which processes and activities should be carried out and in which order (e.g., specify ontology requirements before re-engineering a knowledge-aware resource into an ontology) and (b) to create a graphical representation of the schedule in the form of a Gantt chart with the processes and activities needed, including time restrictions between them. Schedules for ontology development projects can be created either from scratch or in a guided way.

- In the first case, gOntt allows the developer to include processes, activities, phases, and relationships and restrictions among them according to his/her needs.
- In the guided way, gOntt creates a preliminary plan for the ontology development using templates for scheduling ontology projects and a simple two-step wizard that contains simple and intuitive questions that implicitly allow the ontology developer to select the ontology life cycle model and the processes and activities needed in his/her development. gOntt main output is the initial plan for building the ontology in the form of a Gantt chart that the developer can modify later on (a) by including, modifying, or deleting processes, activities, and phases, (b) by changing order and dependencies among processes and activities, and (c) by including resource assignments and restrictions to the plan.

In the case of the second functionality, gOntt provides prescriptive methodological guidelines by means of: (1) a filling card that includes the process or activity definition, its goal, inputs and outputs, who carries it out, and when it should be done, and (2) a workflow describing how the process or the activity should be done with its inputs, outputs, tasks and actors involved. Workflows are implemented with Eclipse Cheat Sheets. In addition, gOntt provides a direct access to NeOn Toolkit plugins associated with each process and activity. This means that gOntt triggers the different NeOn Toolkit plugins associated to each process or activity included in the plan.

Thus, gOntt plugin can be used in the scheduling activity as a schedule tool and along the whole ontology development process as a meta-tool with the aim of helping the ontology developer to carry out a particular process or activity.

3.9 I2Ont

The i2Ont plugin manages the mappings between one invoice and the ontology (in this case the PharmaInnova ontology). It creates instances in the ontology that correspond to invoices and all its fields and can transform files from one model format to another. The main features to highlight are:

- Management modelling of invoices about ontologies
- Create invoices as instances of ontology.
- Conversions between different types of invoices across ontology instances.

These files model the relations between the invoices and the ontology (currently OWL DL). We can perform invoice imports, creating its corresponding instances in the ontology and, also, it can be exported to other invoice formats through other configuration file mappings.

This tool is designed for non ontological resource re-engineering, so it is useful for retrieving and transforming an existing invoice (currently only CSV format is supported) into an ontology and vice versa. With i2Ont plugin we add the following translation process: The supplier models his invoice with the ontology, and the buyer does the same. Then the supplier can convert his/her invoice into the ontology with his/her mapping configuration and the buyer converts the instances of the ontology on his/her buyer invoice with his/her buyer mapping configuration.

3.10 Kali-ma

The Kali-ma plugin supports NeOn Toolkit users in the discovery, organisation and execution of functionalities implemented by other plugins in a running instance of the NeOn Toolkit platform. It offers an alternative interaction mode to the standard Eclipse UI for managing the organisation and usage of plugins. Additionally, it provides tools for Java developers and ontology specialists to achieve integration and interoperability of their plugins in the Kali-ma environment. The main functionalities of Kali-ma are the following:

- Browse the set of NeOn Toolkit plugins, either installed or known to exist, in an organised tree-like or graph-like view.
- Find out which plugins installed on the system can execute an activity, implement a functionality or cover a design aspect, and pick a list of favourite tools which will be represented by widgets.
- Browse access methods for a single plugin and execute them, or let Kali-ma choose for the user.
- Store a set of favourite plugins as a profile and associate it to one or more ontology development projects for sharing among users.
- Search a project or the entire workspace for data and metadata such as ontology entities, argumentation sessions and database schema mappings.
- Allow developers to extend their plugins so that their widgets can execute some of the plugin functionalities straight from the Kali-ma UI, and trigger the execution of these functionalities either by retrieved project (meta)data or the output produced by other plugins.
- Support real-time argumentation of project sessions via an online chat-board.

These functionalities are transversal to the entire ontology lifecycle management, as the Kali-ma plugin was not designed to support specific activities. Its role instead, is to organise and access functionalities with respect to the activities that must be performed. Even metadata management and argumentation, in this context, are considered project-centred instead of ontology-centred. Having an ontology engineering plan as the result of a scheduling activity, Kali-ma can facilitate engineers in executing any of the scheduled activities at a given point in time or during a given lifecycle management phase. Therefore, Kali-ma can support control phases for ensuring that all activities in an ontology development process are completed in the manner intended, and in doing so, it relies upon the plugins designed for performing these activities.

