NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D7.4.2 Prototype System for Managing the Fishery Ontologies Lifecycle

Deliverable Co-ordinator: **Yimin Wang**

Deliverable Co-ordinating Institution: **Universität Karlsruhe – TH (UKARL)**

Other Authors: **Claudio Baldassarre (FAO), Marta Iglesias (FAO) and Peter Haase (UKARL)**

This deliverable describes the first prototype for managing the fishery ontologies lifecycle following the architecture, requirements and use cases defined in D7.4.1. The lifecycle described in this deliverable is an instantiation of the possible lifecycles that can be carried out with the NeOn toolkit, and has been selected according to requirements from FAO. In this deliverable, after introducing different components, we provide an integrated view over isolated components developed by different partners within in NeOn project. Several integrated use cases are introduced to prove the applicability of this integration.

Document Identifier:	NEON/2008/D7.4.2/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 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 writing parts of this document:

- Open University (OU)
- Universidad Politécnica di Madrid (UPM)
- Food and Agriculture Organization of the United Nations (FAO)
- Ontoprise GmbH. (ONTO)
- Software AG (SAG)
- University of Sheffield (USFD)

Change Log

Version	Date	Amended by	Changes
0.1	05-03-2007	Yimin Wang	Creation
0.2	30-11-2007	Yimin Wang	Add Contents
0.3	10-12-2007	Yimin Wang	Organizing Contents
0.4	19-01-2008	Yimin Wang	Integrated Cases
0.5	23-01-2008	Yimin Wang	Introduction and Summary
0.6	27-01-2008	Yimin Wang	Next Steps
0.7	31-01-2008 31-01-2008	Claudio Baldassarre Peter Haase	Revision Revision
0.8	15-02-2008	Yimin Wang	Submit for Review
0.9	20-02-2008	Yimin Wang	Address Comments
1.0	29-02-2008	Yimin Wang Claudio Baldassarre Marta Iglesias Peter Haase	Finalizing Documents

Executive Summary

This deliverable describes the first prototype for managing the fishery ontologies lifecycle, a cornerstone of WP7. The lifecycle described in this deliverable is an instantiation of the possible lifecycles that can be carried out with the NeOn toolkit, and has been selected according to requirements from FAO. We aim to provide an instantiation of NeOn Toolkit to support the requirements of fishery ontology lifecycle management.

This deliverable follows the software architecture and use case descriptions from D7.4.1 [MGKI+07] in terms of requirements that should be fulfilled by the general NeOn lifecycle management support. The architecture described in D7.4.1 is based on the NeOn Toolkit and engineering components. The engineering components is incrementally integrated into the system through the first prototype introduced in this deliverable and future T7.4 deliverables, i.e., D7.4.3.

One major contribution of this deliverable is that we provide an integrated view over isolated components developed by different partners within in NeOn project. Several integrated use cases are introduced to prove the applicability of this integration:

- Managing networked ontologies in a decentralized network
- Consistency checking of ontologies populate from text
- Reusing ontologies populated from database

In general, the Chapter 1 of this deliverable gives an overall introduction to the motivation and challenge to realize the fishery ontologies lifecycle management system. Since delivering D7.4.1, the requirements about the fishery ontologies lifecycle have been evolved more concrete and have been explicitly outlined and refined in D7.4.2. Chapter 2 of this deliverable describes the major requirements of the fishery ontologies lifecycle management system based on the scope of the first prototype. Chapter 3 provides an architecture of this prototype about how the individual components are integrated into the prototype by aligning to the general NeOn architecture. Chapter 4 includes detailed description regarding individual components in this prototype within FAO's usage scenario. Chapter 5 presents some case studies about how the integrated components work together as a unified prototype for managing fishery ontology lifecycle. Chapter 6 describes the plan for subsequent T7.4 milestones and deliverables. Finally, Chapter 7 highlights the major conclusions of deliverable D7.4.1.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we discuss the scope of this deliverable, the motivation of our work, the approach adopted here and how this deliverable is organized.

1.1 Motivation

This deliverable is a part of the work in Workpackage 7 “ontology-driven fishery stock depletion assessment system” of the NeOn project. This deliverable describes the first prototype for managing the fishery ontologies lifecycle based on the architecture, requirements and use cases described in D7.4.1. The lifecycle described in this deliverable is an instantiation of the possible lifecycles that can be carried out with the NeOn toolkit [1], and has been selected according to requirements from FAO.

We are going to implement an instantiation of NeOn Toolkit with plug-ins from multiple partners to support the fishery ontology lifecycle management in FAO. This instantiation is not a new developed set of plug-ins, while the major efforts in deliverable is to identify the solutions to the requirements from FAO and try to find out appropriate plug-ins to support several integrated cases that happen in FAO’s daily work.

With the growing requirements of applying modern semantic technologies to information systems in big organizations like FAO, there have been many efforts to enable semantic data management. In past several years, FAO people have been trying to import/export ontologies to/from databases, editing ontologies using state-of-the art ontology editors and making the ontologies as backbones of the online information system in FAO for both internal an external usage.

However, none of the existing systems is developed and particularly tailored for FAO’s requirements. There are three major challenges to directly apply existing technologies and applications to FAO’s case:

- *Distributed networking scenario.* The data in the ontology-based applications in FAO is usually distributed rather than centralized, whereas the current applications developed within the Semantic Web community usually focus on centralized networking scenario.
- *Dynamics and interconnectivity.* On the one hand, the distribution of FAO data determines that the data are managed in an isolated manner. On the other hand, because the data sets are often interconnected with each other, when one data set is changed, the other data sets that are interconnected with this data set are also required to be changed accordingly. However, in distributed scenario, monitoring the dynamics of semantic data is extremely complicated and difficult to implement without applying networked ontology techniques.
- *Multilingual aspects.* As one important issue in FAO as an international organization, applications often serve a large international community and usually work with data from many places around the world that are naturally encoded in many languages. This multilingual issue is not commonly discussed in ontology research.

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

Let's have a look an example that is close to our scope: FIES and KCEW (different FAO departments) are now collaboratively developing semantic applications in the fishery domain within NeOn project [2] and they are sharing fishery data across departments which may locate around the world and are encoded in many languages. FAO people have been trying to edit and manage ontologies that are distributed and interconnected, but they find it very difficult to handle these physically isolated but logically interconnected ontologies. This difficulty makes FAO people unable to use and propagate their ontologies effectively and efficiently.

As the next generation ontology model, the networked ontology model [HRW$^+$06] aims to provide enhanced functionalities to handle dynamic and heterogeneous ontologies that are interconnected within a networking scenario. Therefore, it's possible to use networked ontologies to manage fishery data to achieve effectiveness and efficiency – this calls for a suite of fully functioned tools to edit, manage, search and propagate ontologies that are distributed, dynamic and multilingual.

1.2 Approach for Integration of Components

This deliverable follows the software architecture and use case descriptions from D7.4.1 [MGKI$^+$07] in terms of requirements that should be fulfilled by the general NeOn lifecycle management support. The original NeOn architecture is defined in deliverable D6.2.1 [WWH$^+$06]. The architecture described in D7.4.1 is based on the NeOn Toolkit and engineering components. The engineering components is incrementally integrated into the system through the first prototype introduced in this deliverable and future T7.4 deliverables, i.e., D7.4.3. We have not implemented all the required use cases introduced in D7.4.1 in this deliverable due to the limitation of software development procedure, but we plan to do this in next deliverable.

Let's first look at the general NeOn architecture before we introduce the integration we are going to adopt in this deliverable.

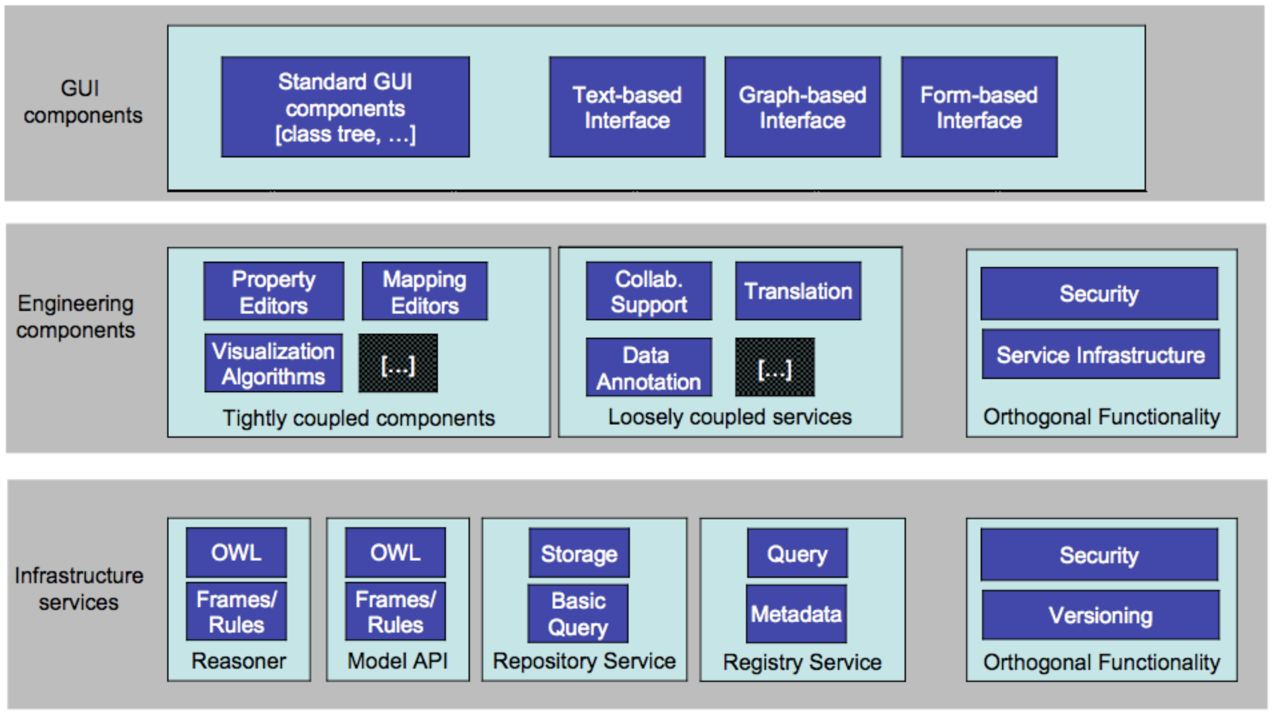


Figure 1.1: The NeOn general architecture

[2]NeOn is an EU IST FP7 project: `http://www.neon-project.org`

There are three layers in the NeOn Toolkit architecture, as displayed in Figure 1.1:

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

The integration approach that we are going to adopt has been proved to be applicable in previous Workpakcage 1 deliverable D1.4.1 [WHP07], which introduces an integrated architecture that includes Oyster, KAONp2p and KAONWeb components [WHP07]. There are several plug-ins in the Core NeOn Toolkit that is an essential plug-in for such an integration: Core Neon Toolkit – Datamodel plug-in. This plug-in represents the core plug-in of the NeOn Toolkit since it represents the internal datamodel and thus the knowledge representation aspect of the application. It provides access to the underlying datamodel via Java APIs and makes other external applications accessible to the internal ontologies in NeOn Toolkit ontology projects. We briefly introduce a typical procedure of component integration with NeOn Toolkit.

1. Identifying necessary extension points and dependencies within Eclipse components;
2. Choosing extension points and dependencies with NeOn Toolkit; components;
3. Probing third-party APIs for potential conflicts with existing components;
4. Integrating the components into NeOn Toolkit using APIs and extension points provided by NeOn Toolkit;
5. Completing the integration process by creating appropriate GUI components and options for configuration;
6. Debugging and package releasing.

For Step 1 to 3, we introduce how we identify the extension points and dependencies in Chapter 3. For Step 4 to 6, each partner is responsible for completing these tasks in an individual basis and the coordinator is responsible for testing and releasing the integrated software.

1.3 Overview of the Deliverable

The overview of the deliverable is outlined as following:

- Chapter 1 of this deliverable gives an overall introduction to the motivation and challenge to realize the fishery ontologies lifecycle management system. Since delivering D7.4.1, the requirements about the fishery ontologies lifecycle have evolved been more concrete and have been explicitly outlined and refined in D7.4.2.
- Chapter 2 of this deliverable describes the major requirements of the fishery ontologies lifecycle management system based on the scope of the first prototype.
- Chapter 3 provides an architecture of this prototype about how the individual components are integrated into the prototype by aligning to the general NeOn architecture.
- Chapter 4 includes detailed description regarding individual components in this prototype within FAO's usage scenario. The general detailed description of each component to be integrated in the system are included in Annex I of D7.4.1.

- Chapter 5 presents some case studies about how the integrated components work together as a unified prototype for managing fishery ontology lifecycle.
- Chapter 6 describes the plan for subsequent T7.4 milestones and deliverables.
- Chapter 7 highlights the major conclusions of deliverable D7.4.1.

In this deliverable, we identify “software component” is individual piece of software that have not yet been integrated into NeOn Toolkit as a “NeOn Toolkit plug-in”.

Chapter 2

Requirements Analysis

In this chapter, we introduce the up-to-date requirements elaborated from FAO case study. The first prototype reported in this deliverable is designed to meet the these requirements by testing the data from FAO based on the use cases listed here as well.

2.1 Requirements for managing fishery ontology lifecycle

The ontologies produced within WP7 will be used by the Food and Agriculture Organization of the United Nations (FAO) in various medium-to-large web-based applications and in particular, they will underpin the Fisheries Stock Depletion Assessment System (FSDAS). To create and maintain distributed ontologies, it is necessary to put in place mechanisms that allow people involved form the fishery knowledge community, to deploy methodologies and technologies as they come from NeOn project . To support the continuous growth of the networked ontologies customized to the fisheries domain, these mechanisms must include:

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

For a successful implementation, and service delivery of the FSDAS, it is crucially required to support ontology editing and maintenance that can cope with the rate of data evolution, peculiar to the nature of the knowledge in a usage context like fisheries division; it is essential that ontology models can sustain high rate of amendments as they happen in the reality.

It is as well important to note that most of the ontologies in the fisheries domain will not be developed from scratch, but migrated from legacy systems. Therefore, the lifecycle management system requires adequate support to either migrate or map to ontologies data, from relational databases or XML schemas. For those ontologies that are already part of public domain, it is required support for coherency of standard and style of knowledge production.

WP7 case study requires articulated approach to design and implementation of ontologies, paying special attention to the editorial workflow, which will realize the ideal environment where several people with different skills, will contribute to the models maintenance. At the same time, having set the flow of amendments activity, will guarantee an appropriate support for the versioning system, to benefit public user's semantic web applications with reliable data support.

Ontology editors will be involved in the editorial workflow – mainly experts of fishery domain. Ontology editors will control the development of fragments of ontologies, revise the work of others, or develop multilingual versions of ontologies. Therefore, intuitive interfaces that hide the purely ontological and engineering options are required to be integrated with the toolkit.

Comprehensively the major objective of the WP7 case study is to have a robust, reliable and intuitive ontology management system to edit the networked ontologies that the applications will use as their basic data.

2.2 Use cases

Here it follows a short introduction to Use Cases descriptions as excerpted from D7.4.1, related to and only those which will form part of the first software prototype. Despite the initial planning of delivery of plug-ins (see D7.4.1), few have been delayed to system's second iteration, and other have been prioritized, because of their status of development; refer to deliverable introduction for extensive explanations. As for the other sections of chapter 2, references to more exhaustive source documents will be provided for interested readers. These use cases are going to be integrated and served as testing cases for the fishery ontology lifecycle management system.

UC-1.2: Search using advanced input (cf.3.2.1 in D7.4.1) Authorized users can search for any ontology element using semantic attributes or relationships held in the underlying model and exposed through template that allows to give preferred values to available fields ex: ?concepts having parent and/or that satisfy condition X (and/or Y); ?instances of classes that satisfy condition X;

UC-2.1: Answer standard query (cf.3.2.2 in D7.4.1) Authorized users can input queries supported by formal query languages available by the toolkit, and a graphical interface will hide the complexity of languages syntax (for example SPARQL).

UC-3.1: Add language (cf.3.2.3 in D7.4.1) Ontology engineers can add a new lexicalization for one single element of the ontology, up to the whole model, this task is performed supported by a conceptual artifact which reflects the exact relationships holding among elements of different lexicalisation,(see D7.1.2).

UC-3.2: Manage multilingual label (cf.3.2.3 in D7.4.1) This use case refers to the ontology localization activity; the system allows ontology editors to add, edit, or delete multilingual labels in individual concepts.

UC-3.3: Select working language (cf.3.2.3 in D7.4.1) Authorized users can select the language (intended as lexicalization) to visualize ontology information; ontology editors are supported in editorial work by the capability of displaying simultaneously multiple translations of the resources, when dealing with maintenance.

UC-4 Export (cf.2.4.8 in D7.1.2) All authorized users can export ontologies according defined conversion formats, in order to facilitate data exchange with legacy systems and uniformity with existing FAO resources. In particular, it should be possible to export ontologies converting their model (and included instances) reflecting existing DB schema (part of current FAO repositories), or as SKOS ontologies [SKOS].

UC-5.1: Convert ontology (cf.3.2.5 in D7.4.1) The ontology engineers are able to deploy the same NeOn metamodel, for a loaded ontology, into different ontology languages.

UC-5.2: Populate from database (cf.3.2.5 in D7.4.1) This use case supports ontology population with instances contained in DB model, by selecting the appropriate mapping document that reports correspondences between entities in the ontology, and in the DB model.

UC-5.4: Define resource converter (cf.3.2.5 in D7.4.1) This use case refers to the creation of the document formalizing correspondences between entities in a selected ontology, and a selected DB model. The same document will be used in population task from that DB model.

UC-6.1: Create mappings manually (cf.3.2.6 in D7.4.1) The users are able to create mappings between concepts and instances according to the set of relations actually supported by NeOn toolkit; the system creates and stores the document reporting the instantiated mappings.

UC-7.1: Visualize ontology for ontology engineers (cf.3.2.7 in D7.4.1) Ontology engineers can choose among different layouts of ontology visualization, with the option of simultaneously view a combination of those.

UC-7.2: Visualize ontology for ontology editors (cf.3.2.7 in D7.4.1) Ontology editors can visualize one or more ontology (modules) in multiple views, faceted by editorial workflow status.

UC-7.3: Visualize mappings between ontologies (cf.3.2.7 in D7.4.1) Authorized users are able to visualize several ontologies simultaneously, showing the mappings and relationship between the concepts and the modules.

UC-9.1: Capture ontology changes (cf.3.2.9 in D7.4.1) The editors are supported by a change tracking system, which will allow to store information as instantiations of the Change Ontology, as described in D1.3.1; a distributed peers of users involved in the maintenance activity, will connect to the model to retrieve last modification information. The same tracking system will provide automatic instantiation of information for those data related to time, object and author of the amendment, while will allow for manual, data annotations.

UC-9.2: View changes history (cf.3.2.9 in D7.4.1) Ontology editors are able to view the history of changes as reflected by the Change Ontology supporting the tracking system (see D1.3.1), together with any other kind of manual annotation by performing editors.

UC-9.3: View use statistics (cf.3.2.9 in D7.4.1) Ontology editors can view relevant information about an ontology, useful to fast analysis or retrieval: system of provenance, working editors, frequency of changes per fragment or module.

UC-9.4: View ontology statistics (cf.3.2.9 in D7.4.1) Authorized users can view statistics of the ontology being edited relatively to structural properties: depth of the class hierarchy, number of child nodes, number of relationships and properties, number of concepts per branch.

UC-10: Populate from text (cf.3.2.10 in D7.4.1) Ontology engineers can populate ontology with individuals suggested from text corpora processing, selected from available documents repository. The ontology engineer commit the instantiation for the appropriate candidates, having the possibility to analyse the excerpts of text source relatively to the extracted items.

UC-11.1: Check ontology (cf.3.2.11 in D7.4.1) This use case refers to the ontology diagnosis and ontology repair activities; Ontology editors and ontology engineers can check for duplicates elements within the selected ontologies, due to the distributed nature of the maintenance activity.

UC-11.2: Compare ontologies (cf.3.2.11 in D7.4.1) Ontology editors and ontology engineers can perform comparison between ontologies or their elements, supported by the visualization capabilities as described in UC-9.3 and UC-9.4.

UC-11.4: Evaluate Structural Properties To evaluate and rate an particular ontology to facilitate the ontology reuse is important to reusing existing ontology resource over internet.

UC-13: Obtain Documentation Authorized users can generate the documentation about the ontologies and modules they can access. The documentation generated should follow a similar style to those of JavaDoc or OWLdoc. This use case is related to the following activities in the NeOn methodology:

- Ontology documentation
- Ontology summarization

2.3 Data sets

Within T7.2, fisheries data resources have been investigated with the purpose of designing ontology models capable of reflecting the same structure, and contain their data as well. Data inventory and transformation methodology are fully described respectively in D7.2.1 and D7.2.2. Here it follows a short description of the outcome of that work, referring interested readers to the appropriate source. This first set of ontologies is the cornerstone to build the network that would allow extracting, analyze, aggregate data and information needed for the FSDAS. All of these ontologies have been populated from the Fisheries RTMS databases. These data sets can be found at: `http://www.fao.org/aims/neon.jsp`

2.3.1 Ontology Resources

Land areas This ontology organizes land areas at national level or group (geographic or economic) level. This information is important since most fisheries statistics are reported by individual or groups of countries.

- Land areas ontology model
- Land areas ontology populated with instance

Fishing areas This ontology organizes the FAO division areas for marine and inland waters, which are useful for statistical data collection and reporting. The division of water areas forms a strict and complete hierarchy.

- Fishing areas ontology model
- Fishing areas ontology populated with instances

Biological entities This ontology manages reference data about biological species needed for fisheries fact sheets and statistical information, among other resources. Species items are organized and maintained in the Aquatic Science and Fisheries Information System (ASFIS) and currently includes nearly 11.000 species items related to Fisheries and Aquaculture.

- Biological entities ontology model
- Biological entities ontology populated with instances

Fisheries commodities Fisheries commodities cover products derived from any aquatic animal (fish, crustaceans, molluscs) as well as residues for commercial, industrial or subsistence uses; fished in inland, fresh and brackish waters, in inshore, offshore or high seas fishing areas. Various classification systems are available for fisheries commodities: FAO's International Standard Statistical Classification of Fishery Commodities (ISSCFC) which is an expansion of the United Nations Standard International Trade Classification (SITC); and the Harmonized Commodity Description and Coding System maintained by theWorld Customs Organization (WCO). The fisheries commodities ontology manages all information necessary to use these classifications together.

- Fisheries commodities ontology model
- Fisheries commodities ontology populated with instances

Vessel types and size This ontology organizes the information necessary to assess fleet capacity and vessel main characteristics, such as its size or lenght. The ontology includes information form classifications used for vessel size, the Gross Register Tonnage (GRT), as defined by the Oslo Convention (1947); and the Gross Tonnage (GT) as defined by the 1969 London Convention.

- Vessels ontology model
- Vessels ontology populated with instances

Gear types This ontology manages reference data about gear types needed for the fisheries fact sheets. The type of gear installed on a vessel determines the type of fish that it can catch and therefore it is often used to determine the fleet power. The main classification of gear types is the International Standard Statistical Classification of Fishing Gear (ISSCFG). Although this classification was initially designed to improve the compilation of harmonised catch and effort data and in fish stock assessment exercises, it has also been found to be very useful for fisheries technology, fishermen training and for the preparation of specialized catalogues on artisan and industrial fishing methods.

- Gears ontology model
- Gears ontology populated with instances

2.3.2 Non Ontology Resources

FIGIS flat-file fact sheets. One branch of fact sheets is stored in directories on a server. A Lucene index is created from these fact sheets to render them queryable. Either they will be indexed by the RDF indexing component or OntoMap can be connected to the Lucene index.

RFB's (Regional Fisheries Bodies) document repositories The document repositories of the approximately fifty regional fishery bodies constitute an important source of information for resource assessments. Agreement can likely be reached to index at least some of these using the RDF indexing component.

Chapter 3

Architecture for Fishery Ontology Management

In this chapter, we depict the overall architecture of fishery ontology management system at the stage of first prototype. We first clarify the mappings between uses cases raised by FAO and individual components developed by different partners within NeOn project. Second, we figure out the relationships between data sets used by FAO for their daily work and the potential components that are going to process these data sets. Last, we design the architecture for the first prototype by giving an overview, finding out possible conflicts by analyzing interactions among different components, and covering the interactions between data sets and components.

3.1 Mappings between Use Cases and Components

Let's first look at the relationships between use cases and components in Table 3.1 below.

From the Table 3.1, we may find several remarks that should be taken into consideration:

- One use case can have different perspectives in usage. For example, user can populate ontologies by using Text2Onto, and they can also use SAFE to annotate the existing text-based documents. Both Text2Onto and SAFE are powered by GATE[1] by means of text mining techniques, whereas they have totally different scopes for actual application.
- One plug-in can be used for several uses cases. For example, Oyster is shared by all the UC-9 sub-use cases to support fishery ontology editing workflow.
- Several use cases that not such critical, such as UC-1.1, UC-5.3 [MGKI$^+$07], are missing in this table because the development of components are still ongoing. However, these missing use cases will be available in next prototypes.

We can see there are many individual components that are developed by different partners within in NeOn project separately, so it is very important to probe the compatibilities among components. We refer readers to Section 3.4 to check possible dependencies between components to see if there is any conflicts.

3.2 Mappings between Data Sets and Components

Along with the mappings between use cases and components, we also elaborate the mappings between data sets for FAO case study and components developed by different partners to clarify the possible cross-interactions between different components in the workflow of managing fishery ontologies. We will discuss several integrated use cases in Chapter 5 to show how these interactions occur in real life usage in FAO.

[1]http://gate.ac.uk

Use cases ID	Components	Use cases name
UC-1.2	Core NTK - Search	Search using advanced input
UC-2.1	Core NTK - Query Answering	Answer standard query
UC-3.1	Core NTK - GUI	Add language
UC-3.2	LabelTranslator	Manage multilingual label
UC-3.3	LabelTranslator	Select working language
UC-4	Core NTK - IO	Export ontology
UC-5.1	Core NTK - IO	Convert ontology
UC-5.2	ODEMapster	Populate ontology from database
UC-5.4	ODEMapster	Define resource converter
UC-6.1	OntoMap	Manage manual ontology mapping
UC-7.1	Core NTK - OntoVisualize	Visualize ontology for ontology engineers
UC-7.2	Core NTK - IO	Visualize ontology for ontology editors
	OWLDoc	Visualize ontology for ontology editors
UC-7.3	OntoMap	Visualize ontology mappings
UC-9.1	Oyster	Capture ontology changes
UC-9.2	Oyster	View changes history
UC-9.3	Oyster	View use statistics
UC-9.4	Oyster	View ontology statistics
UC-10	SAFE	Populate ontology from text
	Text2Onto	Populate ontology from text
UC-11.1	RaDON	Check ontologies
UC-11.2	Oyster	Compare ontologies
UC-11.4	Watson	Evaluate Structural Properties
UC-13	OWLDoc	Ontology documentation

Table 3.1: Mappings between use cases and components. NTK stands for NeOn Toolkit

Components	Input data	Output Data
Core NTK - Search	Ontologies and search text	Search result
Core NTK - OntoVisualize	Ontologies	Visualized ontology
Core NTK - Query Answering	Query and Ontologies	Answers
Core NTK - GUI	FAO user input	GUI events
Core NTK - IO	Ontologies	OWL/F-logic ontology
OntoMap	Ontologies	Mappings
LabelTranslator	Label in FAO official languages	Label in FAO official languages
ODEMapster	Relational databases	Ontology
Oyster	Ontologies	Ontology metadata
OWLDoc	Ontologies	Ontology documents
RaDON	Ontologies with conflicts	Ontologies without conflicts
SAFE	Document and text (e.g. FIGIS)	Ontology annotations
Text2Onto	Documents and text (e.g. FIGIS)	Ontology
Watson	Ontologies	Ontology management resource

Table 3.2: Mappings between components and FAO data sets. NTK stands for NeOn Toolkit. Here we talk about ontologies as OWL or F-logic ontology.

We also have several remarks regarding the relationships between data sets and components outlined in Table 3.2:

- The detailed information about input/output data are described in Chapter 2 briefly.
- There are dataflow between the input/output data for different components. For example, the output ontology from Text2Onto could be the input ontology for RaDON. This requires a more detailed analysis. We provide this analysis in Section 3.4
- For actual use cases that require more than one plugin/ functionalities, we refer readers to Chapter 5.

3.3 Ontology Language Support

As in NeOn project, we apply "dual-language" support for ontologies, it is necessary to figure it out for the users if the individual component supports both OWL and F-logic ontologies or just one of them. In Table 3.3 the status of language support for each component is listed.

Components	OWL Ontology Support	F-logic Ontology Support
Core NTK	Yes	Yes
LabelTranslator	Yes	Yes
ODEMapster	Yes	No
OntoMap	No	Yes
Oyster	Yes	No
OWLDoc	Yes	No
RaDON	Yes	No
SAFE	Yes	No
Text2Onto	Yes	No
Watson	Yes	No

Table 3.3: Ontology language support for individual components.

3.4 Architecture for the First Prototype

By outlining the potential interactions between components and use cases, as well as relationships between data sets and components, we are able to design the system in practical. In this section, we first provide a general overview of the layered architecture of this first prototype for managing fishery ontology lifecycle, and then we depict interactions between components and data sets in a big picture.

3.4.1 General overview

The general architecture for the first prototype of fishery ontology lifecycle management system are designed based on the NeOn architecture introduced in NeOn deliverable D6.1.1 [GAGW05], we group the components into three layers: (1) GUI components layer, (2) engineering components layer and (3) infrastructure services layer.

We may find that not all components fall into the GUI components layer, as some of the components doesn't not pose complicate GUI operations, such as SAFE and Oyster web services. Some plug-ins have both GUI and Engineering components, such as Text2Onto, because they contain rich GUI actions and background functions. Some components' GUI are quite simple and do not contain many actions, such as SAFE web service, so we don't classify them to have GUI components explicitly. The differences between tightly and

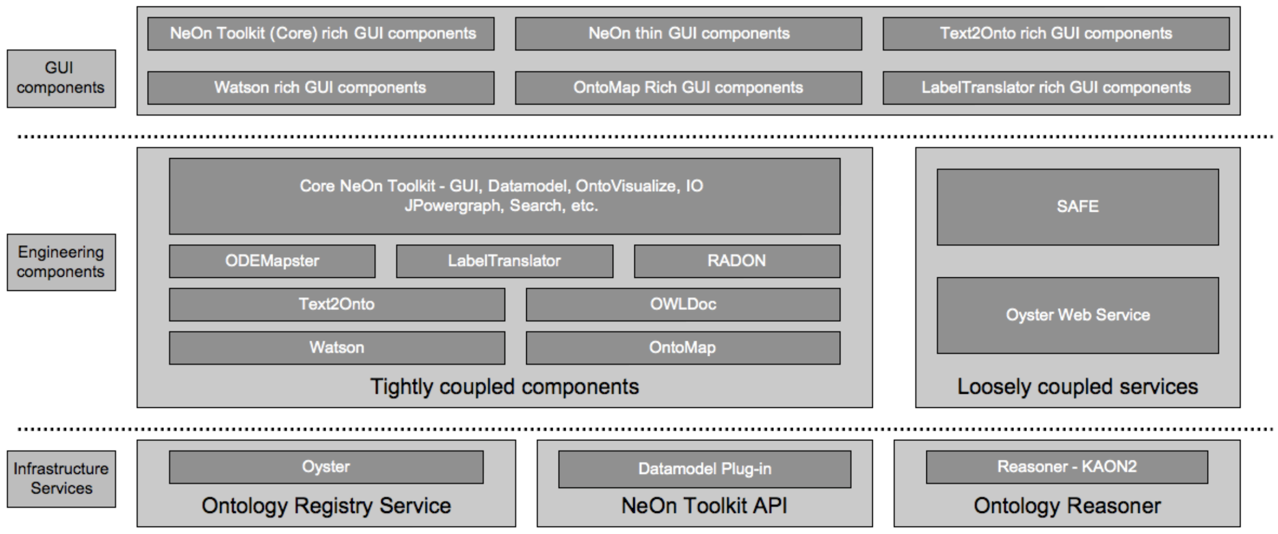

GUI components
NeOn Toolkit (Core) rich GUI components
NeOn thin GUI components
Text2Onto rich GUI components
Watson rich GUI components
OntoMap Rich GUI components
LabelTranslator rich GUI components
Engineering components
Core NeOn Toolkit - GUI, Datamodel, OntoVisualize, IO JPowergraph, Search, etc.
ODEMapster
LabelTranslator
RADON
Text2Onto
OWLDoc
Watson
OntoMap
Tightly coupled components
SAFE
Oyster Web Service
Loosely coupled services
Infrastructure Services
Oyster
Ontology Registry Service
Datamodel Plug-in
NeOn Toolkit API
Reasoner - KAON2
Ontology Reasoner

Figure 3.1: The general architecture for the first prototype of fishery ontology lifecycle management system.

loosely coupled are mainly contributed by the attributes of how the components are deployed. In infrastructure layer, the Oyster plays the role as ontology registry for sharing ontologies in P2P networking with support from KAON2 reasoning server. However, the API for ontology processing is not contributed by KAON2, as Datamodel plug-in in Core NeOn Toolkit provides this functionality.

3.4.2 Interactions between components and data sets

We now probe the dependencies of the major components appeared in this deliverable, and interactions when they operate on different data. The big picture regarding these interactions are shown in Figure 3.2

To understand more on Figure 3.2, we give some explanations below:

- We group ontologies and mappings into four different status to show the interactions between components and data sets: (1) Ontology in use – the ontology has been loaded into application; (2) mapping – a mapping between two ontologies, we assume mapping is a special form of ontology here as it also contains a set of axioms; (3) ontology in different language – an ontologies encoded in different languages compare to the ontology in use; (4) ontology with conflicts – the ontology is not consistent or has other bugs. This grouping is tailored for the different purposes of the components and shown with different color-styles.

- The semantics of single-direction arrows in the figure typically mean “output” or “operate”. For example, Core NeOn Toolkit operates ontologies, RaDON uses ontologies to analyze their consistencies, etc. The double direction arrows that point back to the component mean the contents of the ontology also affects the data posed by components, or data input. For example, Oyster will update its registry once it completes analyzing the metadata of a certain ontology; SAFE has ontologies together with documents as input.

The testing use cases in an integrated scenario is going to be performed in Chapter 5. We are going to do several integrated case studies by using several components introduced here by applying the FAO data introduced in Chapter 2.

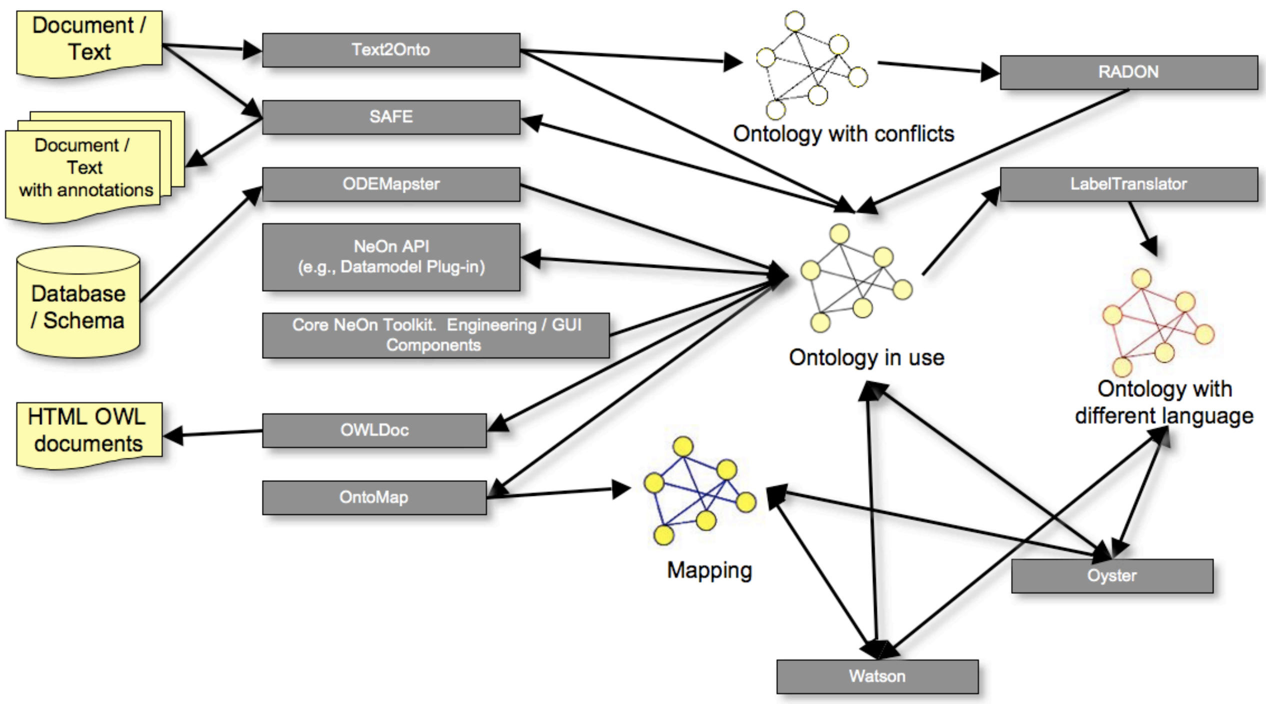


Figure 3.2: The interactions between components and data within first prototype of fishery ontology lifecycle management system. For the semantics of the arrows and detailed explanation, we refer readers to Chapter 5.