3.11 KC-Viz

The user studies carried out at the beginning of the NeOn project clearly indicate that the user interaction metaphors used in ontology engineering toolkits are largely inadequate, especially for those users with limited experience. Hence, in our work on Human-Ontology Interaction we aim to overcome these problems and develop novel interactive frameworks for visualising and navigating large and complex ontologies. To this purpose we have designed and implemented an innovative solution, based on the idea of identifying 'key concepts' in ontologies and using them as starting points for exploring and making sense of large ontologies. Key concepts can be seen as descriptive ontology elements that best summarise what a particular ontology is about. In our work we have defined this notion precisely and realised an algorithm for key concept extraction from ontologies, which combines theories from the cognitive sciences with formal methods from linguistics and ontology engineering. This innovative ontology summarisation method provides the basis for developing a novel approach to visualising and navigating ontologies. In particular, it enables 'middle-out ontology browsing,' where it becomes possible to move through complex information spaces from the most valuable nodes (i.e., key concepts) and then to unfold larger chunks of the ontological graph to inspect specific subparts of an ontology. This approach is similar to map-based visualisation and navigation in Geographical Information Systems, where, e.g., major cities are displayed more prominently than others, depending on the current level of granularity. Empirical user tests will be carried out to validate the hypothesis that this approach can speed up the process of making sense of an ontology, a key precondition for ontology reuse.

KC-Viz can be used to support a number of activities in the NeOn ontology engineering life-cycle. In particular, it can be used for ontology summarisation, ontology evaluation and assessment, especially with respect to informal user requirements. Finally, it indirectly supports ontology reuse, as making sense of an ontology (the key activity supported by KC-Viz) is a key precondition for ontology reuse.

3.12 LabelTranslator

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

Altogether, the plugin supports the following functionalities:

- **Obtains the most probable translation for each ontology label.**

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

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

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

- **Uses a synchronisation mechanism to maintain the ontology and linguistic information synchronised**

  Addition of new terms in the ontology or deletion of existing terms is controlled using the advanced change tracking (based on Resource Delta1) used in NeOn Toolkit. This mechanism is able to capture changes even when ontological terms have changed their position within the ontology model. By adopting this feature, LabelTranslator can accurately identify the minimal set of changes needed to adjust the structure of the linguistic model, a critical step to ensure that a matching change is made in the localised ontology

3.13 ODEMapster

ODEMapster plugin provides users a Graphical User Interface that allows to create, execute, or query mappings between ontologies and databases. The mappings are expressed in R2O language. This plugin supports a subset of the R2O language that includes the most used R2O primitives. This plugin works with OWL/RDF(S) ontologies and with MySQL or Oracle databases. Multiple R2O mappings per ontology can be created. ODEMapster is the processor in charge of carrying out the exploitation of the mappings defined using R2O, performing both massive and query driven data upgrade.

Ontology engineers perform the ontology population, annotation or aggregation from unstructured information sources activities with various manual or (semi)automatic methods to transform unstructured, semi-structured and/or structured data sources into ontology instances. Thus, ODEMapster plays the part where ontology population is performed from structured data sources (RDBMS).

3.14 Ontology Module Composition

The Module Composition plugin provides the functionalities to manipulate two modules. Specifically, it contains four main functionalities as follows:

- Compute the union between two modules by considering or ignoring namespace
- Calculate the intersection between two modules by considering or ignoring namespace
- Compute the difference between two modules by considering or ignoring namespace
- Align two modules

3.15 Ontology Module Extraction

The goal of the ontology partitioning plugin is to support the user in decomposing an ontology into smaller, more manageable modules with appropriate dependencies. It takes the form of a view within the environment of the NeOn Toolkit, which allows the user to select the ontology to modularise, specify some parameters for the partitioning algorithm and execute this algorithm. The result of the algorithm is then presented as a a graph showing the dependencies and the related size of the created modules, as well as a textual description of the content of each module. The user can then save the whole modularisation as a set of ontologies connected to each other through import links, or change to parameters to obtain a more suitable partition. Modules created from this plugin being ontologies, they can then be further tweaked using the module composition or the module extraction plugins.

The ontology partitioning plugin is useful mainly when reengineering existing ontological resources, making them more manageable and therefore more reusable in other ontologies. It can also be considered as an ontology re-factoring mechanisms.