Chapter 4

Integration and Implementation

This chapter is a central part to introduce the major components that were developed and tailored for FAO case study: (1) Core NeOn Toolkit with (2) LabelTranslator, (3) ODEMapster, (4) OntoMap, (5) Oyster, (6) RaDON, (7) SAFE, (8) Text2Onto and (9) WATSON. These plug-ins are integrated with Core NeOn Toolkit to support the fishery ontology lifecycle management. We introduce the purpose and usage of these plug-in briefly, as well as their own targeting usage on FAO data, as the integrated usage examples will be shown in the next chapter.

4.1 Core NeOn Toolkit

The Core NeOn Toolkit addresses the following use cases:

- UC-1.2 Search using advanced input
- UC-2.1 Answer standard query
- UC-3.1 Add language
- UC-4 Export
- UC-5.1 Convert ontology
- UC-7.1 Visualize ontology for ontology engineers
- UC-7.2 Visualize ontology for ontology editors

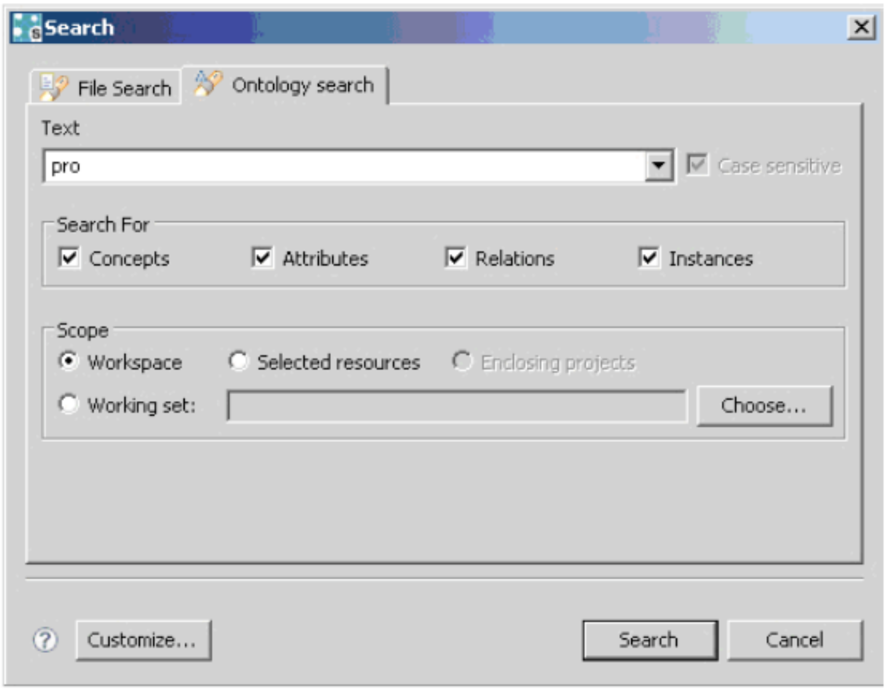
The Core NeOn Toolkit has been introduced in detail in NeOn deliverable D6.6.1 [EW07], there we are not going to details in this deliverable. Here we briefly recall the major plug-ins involved in Core NeOn Toolkit to have a general overview:

- `Core Neon Toolkit - Datamodel Plug-in` This plug-in represents the core plug-in of the NeOn Toolkit since it represents the internal datamodel and thus the knowledge representation aspect of the application. It provides access to the underlying datamodel via an API.
- `Core Neon Toolkit - GUI Plug-in` This plug-in contains the core UI components and connects them with the underlying datamodel via the DataModel plug-in. The main UI elements include the ontology navigator and the property editors for the different ontology entities.
- `Core Neon Toolkit - Import/Export Plug-in` Provides functionality to import and export ontologies in different formats from and to the local file system or a WebDAV server (Web- based Distributed Authoring and Versioning).

- `Core Neon Toolkit - OntoVisualize Plug-in` The ontovisualize plug-in contains a view to graphically visualize ontologies using the JPowerGraph library .
- `Core Neon Toolkit - JPowerGraph Plug-in` Integration of the jpowergraph library and some extensions for the visualization plug-in.
- `Core Neon Toolkit - Search Plug-in` This plug-in provides some search functionalities for ontological entities like concepts, attributes, relations and instances. Alternatively, to enhance the functionality of Core NeOn Toolkit, some plug-ins that are developed together with the above listed plug-ins, such as Query Answering Plug-in, OntoMap, are also provided.

The releases of NeOn Toolkit can be found at: `http://www.neon-toolkit.org/`. Below we briefly introduce how the use cases are addressed in NeOn Toolkit. Some functionalities are not fully complete yet, but they will be provided as soon as the OWL editor is complete.

UC-1.2 Search using advanced input With the "Search" function you can search for elements in ontologies. It opens the "Search" dialog (Figure 4.1). This dialog contains tabs for searching in files and for searching in ontologies. The file-search option is not functional since we operate on the conceptual ontology model, rather than on files storing the ontologies. The "Search" dialog You can search concepts, attributes, relations and instances inside the selected ontology.



Text: pro
[✓] Case sensitive

Search For: [✓] Concepts [✓] Attributes [✓] Relations [✓] Instances

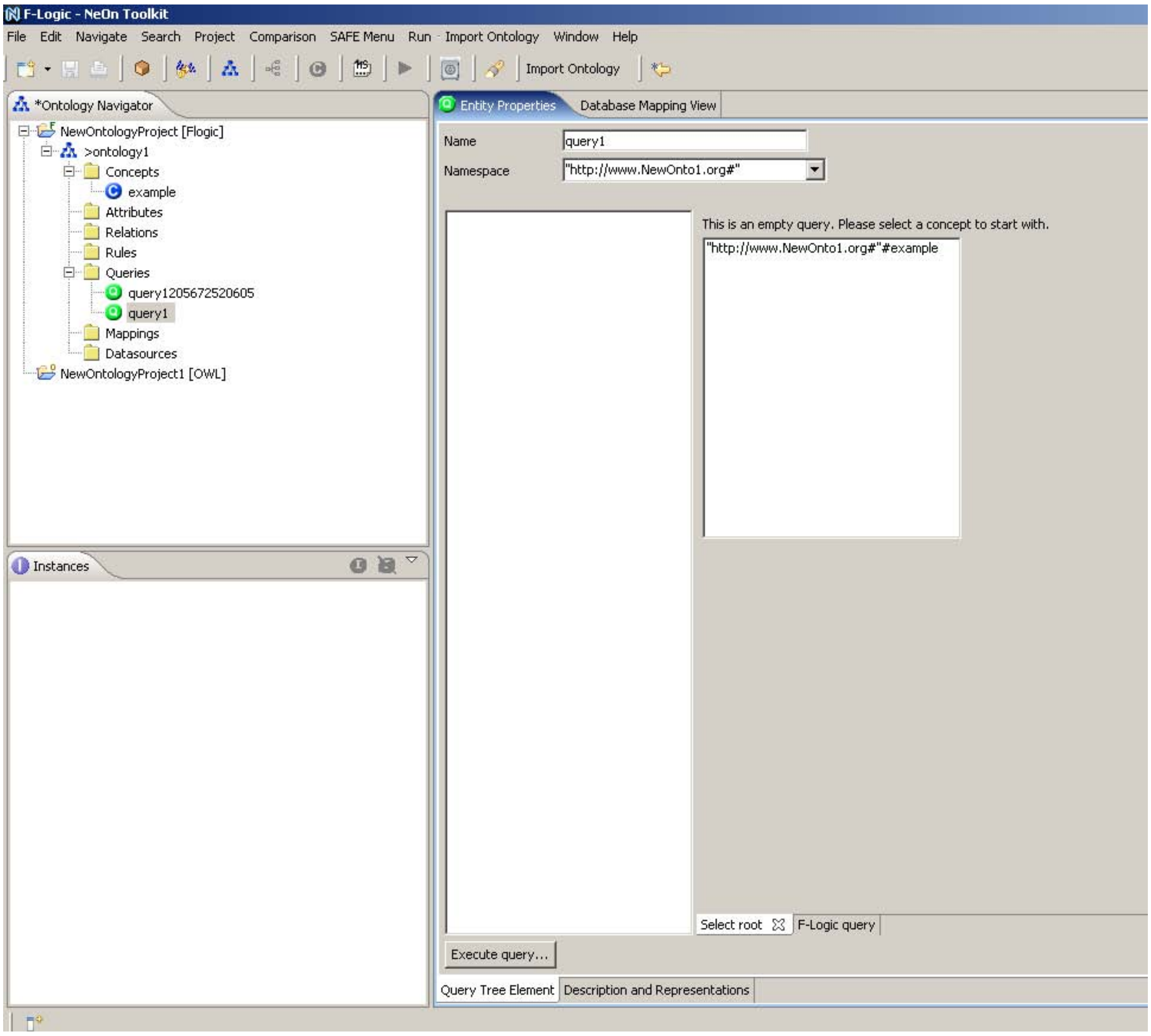
Scope: [✓ Workspace] Selected resources Enclosing projects
Working set:

Figure 4.1: The search dialogue.

UC-2.1 Answer standard query Answering standard query is currently working with F-logic ontologies. To issue a query is straightforward:

1. Creat a new F-logic ontology project or open an existing one;
2. Import an existing F-logic ontology or creat a new one;
3. Add a query entity to this ontology;

4. A view appears for users to issue query. (Figure 4.2)



Name: query1

Namespace: "http://www.NewOnto1.org#"

This is an empty query. Please select a concept to start with.

"http://www.NewOnto1.org#"#example

Figure 4.2: The query interface.

UC-3.1 Add language Within the language preferences you can specify the languages you want to develop your ontology for. To open the language preferences go to the Window menu, and select “Preferences > Language Preferences”. (Figure 4.3) The fields for documentation and representation then let you input values for these languages. All languages according to ISO standard are supported!

UC-4 Export We can export ontologies into the data formats F-Logic, OXML, OWL or RDF(s). If a project contains a number of ontologies (you can see this in the “Ontology Navigator” window), you must select the ontologies you wish to export by marking them specifically.

For exporting ontologies to the local file system choose the File System Export Wizard in the Export menu and confirm with Next. Dialog “Export File System” In the drop-down lists “Select project” and “Select ontology” choose the ontology to be exported. Click Browse to select a target file with a desired data type (“*.oxml”,

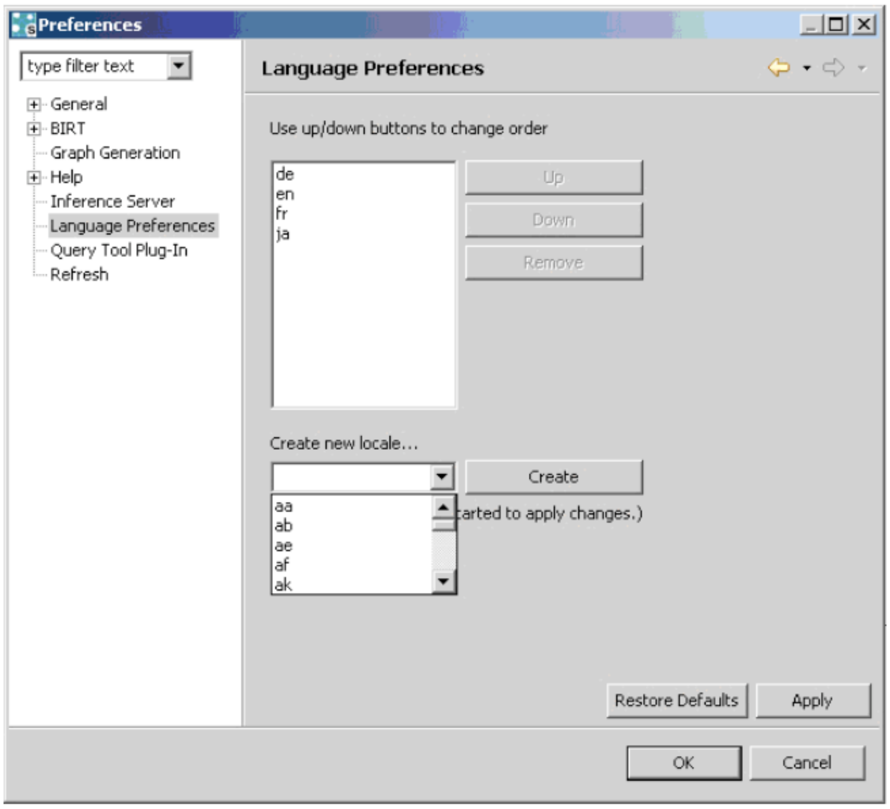


Figure 4.3: The Preferences menu to create or change the language.

"*.flo", "*.rdfs", "*.owl") and confirm with Finish. NeOn Toolkit will write the file to disk. Depending on the size of the ontology and the chosen format, the export process can take some time. (Figure 4.4)

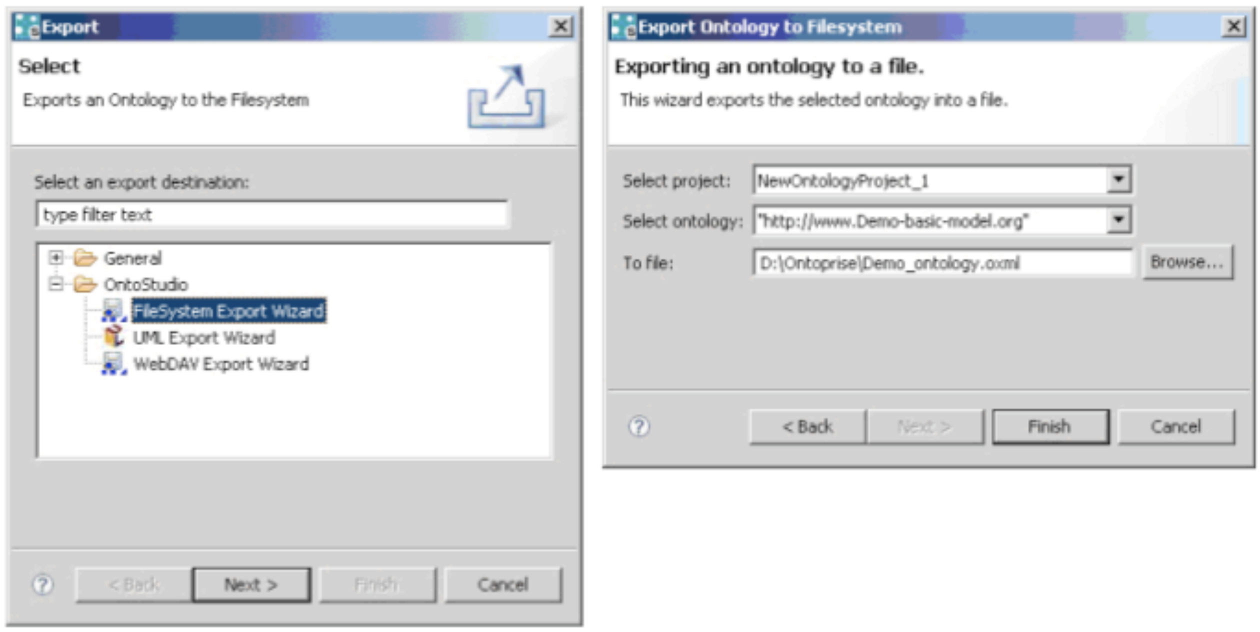
UC-5.1 Convert ontology We can simply import an ontology in one format and export this ontology into another format to support conversion of ontology formats. We have introduced how to export ontology above, we now talk about how to import an ontology: Currently we support:

- Loading ontology files in different formats from the file system or a WebDAV server
- Translating a database schema from a relational database management system into an ontology
- Interpreting the folder structures and files from the file system as instances (FileSystem MetaData Import Wizard)

For importing ontologies from the local file system, the following input formats are available:

- F-Logic – Frame logic is a knowledge representation- and ontology language (extension .flo)
- OXML – An XML-based ontology definition language used by ontoprise GmbH (extension .oxml)
- OWL – The web ontology language of the W3C (extension .owl)
- RDF / RDF (S) – Resource Description Framework (Schema) of the W3C (extension .rdf or .rdfs for the XML serialization or .nt for N-Triple or .n3 for N3)

To trigger this import, select "File > Import > OntoStudio > FileSystem Import Wizard" and click on "Next". The subsequent dialog lets you choose the location of the ontology (Browse button). You can also enter the location manually.(Figure 4.5)


Export
Select
Exports an Ontology to the Filesystem
Select an export destination:
type filter text
General
OntoStudio
FileSystem Export Wizard
UML Export Wizard
WebDAV Export Wizard
< Back
Next >
Finish
Cancel
Export Ontology to Filesystem
Exporting an ontology to a file.
This wizard exports the selected ontology into a file.
Browse...
< Back
Next >
Finish
Cancel

Select project: NewOntologyProject_1
Select ontology: "http://www.Demo-basic-model.org"
To file: D:\Ontoprise\Demo_ontology.oxml

Figure 4.4: Export an ontology.

UC-7.1 & 7.2 Visualize ontology The "Ontology Graph Visualize View" displays the ontology and all elements of the hierarchy (concepts, relations, attributes, instances) in a graph. The values of attributes and relations of instances are not visualized. In the Ontology Navigator you can click with right mouse button on the ontology you want to visualize and select "Visualize Ontology" or click with right mouse button on a single concept which you want to visualize and select "Show in Visualizer". (Figure 4.6 with result Figure 4.7)
Please note: This functionality is also partially covered by OWLDoc (see Sec. 4.5)

4.2 LabelTranslator Plug-in

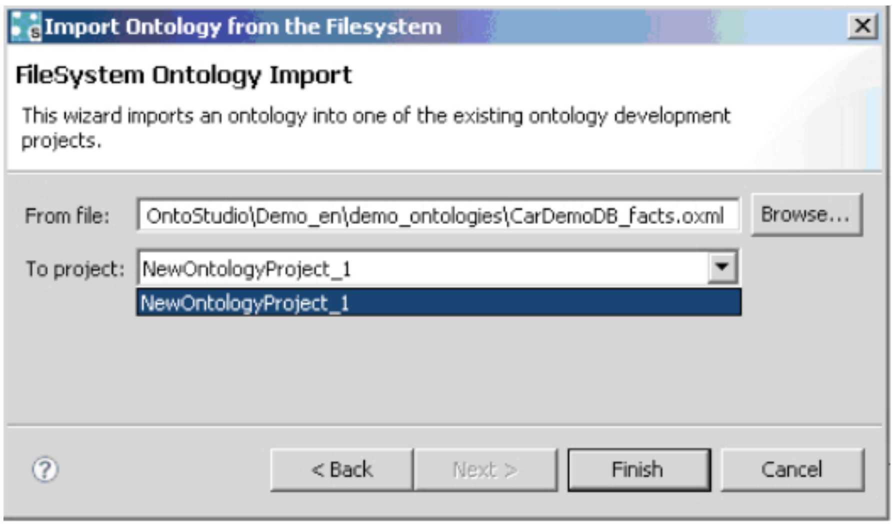
The LabelTranslator plug-in addresses the following use cases:

- UC-3.2 Manage multilingual label
- UC-3.3 Select working language

Motivation of using LabelTranslator The LabelTranslator plugin which extends the NeOn toolkit for supporting the translation of ontological labels using relevant information obtained from different lexical resources

Benefits/Advantages of using LabelTranslator Main charactecristics:

- LabelTranslator will give support to the translation of ontological labels. In this sense, a label can represent a class name, a property name, etc.
- Linguistic information to be considered (i.e. that LabelTranslator will manage) will be: Label, Gloss or Definition, Context or Additional Notes (i.e. explanations), Source of knowledge.
- LabelTranslator will look for the relevant information in the lexical resources that have been implemented, such as EWN databases.


Import Ontology from the Filesystem

FileSystem Ontology Import

This wizard imports an ontology into one of the existing ontology development projects.

From file: OntoStudio\Demo_en\demo_ontologies\CarDemoDB_facts.oxml

To project: NewOntologyProject_1

Figure 4.5: Convert an ontology by firstly importing an ontology.

Implementation and usage The high level architecture of this plug-in is depicted in Figure 4.8. 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.

4.3 ODEMapster Plug-in

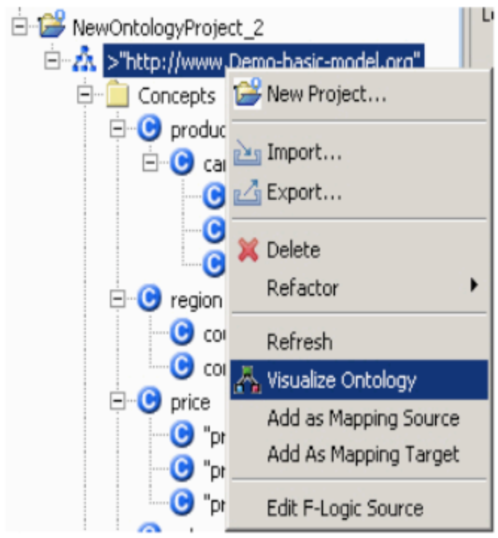
R2O, an extensible and declarative language to describe mappings between relational database (DB) schemas and ontologies, and ODEMapster which is the processor in charge of carrying out the exploitation of the mappings defined using R2O, performing both massive and query driven data upgrade. ODEMapster addresses the following use cases:

- UC-5.2 Populate from database
- UC-5.4 DeÞne resource converter

Motivation of using ODEMapster Within WP7, ontology engineers perform the ontology population, annotation or aggregation from unstructured information sources activities with various manual or (semi)automatic methods to transform unstructured, semi-structured and/or structured data sources into ontology instances. In the fisheries domain, this process consists mainly of converting semi-structured data sources (fishery fact sheets in XML format) and structured data source (from RTMS relational database) into corresponding instances in the conceptualized fisheries ontology.

Benefits/Advantages of using ODEMapster As it was mentioned before, ODEMapster is an engine that executes mappings between an ontology model and a database by means of a declarative language, R2O. ODEMapster offers two modes of execution:

- Query driven upgrade (on-the-fly query translation) each real object stays in the data sources themselves. At the run-time, the mapping converts the ontology-based query into a data source-based query to provide a variety of activities for retrieving instances.



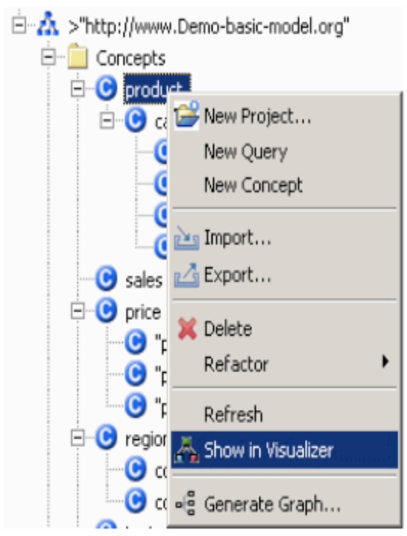


Figure 4.6: Context menu to initiate the ontology graph visualizer.

- Massive upgrade batch process that generates all possible Semantic Web individuals from the data repository. Each real object in the data sources is migrated into the ontologies as a formal instance.

FAO used the stand-alone version of ODEMapster to produce the ontologies delivered in D7.2.2 [Car07]. A number of considerations and lessons learned are also reported in that deliverable, together with the limitations found when dealing with databases such as the one used for fisheries data. The plan is to continue using ODEMapster, as a plug-in integrated in the NeOn toolkit, to create and populate other fisheries ontologies. Two approaches are currently enabled by ODEMapster:

- Population based on lifting/upgrading/migration.
- Query driven population

Implementation and usage By now, ODEMapster works with MySQL and ORACLE databases, and with OWL/RDF(S) ontologies. So far, FAO used ODEMapster to automatically populated the following ontologies (see Figure 4.9):

- Land areas
- Fishing areas
- Biological entities
- Fisheries commodities
- Vessel types and size
- Gear types

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

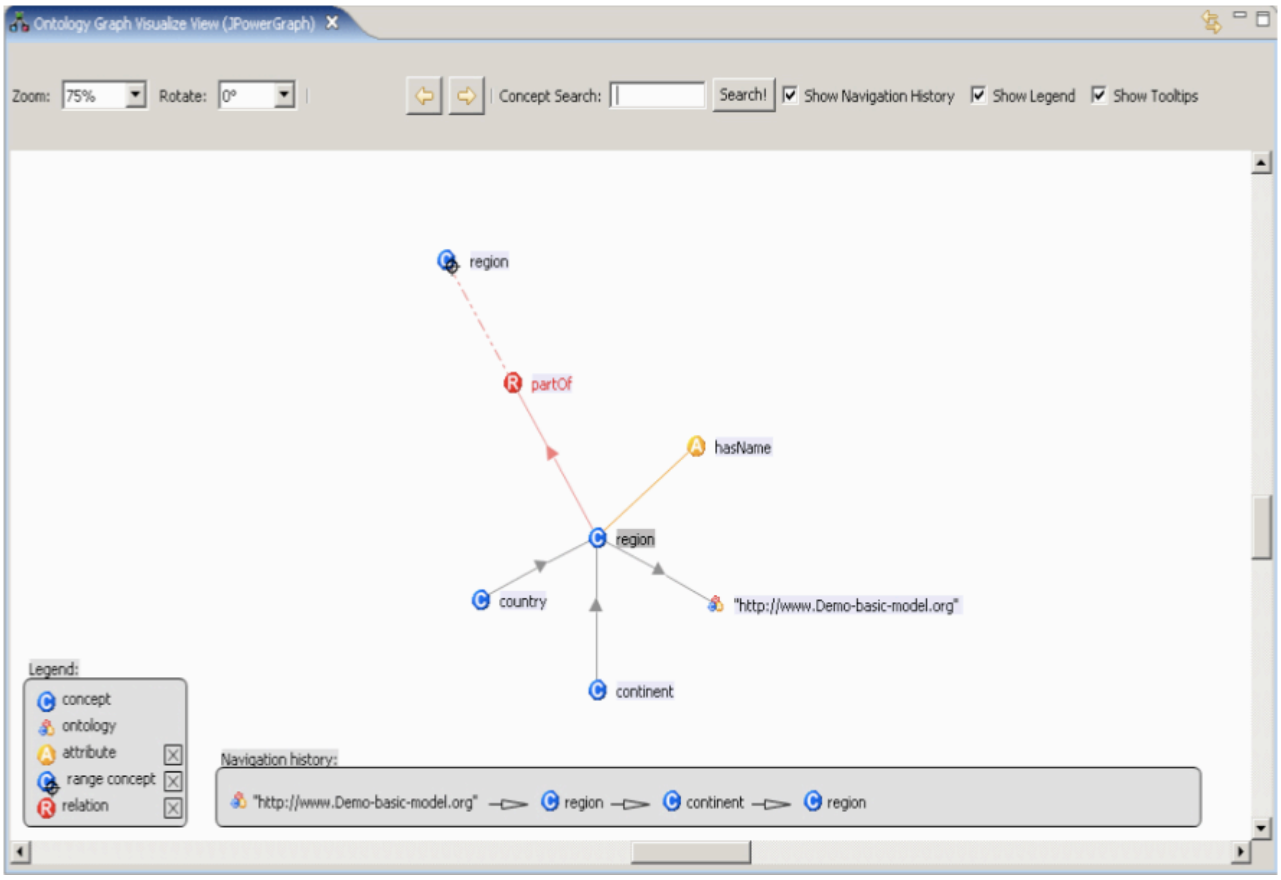


Figure 4.7: Result of using ontology visualizer.

4.4 OntoMap Plug-in

OntoMap is a graphical-based ontology mapping creation and editing tool that has been successfully integrated into NeOn Toolkit. We use OntoMap for daily maintenance of ontology mappings, by defining the relationship between ontology elements, including concepts, properties, etc. Currently, OntoMap supports creating mappings between F-logic ontologies. OntoMap addresses following use cases:

- UC-6.1 Create mappings manually
- UC-7.3 Visualize mappings between ontologies

Motivation of using OntoMap In NeOn project, ontology mapping is one of the major concerns in developing the networked ontology model. As we are dealing with dynamics of ontologies that are interconnected within a network, the ontology mapping is often represented as the interconnections. There are many approaches to create ontology mappings. Before going to an automatic/semi-automatic way, we first introduce OntoMap for manual creation of ontologies.

Implementation and usage of OntoMap OntoMap allows users creating mappings between ontology elements. The basic workflow are presented below:

1. Add Ontologies to the “Mapping View”.

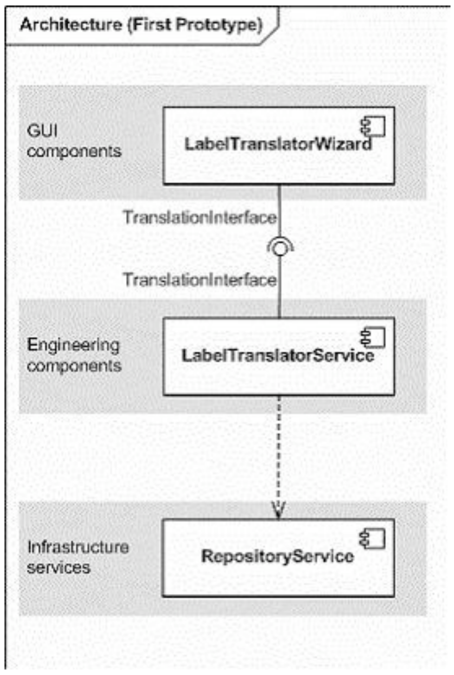


Figure 4.8: The general architecture for LabelTranslator plug-in.

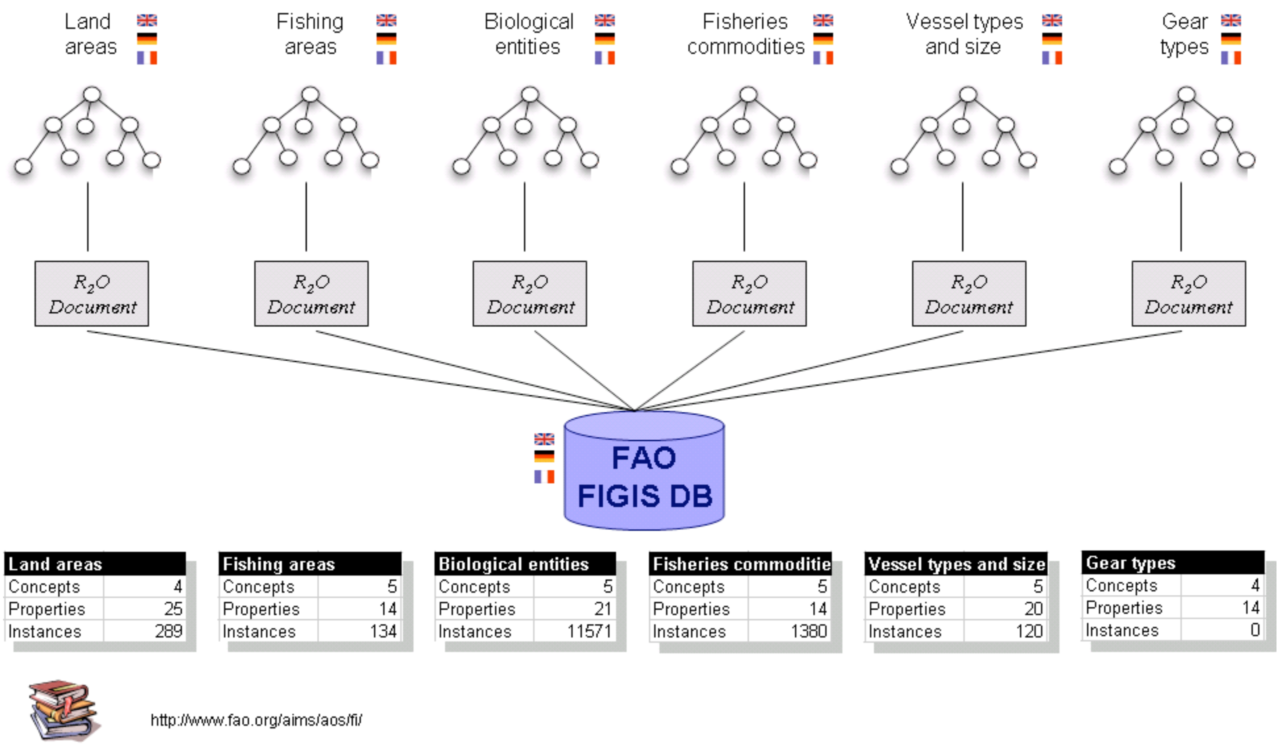
2. Defining new Dependencies
3. Deleting Dependencies
4. Transformation
5. Enhancement of the mapping folder
6. Concept to concept mapping (CC)
7. Attribute to concept mapping (AC)
8. Include Edit Mapping
9. Attribute to attribute mapping (AA)
10. Relation to relation mapping (RR)

The “Mapping View” (Figure 4.10) enables you to define mappings between different source (on the left side) and target- ontologies (on the right side). A released mapping between two elements is visualized by an arrow and can be activated by Drag & Drop from source to target elements.

We show an example of creating concept-to-concept mapping as a typical usage. By connecting a concept of the left side by “Drag & Drop” with a concept on the right side, a concept-concept mapping was produced.

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

1. Click on the arrow in the graphical diagram for which you want to define a filter.
2. Enter the desired values within the range filter. Property: The associated attributes and relations of the source element are offered to you for the selection. Operator: Different operators are offered to you for the selection. Value: Here you enter the value, with which the filter works.



Land areas	
Concepts	4
Properties	25
Instances	289

Fishing areas	
Concepts	5
Properties	14
Instances	134

Biological entities	
Concepts	5
Properties	21
Instances	11571

Fisheries commoditie	
Concepts	5
Properties	14
Instances	1380

Vessel types and size	
Concepts	5
Properties	20
Instances	120

Gear types	
Concepts	4
Properties	14
Instances	0

Figure 4.9: A usage example for ODEMapster plug-in in FAO data.

3. For a mapping several filters can be specified and linked by means of an “and” function. The filter was defined.

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

We are not going to show further details about creating other kinds of mappings, such as AA or RR mapping, as the workflow of creation is similar to create a CC mapping.

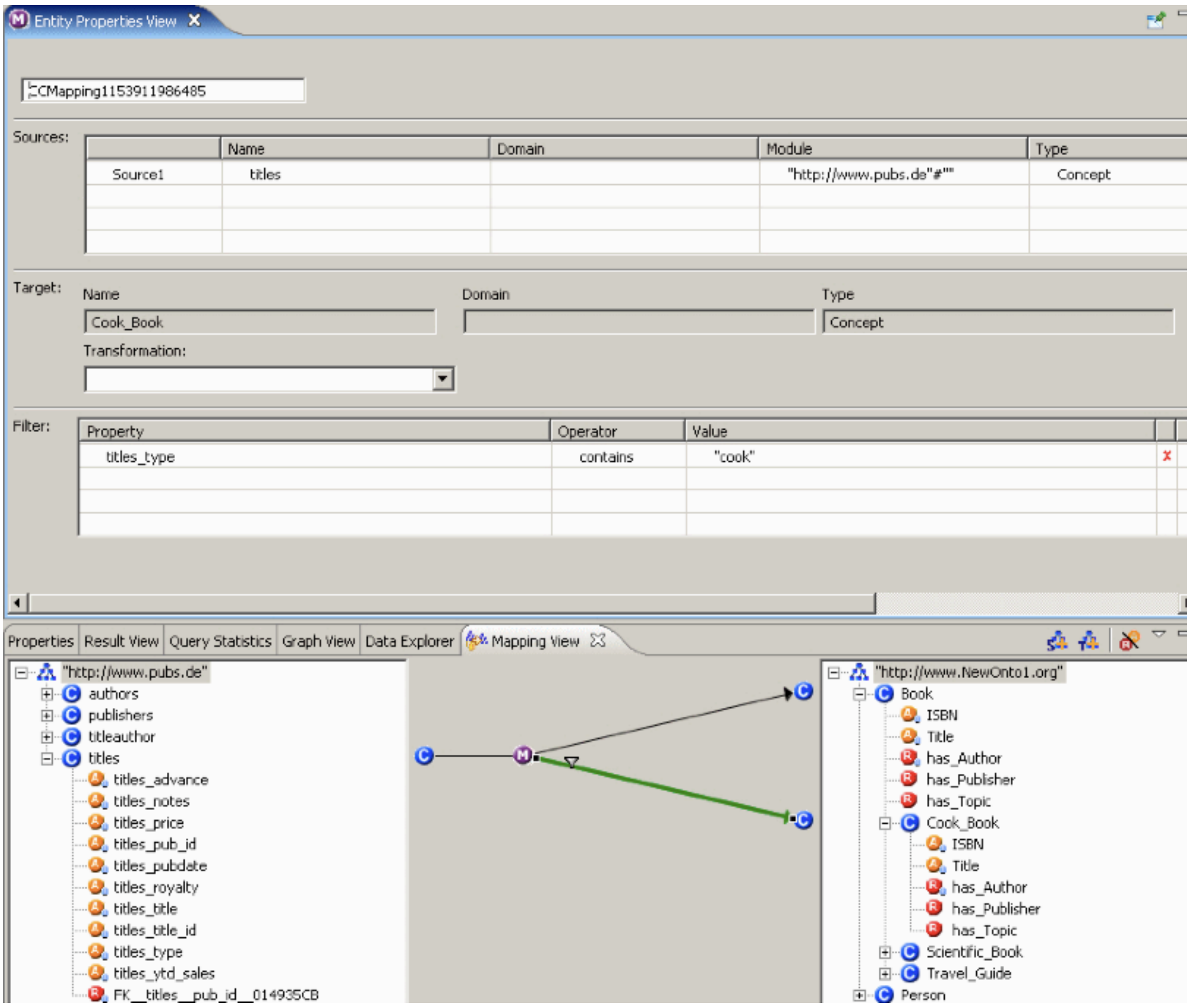
4.5 OWLDoc Plug-in

The OWLDoc Plug-in covers the following use cases:

- UC-7.2 Visualize ontology for ontology editors
- UC-13 Obtain Documentation

Motivation of using OWLDoc plugin The OWLDoc plugin adds to the NeOn Toolkit an option to export an OWL ontology as an HTML Documentation. This plugin uses the KAON2 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.