3.16 Ontology Module Partition

The module extraction plugin provides an interactive and iterative approach to extracting relevant sub-ontologies from existing ones. It integrates a number of different “operators” for module extraction, most of them being relatively elementary (extract all the super/sub-classes of a given set of entities, all the other entities they depend on, etc.) The interface for this plugin allows the user to easily combine these different elementary operators in an interactive way. An initial module can be created, using particular parameters, obtaining an initial set of entities to be included. Then another operator can be used, on other entities and other parameters, to refine the module and extend it with other entities, until an appropriate module is created. At any point of the process, previous operations can be undone and the module cleared. Once a module is created, it can be saved as part of the current ontology project and become itself processable as an ontology (module), to be composed or partitioned using the other modularisation plugins.

As for the ontology partitioning plugin, the module extraction plugin can be used in ontological resources re-engineering and re-factoring, to reduce considered ontologies into their sub-part relevant for the task/ontology at hand. In addition, this plugin is useful in creating modular ontologies by reuse, reducing potentially big ontologies into a focused component that can be easily integrated and reused in an ontology network.

3.17 OWLDoc

The OWLDoc plugin adds to the NeOn Toolkit an option to export an OWL ontology as a HTML document. The plugin uses the OWL API to extract information from the OWL ontology and creates an output that contains an organised set of HTML files that provide the documentation about the ontology and all its resources.

3.18 Oyster GUI

Oyster is a distributed registry that exploits semantic web techniques in order to provide a solution for exchanging and re-using ontologies and related entities. As an ontology registry, it provides services for storage, cataloguing, discovery, management, and retrieval of ontology (and related entities) metadata definitions. To achieve these goals, Oyster implements the proposal for metadata standard OMV as the way to describe ontologies and related entities, supporting advanced semantic searches of the registered objects and providing services to support the management and evolution of ontologies in distributed environments.

Oyster GUI is an interface integrated into the NeOn Toolkit to ease access to Oyster servers. These servers act as repositories of OMV instances, thus allowing to store and to manage metadata about ontologies and related concepts. Main features:

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

3.19 OntoAtlas

OntoAtlas plugin visualises ontologies inside a context. First, the user can create a context out of non ontological textual resources. The plugin can, out of a repository of a documents, create a semantic landscape showing topics present in the corpus as higher ground in the landscape. Documents are also positioned on this landscape near the topics most relevant to them. Next, OntoAtlas visualises any ontology on top of this semantic landscape, providing zoom in and zoom out features, contextualising each ontology concept by its position on the landscape.

The plugin is used whenever there is need to understand an ontology in the context of some non ontological resources. For example, it can be used to select the best ontology to model a specific context. It can also be used to quickly check the coverage of an ontology and identify knowledge not yet modelled with the ontology. Lastly, it can offer help when populating an ontology. Individuals can be used for the creation of the context with the relative position between ontology concepts and documents giving insight to the correct annotation.

3.20 OntoConto

OntoConto is a plugin that visualises ontology alignment and offers functionality to edit, store and load alignments. To visualise the alignment, first two ontologies are partially visualised side by side. It is easy to navigate to a certain part of an ontology. Next, the user can request for automatic generation of an alignment between ontologies with several methods from the alignment server, or start a new manual alignment. Mappings of the alignment are visualised as strings spanning the two ontologies.

OntoConto can be a great help when engineering an ontology as it visualises it in the context of another networked ontology. An engineer can quickly understand the differences of coverage and structure between the ontology being worked on and the context ontology. Second, OntoConto is a great tool to help a domain expert create a manual alignment. The user can first check several automatic alignments, select the best one and manually edit it to remove any mistaken mappings, or add missing ones. OntoConto also hides all of the ontology notations making it easy to use for any user unfamiliar with ontologies.

3.21 RaDON

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

- Handle incoherence: This functionality corresponds to the button of “Handle Incoherence” which can be activated if the ontology is incoherent. That is, there is at least one unsatisfiable concept in the ontology. All the minimal unsatisfiability-preserving subsets (MUPS) can be computed for each unsatisfiable concept;

- Handle inconsistency: This corresponds to the button of “Handle Inconsistency” which is activated if the ontology is inconsistent. That is, there is no model for the ontology. All the minimal inconsistent subsets (MIS) can be calculated;

- Repair automatically: This corresponds to the button of “Repair Automatically”. If the button of “Repair Automatically” is pressed, our algorithm will provide some axioms to be removed to keep the coherence or consistency of the ontology. Specifically, this can be done by computing the minimal incoherence-preserving subsets (MIPS) or MIS respectively and then choosing automatically some axioms to be removed.

- Repair manually: This corresponds to the button of “Repair Manually”. If this button is activated, a new dialog will be shown with the information of MIPS or MIS which are computed based on the found MUPS or MIS respectively. The user could choose the axioms to be removed by themselves.