Benefits/Advantages of OWLDoc plugin 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.



CMapping1153911986485

Sources:

	Name	Domain	Module	Type
Source1	titles		"http://www.pubs.de"#""	Concept

Target:

Name	Domain	Type
Cook_Book		Concept

Transformation:

Filter:

Property	Operator	Value
titles_type	contains	"cook"

Figure 4.10: A usage example for OntoMap plug-in.

Implementation and usage 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 (Figure 4.11). 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 4.12)

4.6 Oyster Plug-in and Service

The Oyster aims to support following use cases:

- UC-9.1 Capture ontology changes
- UC-9.2 View changes history
- UC-9.3 View use statistics
- UC-9.4 View ontology statistics

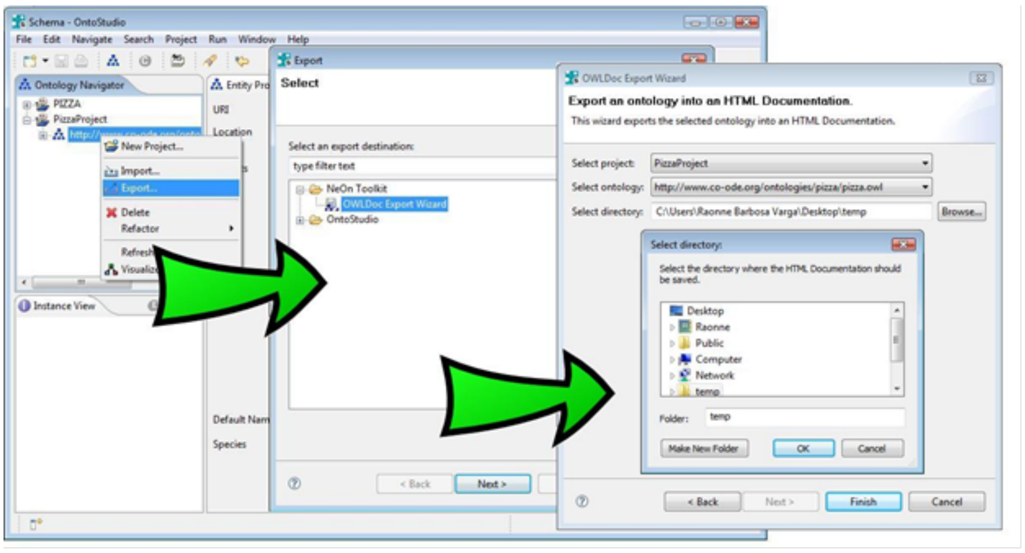

Export an ontology into an HTML Documentation.
This wizard exports the selected ontology into an HTML Documentation.
Select project: PizzaProject
Select ontology: http://www.co-ode.org/ontologies/pizza/pizza.owl
Select directory: C:\Users\Raonne Barbosa Varga\Desktop\temp
Select the directory where the HTML Documentation should be saved.
Folder: temp

Figure 4.11: A usage example for OWLDoc plug-in.

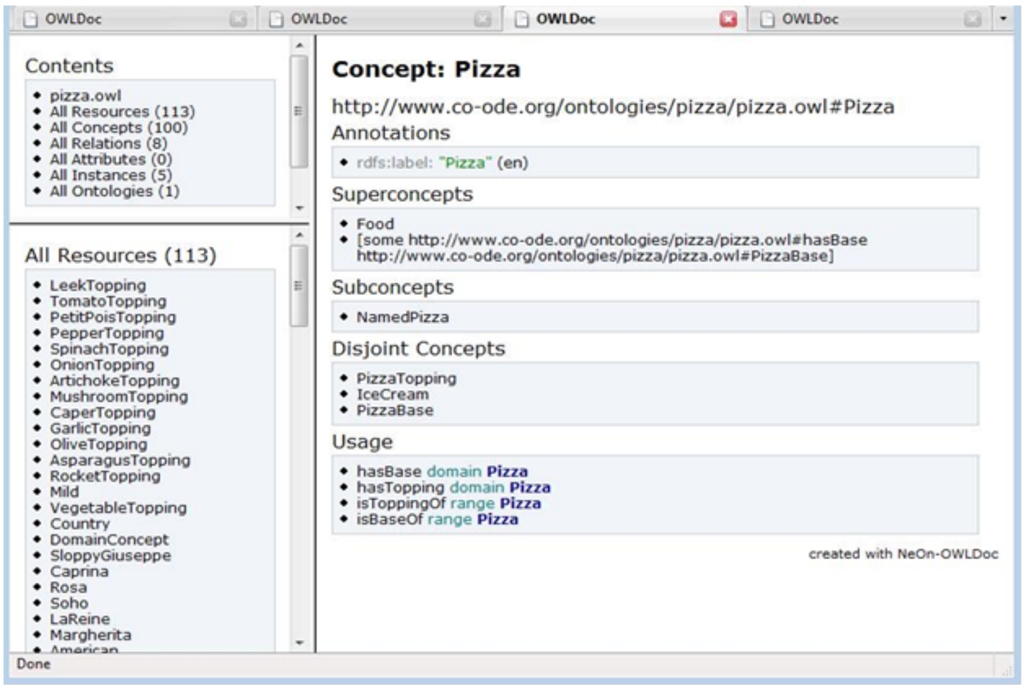
The Oyster system (Oyster system is freely available under `http://ontoware.org/projects/oyster2/`) was designed using a service-oriented approach, and it provides a well defined API. Accessing the registry functionalities can be done using directly the API within any application, invoking the web service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

Please note, Oyster supports the use cases list above as it provides basic functionality and metadata support for the use cases, while the actual GUI is going to be implemented soon as next step. We refer readers to D1.3.1 [PHWd07] for future information.

Motivation of using Oyster One of the goals of the FAO use case partner is that fisheries ontologies produced within WP7 will underpin the Fisheries Stock Depletion Assessment System (FSDAS). However, for such a dynamic domain like fisheries that is continuously evolving, we will need to provide the appropriate support for a successful implementation and service delivery of the FSDAS. In particular, for this task we need to support a collaborative editorial workflow that will allow Ontology editors to consult, validate and modify 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.

Benefits/Advantages of using Oyster

- Oyster implements the standard proposal for annotating ontologies OMV (Ontology Metadata Vocabulary).
- Ontology changes are formally represented as an ontology that is an extension to the OMV. Oyster provides the support to store and manage that information.



Contents

- pizza.owl
- All Resources (113)
- All Concepts (100)
- All Relations (8)
- All Attributes (0)
- All Instances (5)
- All Ontologies (1)

All Resources (113)

- LeekTopping
- TomatoTopping
- PetitPoisTopping
- PepperTopping
- SpinachTopping
- OnionTopping
- ArtichokeTopping
- MushroomTopping
- CaperTopping
- GarlicTopping
- OliveTopping
- AsparagusTopping
- RocketTopping
- Mild
- VegetableTopping
- Country
- DomainConcept
- SloppyGiuseppe
- Caprina
- Rosa
- Soho
- LaReine
- Margherita
- American

Concept: Pizza

http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza

Annotations

- rdfs:label: "Pizza" (en)

Superconcepts

- Food
- [some http://www.co-ode.org/ontologies/pizza/pizza.owl#hasBase http://www.co-ode.org/ontologies/pizza/pizza.owl#PizzaBase]

Subconcepts

- NamedPizza

Disjoint Concepts

- PizzaTopping
- IceCream
- PizzaBase

Usage

- hasBase domain **Pizza**
- hasTopping domain **Pizza**
- isToppingOf range **Pizza**
- isBaseOf range **Pizza**

created with NeOn-OWLDoc

Figure 4.12: Result of using OWLDoc plug-in.

- Oyster keeps the track and information about the different versions of ontologies and the relationship between them.
- Oyster provides a well defined API and a Web Service interface to support the integration with other components (e.g. the visualisation component).
- Oyster is in charge of the propagation of the ontology changes to their related metadata and ensures their consistency in a distributed environment.

Implementation and Usage Currently, Oyster can extract metadata from ontologies in OWL, DAML-OIL and RDFS. Users are annotating ontologies that will become available in the distributed environment. One special big provider will be the Watson Node. The major functions implemented in current Oyster are:

- 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.

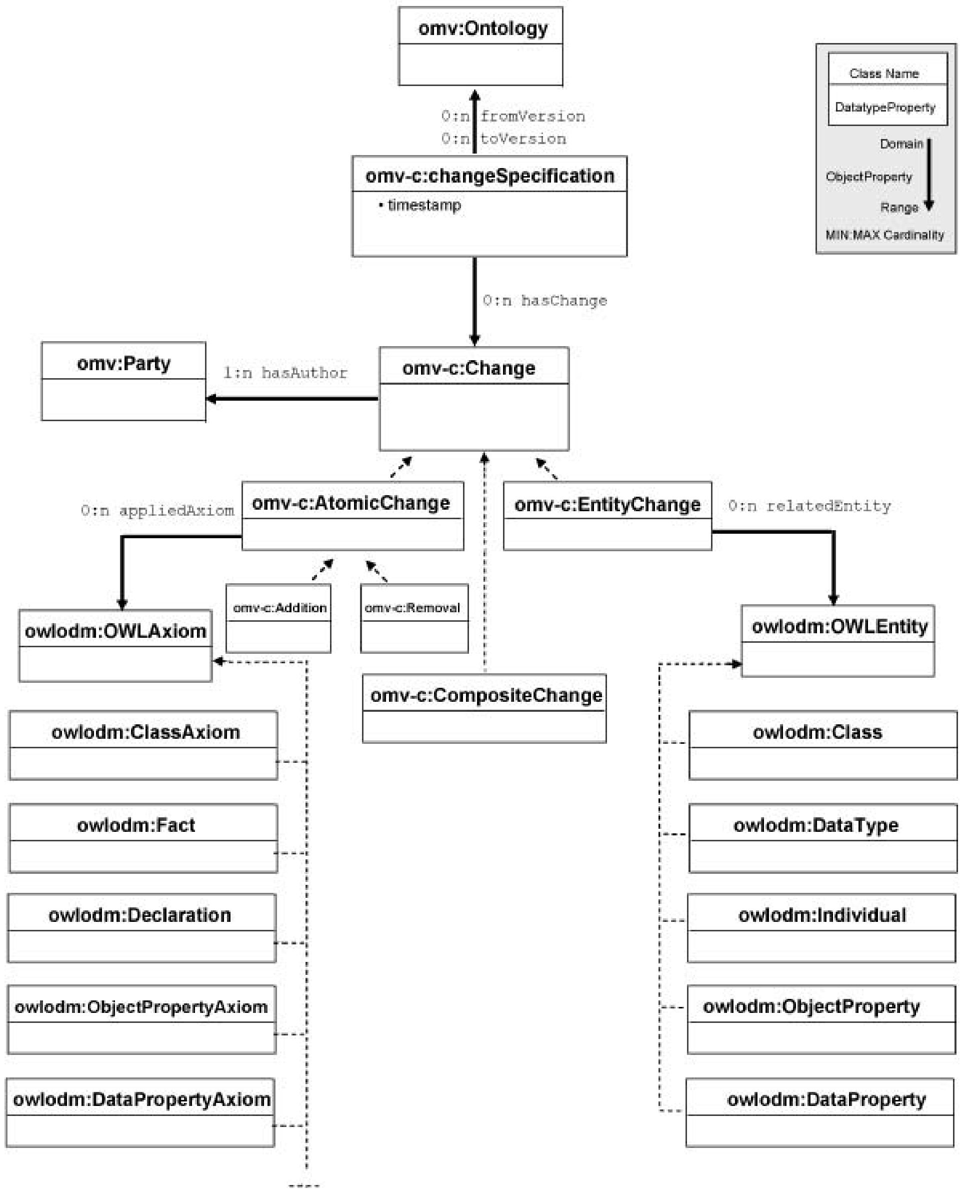


Figure 4.13: Overview of change ontology in Oyster.

- 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.

For the maintenance of changes (UC9.1) The system keeps tracks of changes in all ontology elements. That is, the date of creation/editing, the editor that made the change, etc. In Oyster all actions are logged according to the change ontology (see Figure 4.13), which captures among others the required following minimum fields for ontology changes:

- description
- operaton executed
- timestamp

- URI
- user

4.7 RaDON Plug-in

RaDON plugin is to deal with inconsistency and incoherence occurring in networked ontologies. Specifically, RaDON provides the following motivated functions. RaDON supports following use cases:

- UC-11.1 Check ontology

Motivation of using RaDON Ontology Debug:

- Incoherence : If the TBox of an ontology is incoherent, the minimal unsatisfiability-preserving subsets (MUPSs) or/and minimal incoherence-preserving subsets(MIPSs) will be returned to users.
- Inconsistency: If the ontology is inconsistent, all the minimal conflict subsets which are responsible for the inconsistency are provided.

Ontology repair:

- Repair a whole ontology automatically
- Repair an ontology with the users' decision. In such case, the MIPSs or conflict sets will be provided to users. And for each axiom in the MIPSs or conflict sets, a score will also be given to help user make a decision.

Benefits/Advantages of using RaDON RaDON plugin is a view in NeOn toolkits 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:

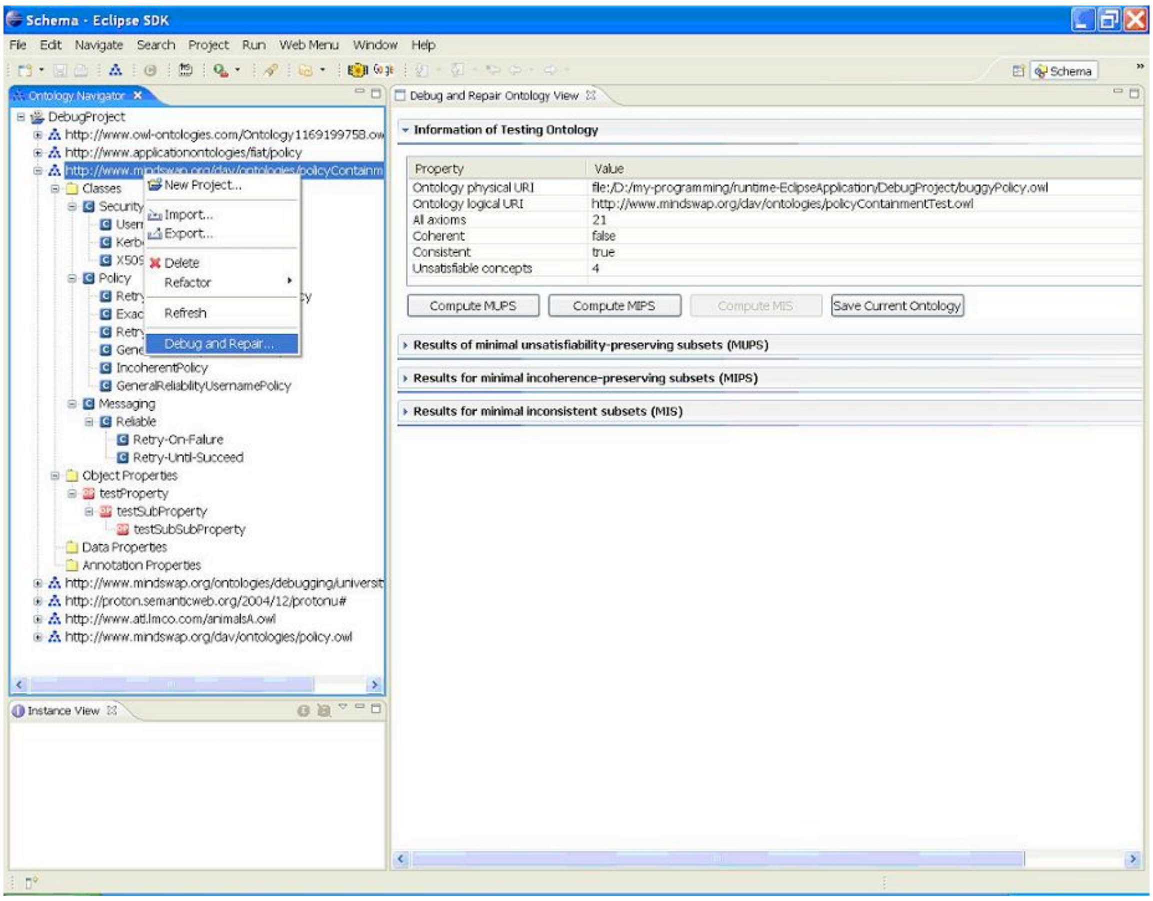
- Configure part: Where user can configure some parameters for debugging or repairing.
- Execution part: Here one can choose whether an approximate approach is used or not.
- Results part: All the results including the execution time except manual repair are shown here.

Implementation and usage The user interface for debugging and repairing is a view of “Debug and Repair Tool”. Figure 4.14 shows the current development status of the view which consists of the followings parts :

1. The user interface for debugging and repairing is a view of “Debug and Repair Ontology View”.
2. 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...”. 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 4.14)
3. To compute 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 4.15)

4. To repair automatically. According to the obtained MUPS / MIPS / MIS, user can repair the ontology automatcially 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 4.16)

5. To repair manually. 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 removed the selected axioms from the test ontology. (Figure 4.17)


Schema - Eclipse SDK
File Edit Navigate Search Project Run Web Menu Window Help
Ontology Navigator
Debug and Repair Ontology View
Information of Testing Ontology
New Project...
Import...
Export...
Delete
Refactor
Refresh
Debug and Repair...
Compute MUPS
Compute MIPS
Compute MIS
Save Current Ontology
Results of minimal unsatisfiability-preserving subsets (MUPS)
Results for minimal incoherence-preserving subsets (MIPS)
Results for minimal inconsistent subsets (MIS)
Instance View

Property	Value
Ontology physical URI	file:/D:/my-programming/runtime-EclipseApplication/DebugProject/buggyPolicy.owl
Ontology logical URI	http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl
All axioms	21
Coherent	false
Consistent	true
Unsatisfiable concepts	4

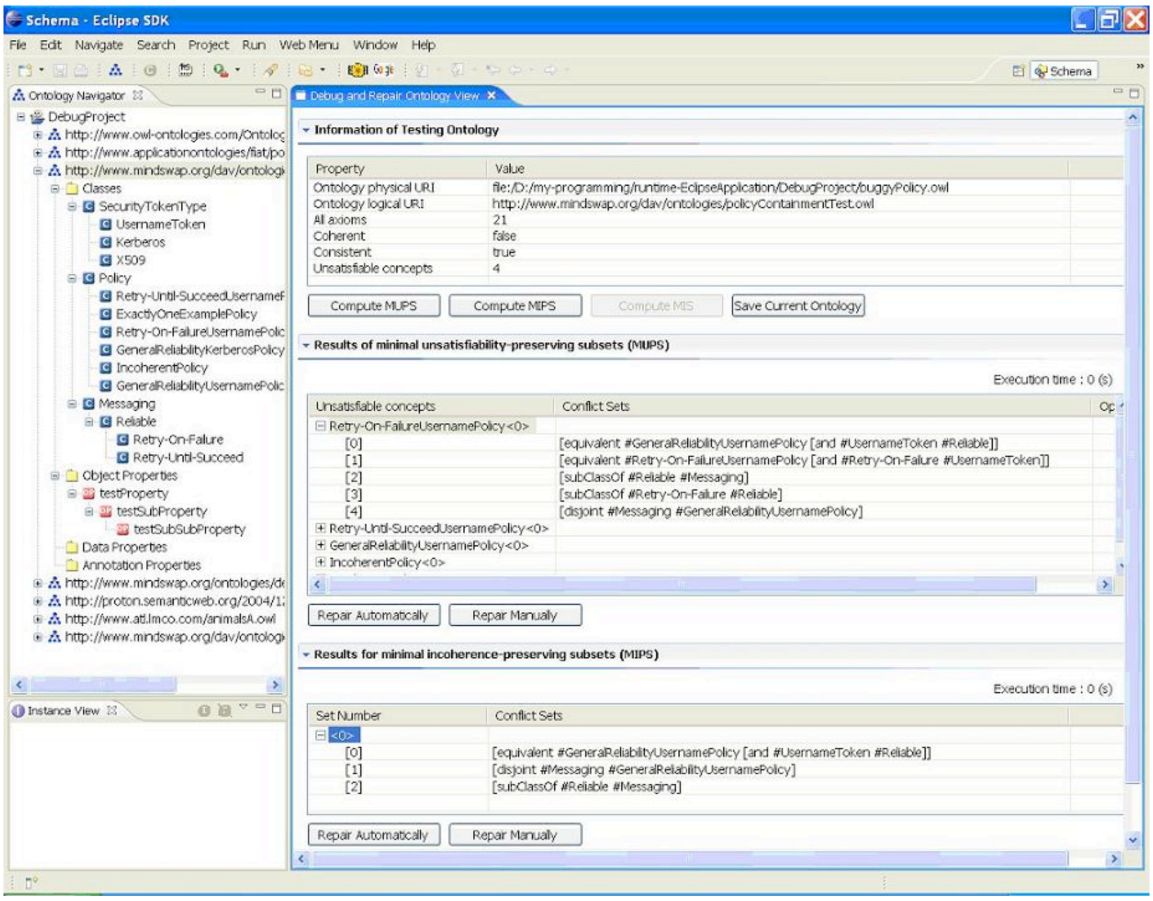
Figure 4.14: A usage example for RaDON plug-in - part 1.

4.8 SAFE Web Service

SAFE addresses following use cases to support ontology population from text:

- UC-10 Populate from text

Motivation of using SAFE SAFE has in the following features for practical usage:



```
DebugProject
  http://www.owl-ontologies.com/Ontolog
  http://www.applicationontologies/fiat/po
  http://www.mindswap.org/dav/ontologi
    Classes
      SecurityTokenType
        UsernameToken
        Kerberos
        X509
      Policy
        Retry-Until-SucceedUsernameF
        ExactlyOneExamplePolicy
        Retry-On-FailureUsernamePolic
        GeneralReliabilityKerberosPolicy
        IncoherentPolicy
        GeneralReliabilityUsernamePolic
      Messaging
        Reliable
          Retry-On-Failure
          Retry-Until-Succeed
    Object Properties
      testProperty
        testSubProperty
          testSubSubProperty
    Data Properties
    Annotation Properties
  http://www.mindswap.org/ontologies/de
  http://proton.semanticweb.org/2004/1:
  http://www.atl.lmco.com/animalsA.owl
  http://www.mindswap.org/dav/ontologi
```

Information of Testing Ontology

Property	Value
Ontology physical URI	file:/D:/my-programming/runtime-EclipseApplication/DebugProject/buggyPolicy.owl
Ontology logical URI	http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl
All axioms	21
Coherent	false
Consistent	true
Unsatisfiable concepts	4

Results of minimal unsatisfiability-preserving subsets (MUPS)

Execution time : 0 (s)

Unsatisfiable concepts	Conflict Sets	Op
Retry-On-FailureUsernamePolicy<0>		
[0]	[equivalent #GeneralReliabilityUsernamePolicy [and #UsernameToken #Reliable]]	
[1]	[equivalent #Retry-On-FailureUsernamePolicy [and #Retry-On-Failure #UsernameToken]]	
[2]	[subClassOf #Reliable #Messaging]	
[3]	[subClassOf #Retry-On-Failure #Reliable]	
[4]	[disjoint #Messaging #GeneralReliabilityUsernamePolicy]	
Retry-Until-SucceedUsernamePolicy<0>		
GeneralReliabilityUsernamePolicy<0>		
IncoherentPolicy<0>		

Results for minimal incoherence-preserving subsets (MIPS)

Execution time : 0 (s)

Set Number	Conflict Sets
<0>	
[0]	[equivalent #GeneralReliabilityUsernamePolicy [and #UsernameToken #Reliable]]
[1]	[disjoint #Messaging #GeneralReliabilityUsernamePolicy]
[2]	[subClassOf #Reliable #Messaging]

Figure 4.15: A usage example for RaDON plug-in - part 2.

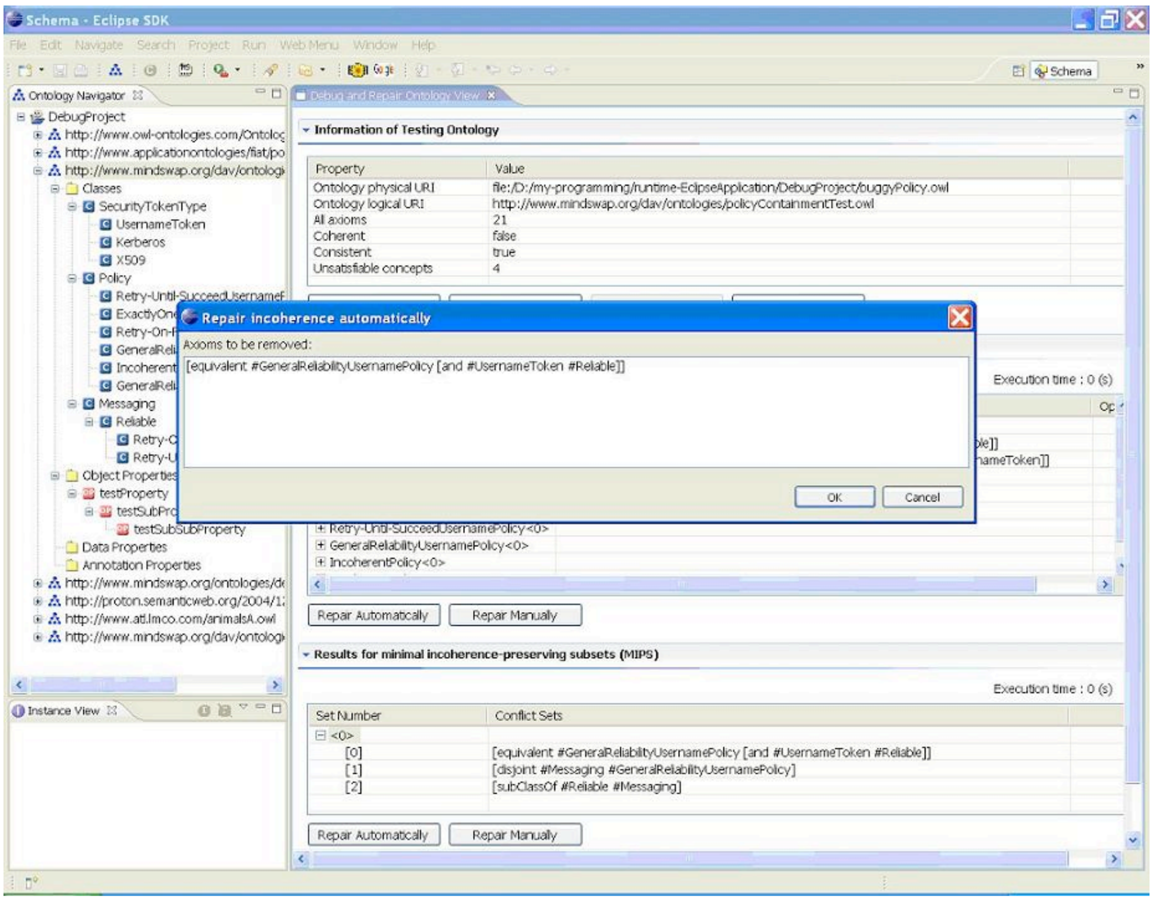
- SAFE is very easy-accesible as it doesn't require much effort in configuring the local third party application support because it is a web service.
- It is quite easy from users to access SAFE web services and it does cost local computing resource.

BeneÞts/Advantages of using SAFE 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.

Implementation and Usage 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

]]


Schema - Eclipse SDK
File Edit Navigate Search Project Run Web Menu Window Help
Ontology Navigator
Debug and Repair Ontology View
Schema
Instance View

Information of Testing Ontology

Property	Value
Ontology physical URI	file:/D:/my-programming/runtime-EclipseApplication/DebugProject/buggyPolicy.owl
Ontology logical URI	http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl
All axioms	21
Coherent	false
Consistent	true
Unsatisfiable concepts	4

Repair incoherence automatically

Axioms to be removed:

[equivalent #GeneralReliabilityUsernamePolicy [and #UsernameToken #Reliable]]

OK Cancel

Execution time : 0 (s)

Retry-Until-SucceedUsernamePolicy<0>
GeneralReliabilityUsernamePolicy<0>
IncoherentPolicy<0>

Repair Automatically Repair Manually

Results for minimal incoherence-preserving subsets (MIPS)

Execution time : 0 (s)

Set Number	Conflict Sets
<0>	
[0]	[equivalent #GeneralReliabilityUsernamePolicy [and #UsernameToken #Reliable]]
[1]	[disjoint #Messaging #GeneralReliabilityUsernamePolicy]
[2]	[subClassOf #Reliable #Messaging]

Repair Automatically Repair Manually

Figure 4.16: A usage example for RaDON plug-in - part 3.

- 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.

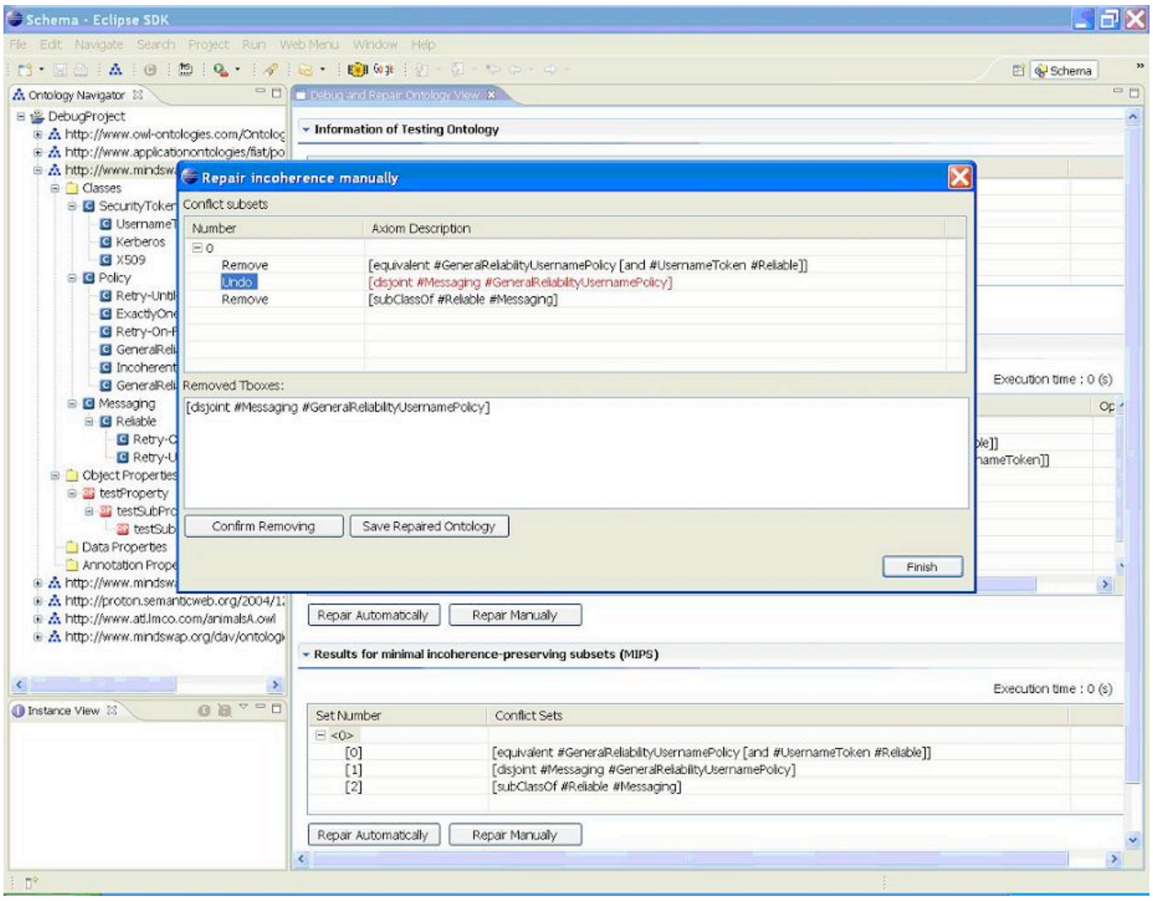
4.9 Text2Onto Plug-in

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. Text2Onto also addresses:

- UC-10: Populated ontology from text

This use case is also supported by SAFE, however, they have different scope in actual usage.

Motivation of using Text2Onto The operations of enriching, populating and mapping fisheries ontologies currently adopted at FAO are very tedious and cost intensive, since they require ontology editors with consolidated expertise in the domain of interest. During the population task, especially when a totally new ontology


Schema - Eclipse SDK
File Edit Navigate Search Project Run Web Menu Window Help
Ontology Navigator
Debug and Repair Ontology View
Information of Testing Ontology
Repair incoherence manually
Confirm Removing
Save Repaired Ontology
Finish
Repair Automatically
Repair Manually
Instance View

Conflict subsets

Number	Axiom Description
0	
Remove	[equivalent #GeneralReliabilityUsernamePolicy [and #UsernameToken #Reliable]]
Undo	[disjoint #Messaging #GeneralReliabilityUsernamePolicy]
Remove	[subClassOf #Reliable #Messaging]

Removed Tboxes:

[disjoint #Messaging #GeneralReliabilityUsernamePolicy]

Results for minimal incoherence-preserving subsets (MIPS)

Execution time : 0 (s)

Set Number	Conflict Sets
<0>	
[0]	[equivalent #GeneralReliabilityUsernamePolicy [and #UsernameToken #Reliable]]
[1]	[disjoint #Messaging #GeneralReliabilityUsernamePolicy]
[2]	[subClassOf #Reliable #Messaging]

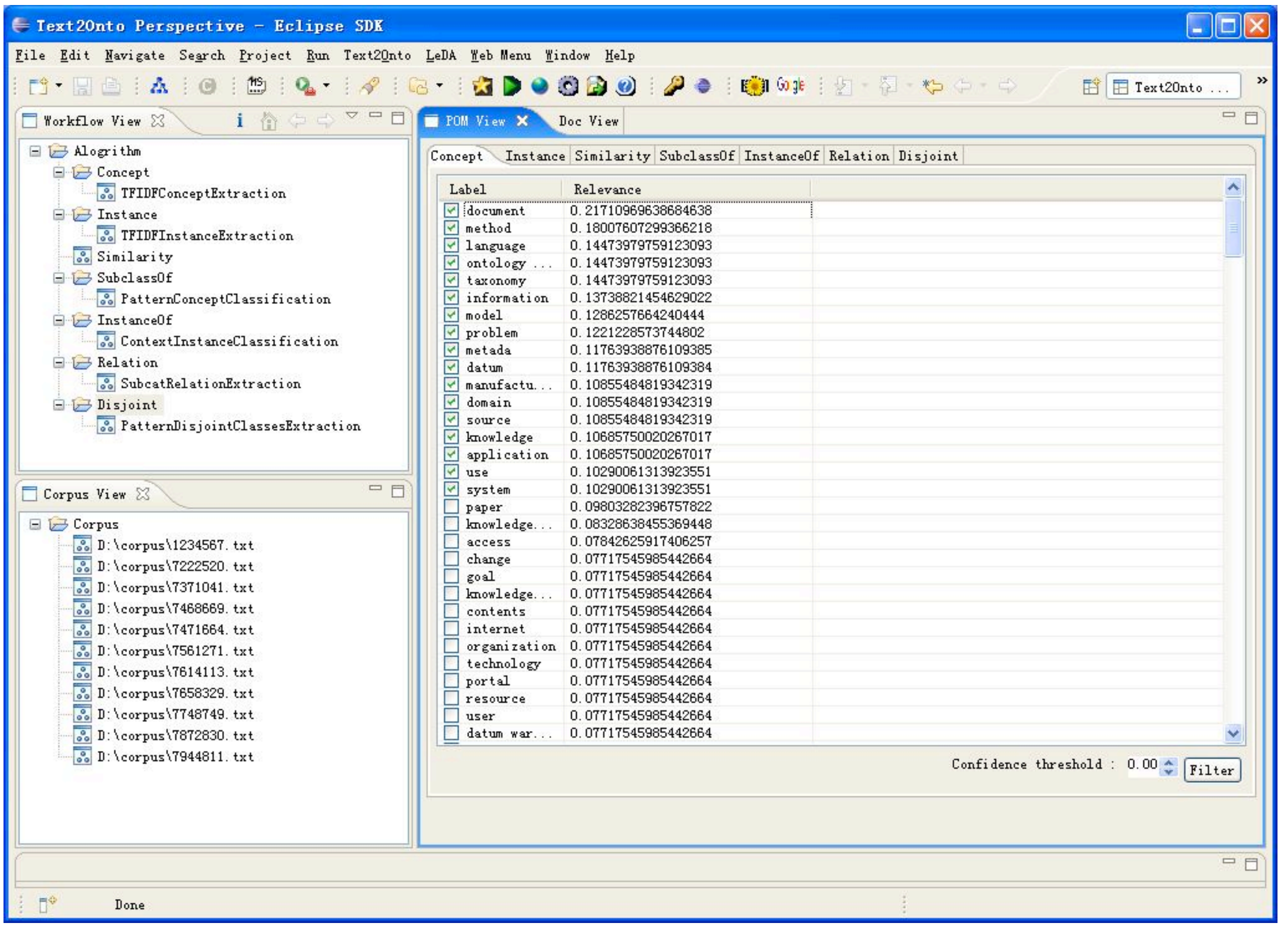
Figure 4.17: A usage example for RaDON plug-in - part 4.

is being developed, the number of instances, concepts and relations to be included in the ontology is typically huge (e.g. in the use case provided by the fisheries domain it involves thousands of concepts). In addition, it contains domain specific terms, which typically refer to domain specific literature, requiring a very high knowledge of the field to be understood, retrieved and then conceptualized. Furthermore, existing ontologies sometimes need to be mapped, for example by discovering semantic relations among their concepts. When the dimension of the ontology becomes huge, the number of concepts to be checked to perform such operation becomes unmanageable by means of a standard manual inspection or a keyword based search, forcing us to adopt semi-automatic techniques for mapping. The aim of this document is to demonstrate that by combining intelligent techniques for text processing, knowledge acquisition, reasoning, and by exploiting the semantic WEB infrastructure such as SWOOGLE, the effort required during the editorial process will be highly reduced, both in terms of the overall time required and in terms of coverage/accuracy of the ontology produced so far.