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

3.22 Reasoner

The reasoner plugin is an infrastructure plugin which provides access to Pellet[2] and HermiT[3] for other plugins relying on reasoner functionality.

3.23 Relationship Browser

This plugin adds a new visualisation paradigm to the NeOn Toolkit. The plugin provides a navigation based on the relations contained in the classes of the ontology.

- Visualise classes or individuals with its relations
- Graphically navigate through ontology classes or individuals

The Relationship Browser is a tool for viewing and navigating through the relations between ontology classes and individuals, and can be used throughout the entire life cycle as a visual aid. Also, during the documentation phase we can export the display to graphics files (png, jpg).

3.24 SAIQL

As one main contribution, SAIQL query language can handle class expressions and property names in addition to class names and individual names. The extraction of these class expressions is of major importance as they provide the definition for a certain class name. Instead of extracting isolated class names and individuals like in OWL-QL, the class names, class expressions, property names and individual names are returned contained in OWL DL axioms. Thus, the result is a fully working OWL DL ontology. The plugin is a SAIQL query engine and an editor to compose those queries.

The plugin can be use at the testing phase of the ontology engineering life cycle and of course at the exploitation time. During the testing phase the ontology engineer can easily find out modelling issues by looking at the result of his queries. During the exploitation the user can use the plugin to query the ontology and to retrieve part of the schema and its corresponding instances.

3.25 SearchPoint/Watson

SearchPoint plugin offers contextualised search for ontologies. It accesses several ontology search engines and offers a contextualising panel visualising different topics found in the search results. The user can change the ranking of the hits by moving a focus point near the topics of interest. Extracting topics from search results is greatly enhanced by Watson ontology search engine which is providing "key concepts" as a form of good summarisation of each ontology.

[2]http://clarkparsia.com/pellet/
[3]http://hermit-reasoner.com/

Ontology Search is one of the first steps in the ontology engineering lifecycle. When creating a new ontology, it makes sense to take a pre-existing ontology and reuse it whole or in part. As ontology search is not very accurate, the contextualising power of SearchPoint enables the user to find the needed ontologies by additionally selecting a context which positions the needed ontologies higher in the result set.

3.26 SemanticEIINavigator

The plugin allows users of the Neon Toolkit

- to visualise the integration process of data sources
- to have quick access to various tools related to the integration
- to investigate complex links between ontologies, queries and data sources

This plugin depends on the DataSources plugin in order to find data sources and which ontologies are imported. It recognises four types of data sources - database, XML, web services and Adabas. For additional data sources, the EII Navigator functionality should be extended by additional plugins. Importing XML schemas without a linked external document leaves no rule that would identify a data source ontology. That is why these ontologies are not marked as such. The Visualisation mechanism allows for overlapping of some nodes (ontologies, queries or data sources) inside the respective panels. This was allowed to decrease the complexity of the algorithm and the size of the horizontal scroll when reordering the elements. Adding menu items and actions to the drop down menu of each artefact in the EII Navigator view is done by external plugins. The Ontology Visualiser action is an example of one such extension. More should be added by the respective plugins.

3.27 SPARQL

The SPARQL Plugin allows a user to SPARQL queries over ontologies loaded into the NTK. The input to the plugin are an OWL ontology and a query expressed in SPARQL syntax; the results are query answers listed in a table. In addition to querying local ontologies, the SPARQL Plugin allows for evaluating queries over a Linked Data substrate, i.e. translates SPARQL queries to Linked Data lookups on the web.

The SPARQL plugin can be applied in the reuse activities, i.e. to find out about already existing ontological resources, and in the evaluation activity to test if specified queries can be answered using the modelled ontology.

3.28 Watson for Knowledge Reuse

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. Indeed, with this plugin, it is possible to discover, inspect and reuse ontology statements originating from various online ontologies directly in the ontology engineering environment. In addition to the Watson Semantic Web search engine, the Watson plugin can now also interrogate the Cupboard ontology publication environment, and specific ontology spaces in this environment. Options are also available to keep track of the element reused, automatically linking entities from the local ontology to external ones.

Two main activities in the ontology life-cycle have been identified where the Watson plugin is relevant. First, at the beginning of the ontology development process, immediately after specification, the plugin can be used to create an initial ontology, integrating various elements from external ontologies concerning concepts and relations identified as core for the ontology to be developed. This initial, skeleton ontology can then be refined manually, or by integrating additional knowledge reused from Watson or other sources. Second, the Watson plugin can be very useful in identifying elements to add to an ontology during the ontology evolution process.