Benefits/Advantages of using TextOnto Even though the achieved results support the claim that a fully automatic process for ontology learning from texts is still a chimera, very effective and practical techniques are now available to achieve partial goals. Nonetheless, we performed punctual and well motivated experiments, which show the ease of applicability of the proposed methods, and suggest that the impact of adopting such techniques in a large scale ontology engineering process would be highly beneficial, allowing drastic reduction of time and human effort, avoiding subjective judgments and conceptual gaps, and therefore augmenting the overall quality of the produced ontology.

Implementation and usage 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.

The graphical user interface of the plugin (c.f. Figure 4.18) is very similar to the original Swing-based GUI of Text2Onto. It is composed of different views for the configuration of the ontology learning process and the presentation of the results.


Text2Onto Perspective - Eclipse SDK
File Edit Navigate Search Project Run Text2Onto LeDA Web Menu Window Help
Text2Onto ...
Workflow View
POM View
Doc View
Corpus View
Concept Instance Similarity SubclassOf InstanceOf Relation Disjoint
Confidence threshold : 0.00
Filter
Done

- Alogrithm
  - Concept
    - TFIDFConceptExtraction
  - Instance
    - TFIDFInstanceExtraction
  - Similarity
  - SubclassOf
    - PatternConceptClassification
  - InstanceOf
    - ContextInstanceClassification
  - Relation
    - SubcatRelationExtraction
  - Disjoint
    - PatternDisjointClassesExtraction

- Corpus
  - D:\corpus\1234567.txt
  - D:\corpus\7222520.txt
  - D:\corpus\7371041.txt
  - D:\corpus\7468669.txt
  - D:\corpus\7471664.txt
  - D:\corpus\7561271.txt
  - D:\corpus\7614113.txt
  - D:\corpus\7658329.txt
  - D:\corpus\7748749.txt
  - D:\corpus\7872830.txt
  - D:\corpus\7944811.txt

	Label	Relevance
✓	document	0.21710969638684638
✓	method	0.18007607299366218
✓	language	0.14473979759123093
✓	ontology ...	0.14473979759123093
✓	taxonomy	0.14473979759123093
✓	information	0.13738821454629022
✓	model	0.1286257664240444
✓	problem	0.1221228573744802
✓	metada	0.11763938876109385
✓	datum	0.11763938876109384
✓	manufactu...	0.10855484819342319
✓	domain	0.10855484819342319
✓	source	0.10855484819342319
✓	knowledge	0.10685750020267017
✓	application	0.10685750020267017
✓	use	0.10290061313923551
✓	system	0.10290061313923551
☐	paper	0.09803282396757822
☐	knowledge...	0.08328638455369448
☐	access	0.07842625917406257
☐	change	0.07717545985442664
☐	goal	0.07717545985442664
☐	knowledge...	0.07717545985442664
☐	contents	0.07717545985442664
☐	internet	0.07717545985442664
☐	organization	0.07717545985442664
☐	technology	0.07717545985442664
☐	portal	0.07717545985442664
☐	resource	0.07717545985442664
☐	user	0.07717545985442664
☐	datum war...	0.07717545985442664

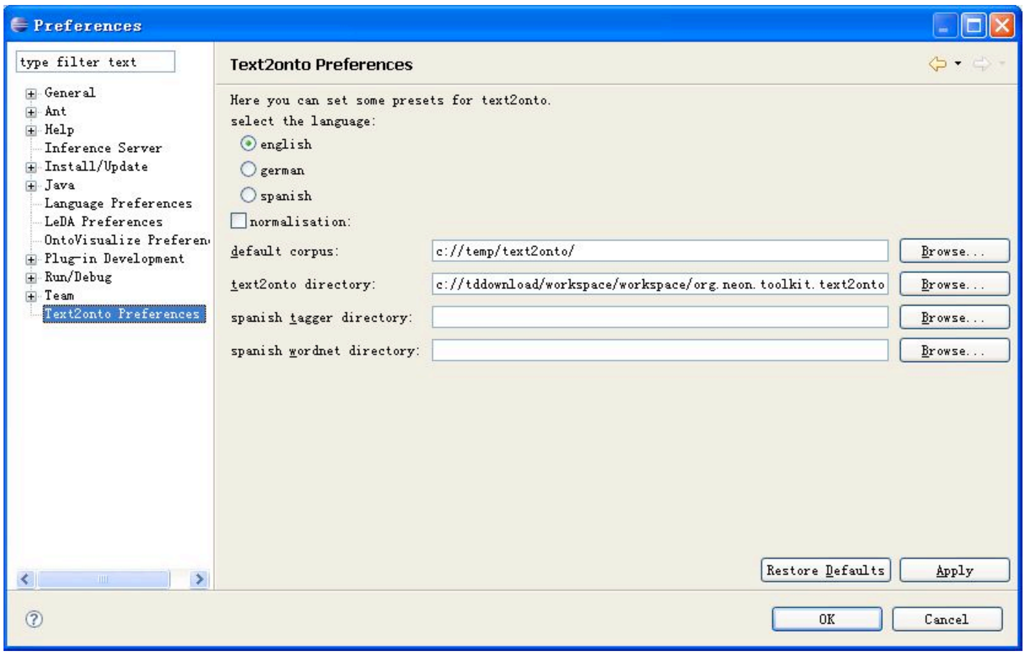
Figure 4.18: A usage example for Text2Onto plug-in.

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.

Preference The preference page (c.f. Figure 4.19), which is accessible from the main menu of on the top of the Text2Onto perspective (“Window” – “Preferences” – “Text2Onto Preference”) replaces the original configuration file of Text2Onto’s API. It allows for setting the following parameters:



Text2onto Preferences

Here you can set some presets for text2onto.

select the language:

[✓ english] german spanish

normalisation: ☐

default corpus: c://temp/text2onto/

text2onto directory: c://tddownload/workspace/workspace/org.neon.toolkit.text2onto

spanish tagger directory:

spanish wordnet directory:

Figure 4.19: The preference configuration GUI of Text2Onto plug-in.

- *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/Ènlp/web/index.php).

4.10 Watson Plug-in

The Watson plugin for knowledge reuse allows the developer of an ontology to automatically find all online ontologies conceptualizing the same domain, and to integrate their elements into the currently edited model. Watson plugin thus gives the capability of reusing what is already existing, facilitating the link between what is actually under construction (new knowledge), and what have been published online. Once the conceptualization and implementation is done there will be a set of connection generating a network, interlinking the models that are contributing to form the new one. Watson supports following use case:

- UC-11.4 Evaluate Structural Properties

Motivation of using Watson plugin 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.

Benefits/Advantages of Watson plugin There are several potential benefits and advantages of using Watson plug-in here in fishery ontology management system:

- *Fasten development:* Once the user becomes familiar with the mechanism of searching for existing models of what s/he wants to conceptualize, all the process of sketching or drafting an initial version of the new ontology with already published fragments of ontology speed up the development of this ontology. Next steps can consist of further refining according peculiar needs, with expansion or local reengineering. Moreover even when the elements are not directly reused, having an overview of conceptualizations made by others on the same domain facilitate the design of the considered ontology.

- *Push reusability:* Finding, selecting, extracting and integrating existing knowledge are the basic tasks required to enable reuse when building a new ontology (or even extending an existing one). By integrating a tool realizing all these tasks into an ontology engineering environment, making possible the incorporation of knowledge from the entire Semantic Web, the Watson plugin facilitates knowledge reuse so that it simplify the ontology development process rather than making it more complex.

- *Reduce design mistakes:* Reuse of knowledge helps inexperienced user in producing ontologies, as they can simply start from an initial concept name and then follow the suggested conceptualization from the plugin. Even for less amateur designers, by providing different examples of conceptualization of the same notions, the plugin allows the developer to chose the most appropriate one, avoiding possible modeling traps. In this sense, the plugin helps in producing better quality ontologies.

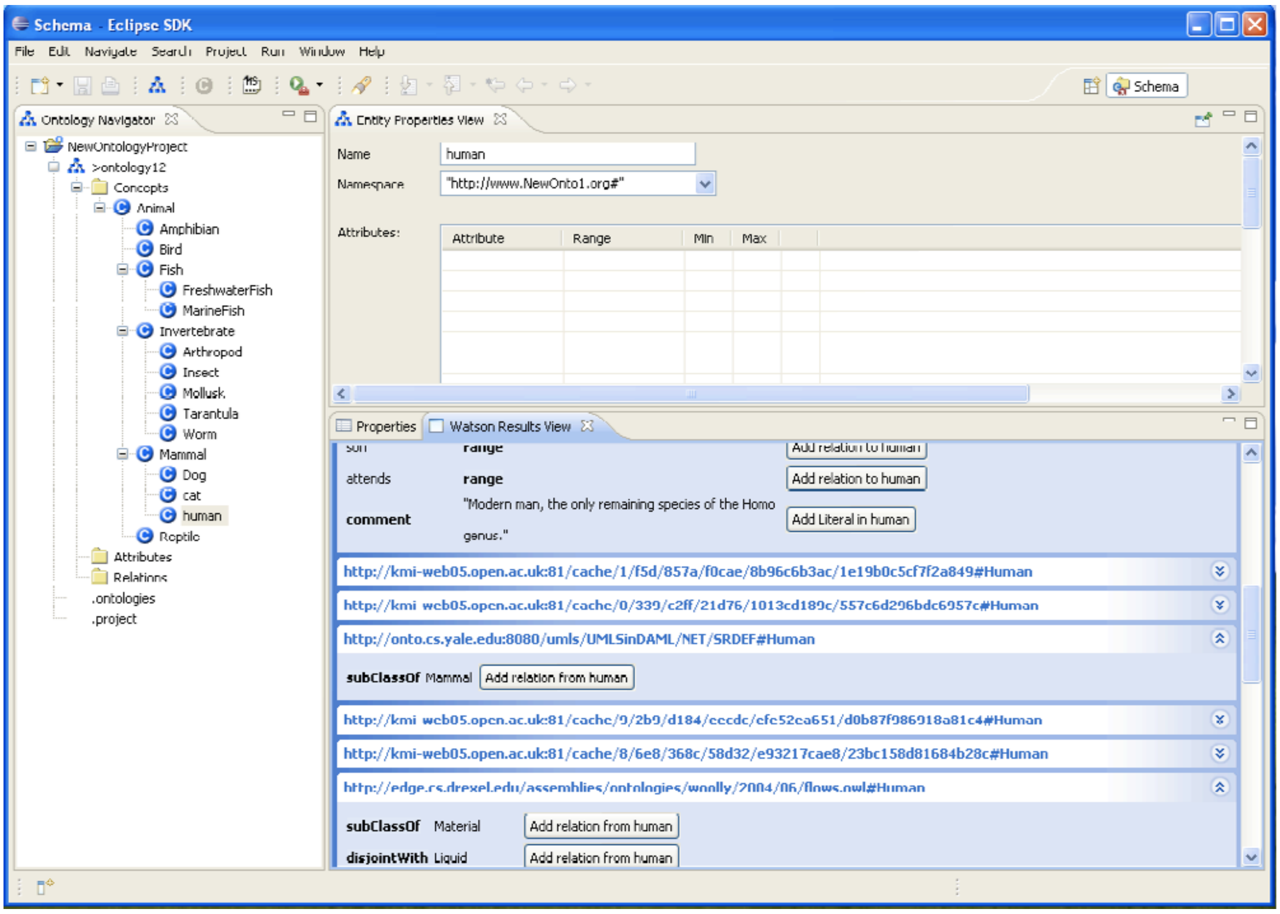
- *Generate mappings, networks, and reduce redundancy:* One of the issues related to the Semantic Web is that a number of different ontologies may be published on the same topic and domain. To make the most out of this knowledge, it is required to align these different models. Through knowledge reuse, the Watson plugin addresses this issue in two ways. First, since they are made from existing fragments of ontology, the models produced using the Watson plugin do not add the existing body of knowledge, and to its redundancy, but directly integrate with it. Second, reusing means integrating elements of several different ontologies, creating links between these models and the created one. Once, in turn, published on the Web, the new ontology is already networked, as it is interlinked with the ones that have been used to build it, and acts as an intermediary to interlink these different models.

Implementation and usage The Watson plugin for knowledge reuse allows the ontology engineer to query Watson and reuse results from within the NeOn Toolkit. For that purpose, it provides interface elements that seamlessly integrate with the graphical user interface of the NeOn toolkit (c.f. Figure 4.20). Through these interface elements, the Watson plugin allows the user:

- to select entities of the currently edited ontology he/she would like to inspect,

- to automatically trigger queries to Watson, as a remote web service. This web service can be accessed to retrieve the information of edited ontology and can be used to issue query,

- to view the result of these issued queries.



Name: human

Namespace: "http://www.NewOnto1.org#"

Attributes:

son range Add relation to human

attends range Add relation to human

comment "Modern man, the only remaining species of the Homo genus." Add Literal in human

http://kmi-web05.open.ac.uk:81/cache/1/f5d/857a/f0cae/8b96c6b3ac/1e19b0c5cf7f2a849#Human

http://kmi-web05.open.ac.uk:81/cache/0/339/c2ff/21d76/1013cd189c/557c6d296bdc6957c#Human

http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRDEF#Human

subClassOf Mammal Add relation from human

http://kmi-web05.open.ac.uk:81/cache/9/2b9/d184/cccdc/cfc52ca651/d0b87f986918a81c4#Human

http://kmi-web05.open.ac.uk:81/cache/8/6e8/368c/58d32/e93217cae8/23bc158d81684b28c#Human

http://edge.cs.drexel.edu/assemblies/ontologies/woolly/2004/06/flows.owl#Human

subClassOf Material Add relation from human

disjointWith Liquid Add relation from human

Figure 4.20: A usage example for Watson plug-in.

Results of these queries, i.e. semantic descriptions of the selected entities in online ontologies, are displayed in an additional view allowing further interactions (see Figure 1).

Finally, the core of the plugin is the component that interacts with the core of the NeOn toolkit: its datamodel. Statements retrieved by Watson from external ontologies can be integrated in the edited ontology, requiring for the plugin to extend this ontology through the NeOn toolkit datamodel and data management component. Finally, thanks to the common infrastructure provided by the NeOn toolkit, interactions with other plugins, supporting other activities in the lifecycle of ontologies, can be envisaged, like the use of the plugin managing mappings between ontologies to keep track of the external resources included by the Watson plugin.

Chapter 5

Examples of Integrated Use Cases

Here we provide three integrated use cases which all contains two or three plug-ins to complete a common ontology editing and management task. However, it is obvious to see they are just a subset of all potential usages in the first prototype for fishery ontology management. We are not going to show all possible usages as we think these integrated use cases are typical ones in FAO's daily work.

1. Managing networked ontologies in a decentralized network
2. Consistency checking of ontologies populate from text
3. Reusing ontologies populated from database

It is essential to prove that our approach of integration is well applicable for developing such a multiple component system from different parties, so that we can follow on this direction for further development and deployment of an integrated application within Workpackage 7 and NeOn project.
Next we describe the integrated use cases based on extracting subsets of Figure 3.2, aligning with the data sets provided in Chapter 2, in the mode of workflow. In all use cases, the fundamental NeOn API and Core NeOn Toolkit plug-in are mandatory components to work with.

5.1 Managing Networked Ontologies in a Decentralized Network

In this integrated use case, our purpose is to manage networked ontologies in a peer-to-peer network. For example, a user wants to access a certain remote ontology to create a mapping between this remote ontology with local ontology, and publishes the created ontology mapping together with the mapped local and remote ontologies. In this case, the following plug-ins are required:

- NeOn API (Datamodel Plug-in)
- Engieering / GUI components in Core NeOn Toolkit
- Oyster plug-in and service
- OntoMap

The interactions among data within these plug-ins are illustrated in Figure 5.1.
If we work with FAO data, a typical workflow in steps for this integrated use case is:

1. Open NeOn Toolkit ontology navigator;
2. import an ontology into local ontology project;

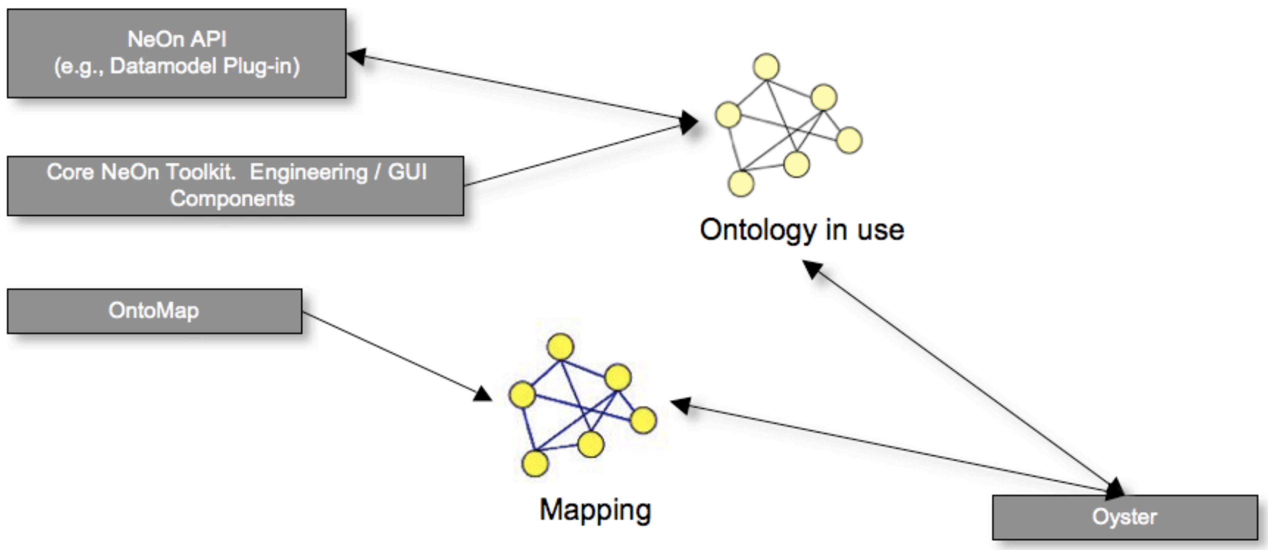

NeOn API
(e.g., Datamodel Plug-in)
Core NeOn Toolkit. Engineering / GUI Components
Ontology in use
OntoMap
Mapping
Oyster

Figure 5.1: Interactions between components in performing tasks of managing networked ontologies in decentralized network.