3.29 XDTools

The eXtreme Design tools for NeOn Toolkit (XD Tools) offers the possibility to perform operations on ontology design patterns (ODPs) according to the eXtreme Design (XD) methods. For further details we refer to D2.5.2, "Pattern based ontology design: methodology and software support". The concept of networked ontology is native in XD. The main idea is to provide the user with pattern-based operations, such as specialisation, composition, and instantiation, according to the guidelines defined in D2.5.1 in the context of the XD methodology. The tool provides a perspective - “eXtreme Design” - that groups all the user components: the ODP Registry browser and ODP Details view, the XD Selector, and the XD Analyzer. Other components can be activated in several ways within the interface, i.e. the XD Wizards, the ODP Publish dialog, and the XD Annotation dialog.

- ODP Registry and ODP Details view: through this view users can have access to a set of reusable OWL patterns that can be directly browsed and exploited in the modelling process, without necessarily having them locally stored. When a pattern is selected from the registry tree, all OWL annotations are visualised in the ODP Details view;

- XD Selector: The aim is to give support to the end-user with a component for proposing patterns to be reused to cover requirements expressed as competency questions. The user can query these basic selection services providing either simple keywords or a complete competency question (CQ). The XD Selector view displays as result a list of proposed ODPs to be reused in the current working ontology. By clicking on a result, details (ontology annotations) are shown in the ODP Details view.

- Specialization Wizard: The Specialization wizard is the component provided by XD Tools for guiding the user in the specialisation of Content ODPs. The wizard can be accessed either from the Navigation view, i.e. through the contextual menu activated while right-clicking on an ontology, from the ODP Registry view or from XD Selector’s results view. When the process is finalised, a message is displayed with the option to open the dialog for ontology annotation, i.e. to annotate the new ontology module that has been created.

- XD Annotation dialog: It provides the facility to annotate the ontology using other vocabularies in addition to the annotation properties already provided by OWL/RDF. For the Content ODPs the CPAnnotationSchema[4] is supported[5]

- XD Analyzer: The aim of this tool is to provide suggestions and feedback to the user with respect to how good practices in ontology design have been followed, according to the XD methodology (for instance missing labels and comments, isolated entities, unused imported ontologies).

For more detailed information about functionalities, architecture and extendibility we refer to D2.5.2.
The XD Tools supports the following activities according to the NeOn methodologies:

- Ontology Annotation and Ontology Documentation: the XD Annotation Dialog provides facility for annotating ontology patterns. This task is particularly important for Content ODPs, since the annotations are the base documentation provided to the user.

[4]Published at `http://ontologydesignpatterns.org/schemas/cpannotationschema.owl`
[5]Ongoing work includes to support the possibility to add additional, user-defined, vocabularies through a preference page.

- Ontology Reuse: ontology design patterns can be accessed from the ODP Registry view and reused by cloning locally (the 'get' button in the user interface) and by specialisation (through the 'Specialization Wizard');
- Ontology Selection: ODPs exposed in the ODP Registry can be selected through the XD Selector component, that allows a quick search functionality and support multiple selection services.
- Ontology Diagnosis: the XD Analyzer provides end-user support for good practices in pattern based ontology design.

3.30 XML Mapping

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: i) the order of XML elements within a parent element will not be preserved (simpler case with potential loss of information) ii) 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 visualised on the basic Studio, because this requires extended functionality).
- to access data that is stored in XML documents i) statically during import process ii) dynamically from a rule using the XML documents URL (only supported in the extended version where rules can be executed).

Chapter 4

Conclusions

In this deliverable we have described the third and final set of plugins developed for the NeOn Toolkit by the partners of the NeOn project. Existing plugins have been updated to be compatible with the latest version of the NeOn Toolkit based on the OWL API and extended with new functionalities. New plugins have been developed to address previously unsupported ontology life-cycle activities. Further, the new plugins have been developed with an improved quality assurance process and include user documentation integrated into the NeOn Toolkit using the Eclipse help infrastructure.

Bibliography

[ea07a] Mari Carmen Suarez-Figueroa et al. D5.3.1 NeOn Development Process and Ontology Life Cycle. NeOn Deliverable 5.3.1, NeOn Project Members, August 2007.

[ea07b] Peter Haase et al. D6.10.1 Realization of core engineering components for the NeOn Toolkit. NeOn Deliverable 6.10.1, NeOn Project Members, February 2007.

[ea08] Peter Haase et al. D6.10.2 Realization of core engineering components for the NeOn Toolkit, v2. NeOn Deliverable 6.10.2, NeOn Project Members, November 2008.