3. open the Oyster Import Wizard;
4. choose a remote ontology that is managed by remote Oyster peers in distributed repository;
5. load the chosen ontology into local ontology project;
6. choose ontology for schema mapping with ontology;
7. publish the mapping using Oyster web services so that remote peers are able to access and reuse this mapping.

5.2 Consistency Checking of Text-Populated Ontologies

In this integrated use case, we need to check the consistency of ontologies that are populated from text. Populate ontologies from text using text mining techniques is one hot topic in ontology engineering research. People often bootstrap their ontologies by extracting the concepts and relationships from text manually, which is extremely time consuming and labor intensive. Nowadays, automatic ontology generator from text enables people to extract the knowledge from text much easier and faster, however, the populated ontologies are usually error-prone. Therefore, it is necessary to use a debugging and consistency checking software to help people in refining these populated ontologies. In this case, the following plug-ins are required:

- NeOn API (Datamodel Plug-in)
- Engieering / GUI components in Core NeOn Toolkit
- Text2Onto plug-in
- RaDON

The interactions among data within these plug-ins are illustrated in Figure 5.2.

If we work with FAO data, a typical workflow in steps for this integrated use case is:

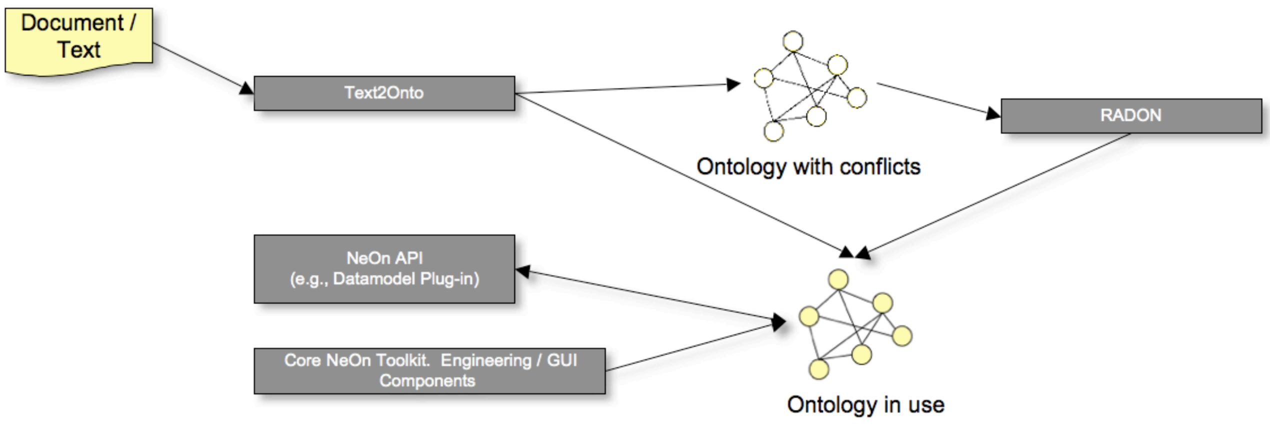

Document / Text
Text2Onto
Ontology with conflicts
RADON
NeOn API (e.g., Datamodel Plug-in)
Core NeOn Toolkit. Engineering / GUI Components
Ontology in use

Figure 5.2: Interactions between components in performing tasks of consistent checking of text-populated ontologies.

1. Open Text2Onto perspective in NeOn Toolkit;
2. choose an algorithm for extracting concepts and properties, here we use ASFA abstracts and FIGIS fact sheets as documents;
3. choose an target corpus from which the populated ontology is expected;
4. run Text2Onto to have a populated ontology;
5. export this populated ontology into local NeOn Toolkit ontology project;
6. use RaDON to check the consistency of this ontology and find out where the inconsistencies are;
7. execute possible debugging activities to have a ontology that is less error-prone.

5.3 Reusing Ontologies Populated from Database

The fishery departments in FAO poses huge size of databases as the backbones of their information systems. There have been a challenge regarding how to transfer the data from relational databases into data that can be easily accessed by modern semantic technologies. For example, export database to ontology is a hot research topic in the Semantic Web community. ODEMapster creates ontologies by exporting instance data from database. However, these data are usually preliminary as there is a logical gap between relational database and ontologies – the ontologies often rely on Description Logics [BCM+03] while the classic relational databases are not. We need to refine the ontology populated from databases using ontology refinement tools to enhance the reusability of these populated ontologies. Thus, Watson can be applied here.

- NeOn API (Datamodel Plug-in)
- Engieering / GUI components in Core NeOn Toolkit
- ODEMapster
- Watson

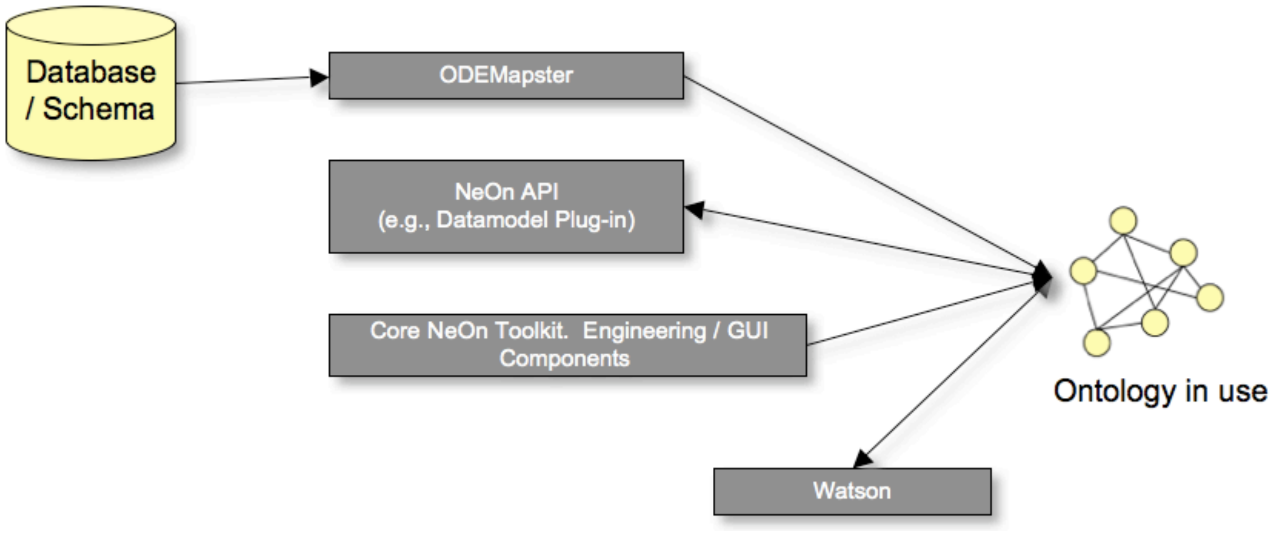

Database / Schema
ODEMapster
NeOn API (e.g., Datamodel Plug-in)
Core NeOn Toolkit. Engineering / GUI Components
Watson
Ontology in use

Figure 5.3: Interactions between components in performing tasks of reusing ontologies created by importing from database.

The interactions among data within these plug-ins are illustrated in Figure 5.3.
If we work with FAO data, a typical workflow in steps for this integrated use case is:

1. Assume we have database;
2. export database to local ontology by using the import wizard provided by ODEMapster: Tthe user inputs database information, selects the ontology file (rdf(s) or owl), selects the R2O file, selects the query file and then the plugin creates an RDF instances file into the file system;
3. choose the imported RDF file, launch Waston;
4. the user can use Waston to refine the ontology. (c.f. Section 4.10)

Chapter 6

Next Steps

There are several plug-ins are planned for three central use cases in the next prototype of fishery ontology lifecycle management system:

1. Support of ontology editorial workflow
2. Support of ontology modularization
3. Support of semi-automatic mapping discovery

Further, other use cases, such as ontology documentation, natural language query over ontologies and ontology evaluation are planned for the final prototype.

6.1 General Actions

As next steps, we are going to proceed first by identifying the individual responsibilities of software development tasks from different partners. We track the unsolved use cases and address these use cases into detailed software components that are going to be integrated to NeOn Toolkit plug-in. We elaborate quality assurance by using bug reporting, tracking and fixing mechanism. The person that is responsible for coordinating T7.4 control and entire procedure in general by monitoring the development of each individual plug-in.

6.2 Ontology Editorial Workflow

6.2.1 Use Cases

A number of operations are required in order to support the editorial workflow. Users may have different roles in the workflow, and assigned different operations to be performed in different moments. For a detailed description of the editorial workflow required for the ontology lifecycle, please see deliverable D7.4.1 [MGKI$^+$07].

UC-15.1: Edit Ontology Element Ontology editors (subject experts and validators) are allowed to edit ontology elements, depending on the status of the elements and their user rights and roles.

UC-15.2: Delete an ontology element Ontology editors are able to propose for deletion and ontology element that is in the “Approved” status. In any case this is not a definitive action, and the element is automatically assigned the “To be deleted” status. Only validators are able to definitely destroy an element in the “To be deleted” status.

UC-15.3: Change status of element While inserting, updating and deleting elements, the status of those are automatically changed, when required, by the system. There are other cases where a specific action by

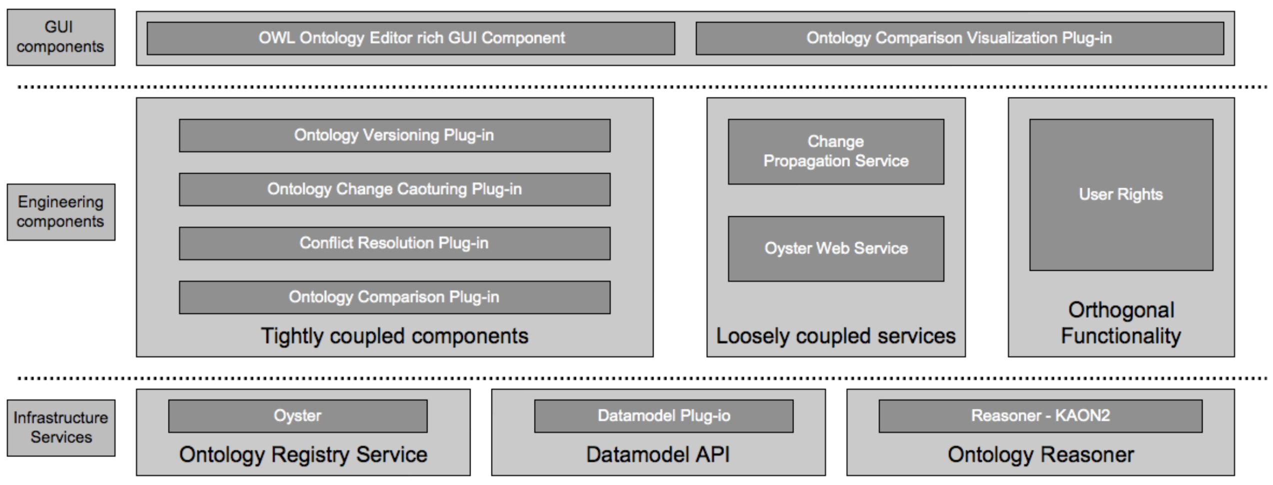

GUI components
OWL Ontology Editor rich GUI Component
Ontology Comparison Visualization Plug-in
Engineering components
Ontology Versioning Plug-in
Ontology Change Caoturing Plug-in
Conflict Resolution Plug-in
Ontology Comparison Plug-in
Tightly coupled components
Change Propagation Service
Oyster Web Service
Loosely coupled services
User Rights
Orthogonal Functionality
Infrastructure Services
Oyster
Ontology Registry Service
Datamodel Plug-io
Datamodel API
Reasoner - KAON2
Ontology Reasoner

Figure 6.1: The overall architecture of workflow for ontology editing related components

Ontology editors is required to move an element from one status to another and make the editorial workflow to function.

UC-15.4: Publish Ontology Validators are allowed to copy an ontology where all its elements are in the "Approved" status, from the test and validation environment (editorial workflow in the Intranet) to the production environment (Internet). By doing so, the system automatically assigned the right version to the published ontology. From V1 for the first time a particular ontology goes live, to Vn+1 for the N upgrade of the ontology.

UC-15.5: Wiki based Editing of Ontology Elements Validators are allowed to browse and modify the ontology by adopting the Wiki based visualization interface. The generated portal, representing ontology elements, allows subject experts and validators to perform a community based editing/validation.

6.2.2 Plug-in Plan – Supporting of Workflow Model

By aligning to general NeOn architecture introduced in WP6 [GAGW05], we depict the software architecture of workflow for ontology editing related components.

Figure 6.1 represents the overall architecture of workflow for ontology editing related components, which are going to be implemented in the second prototype. We can see there are three layers in this architecture:

- Schema Modeling in OWL Ontology Editor as central part in realizing user interface in the software, while we both provide rich and thin GUI components based on the requirement of software component itself.
- In NeOn architecture, we have identified the difference between loosely and tightly coupled engineering components. Apparently, Oyster and Change Propagation web services are classified as loosely coupled engineering components, and As we consider our components to be distributed over network, we implemented a distributed software component management infrastructure to coordinate all distributed software components.
- The updated Oyster ontology registry services supports the workflow related tasks with underlying ontology reasoner – KAON2.

However, we are not going to implement all functions at once. Basic components, such as Change Capturing plug-in, are fundamental requirements to implement other functions , such as ontology versioning. Obviously, Ontology Versioning requires capturing the ontology changes at first. Therefore, following components are included in the second prototype of fishery lifecycle management system:

- Change Capturing plug-in with Status Management
- Ontology Comparison plug-in
- Change Propagation plug-in
- the Oyster aims to provide web services for propagating information in distributed ontology registry as the role of ontology registry service;

6.3 Ontology Modularization

6.3.1 Use Cases

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

UC-8.1: Create ontology module This use case is related to the ontology partitioning activity. The system allows the ontology engineer to create ontology modules. The ontology engineer specifies the branch of the ontology (or a set concepts), a set of properties and a set of instances to be included in the module (and the metadata of the module, name and comments).

UC-8.2: Modularize semi-automatically This use case is related to the ontology partitioning and to the ontology module extraction activities. The system allows ontology engineers to modularize an ontology in a semi-automatic way. The system returns the resulting candidate modules and the related ontology elements. The ontology engineers select from the proposed modules, those that they want to create, including the ontology elements involved in each module.

UC-8.3: Merge module This use case refers to the ontology merging activity. The system allows ontology engineer to merge a set of modules. The system creates a unique module from the original modules. Requirement: this use case was added during T7.4 analysis of requirements and was not originally included in D7.1.1.

6.3.2 Plug-in Plan - Support of Ontology Modularization

It is a common opinion that ontology engineering shares a lot with software engineering, and the advantages of having modular ontologies instead of big and monolithic ones are easy to understand. Ontology modules are made to be reusable knowledge components, facilitating collaborative design and maintenance within a network of ontologies. Therefore we list several points that will be taken into consideration throughout the development of ontology modularization plug-in.

- A Module is an Ontology.
- Encapsulation/Information Hiding.
- Partial Import.
- Links Between Modules.

For detailed information regarding the design of modular ontology metadata on which the ontology modularization plug-in depend, we refer readers to NeOn deliverable D1.1.2 [HBP$^+$07].

6.4 Automatic Mapping Discovery

6.4.1 Use Cases

UC-6.2: Create mappings semi-automatically The ontology mappings can be created and supported by software plug-in; the user selects the ontologies, the type of ontology elements, and preferred relationships to be instantiated, finally the software lists discovered candidates to be confirmed as mapped by the user.

6.4.2 Plug-in Plan – FOAM / Alignment Server

We need provide tools to fully or semi-automatically align two or more OWL ontologies. In FOAM / Alignment Server plugins, we plan to provide the following functions:

- Choose base matchers: (1) Label based matcher, (2) comment based matcher, (3) instance based matcher and (4) structure based matcher.
- Choose combination operator: (1) Average operator, (2) weighted operator, (3) Maximal operator, (4) at least half OWA operator, (5) most OWA operator and (6) as many as possible OWA operator.
- Configure some parameters: (1) The file path of existing mappings if available, (2) the number of iterations to be executed, (3) use complete alignment or efficient one, (4) the threshold to obtain the correct mappings from candidate ones.

Chapter 7

Summary and Lessons Learned

7.1 Summary

Following the work in Task 7.4, this deliverable described the first prototype for managing the fisheries ontologies lifecycle following the architecture, requirements and use cases described in D7.4.1. The lifecycle described in this deliverable was an instantiation of the possible lifecycles that could be carried out with the NeOn toolkit [1], and has been selected according to requirements from FAO.

To be specific, this deliverable's contributions are outlined as below:

- Chapter 1 discussed the scope of this deliverable, the motivation of our work, the approach adopted here and how this deliverable is organized.

- Chapter 2 introduced the up-to-data requirements elaborated from FAO case study. FAO people also provided data and relevant use cases as guidelines and benchmark for testing the prototype developed along with this deliverable.

- Chapter 3 depicted the overall architecture of fishery ontology management system at the stage of first prototype. There were three major concerns of this chapter: (1) Clarifying the mappings between uses cases raised by FAO and individual components developed by different partners within NeOn project; (2) figuring out the relationships between data sets used by FAO for their daily work and the potential components that processed these data sets; (3) designing the architecture for the first prototype by giving an overview, finding out possible conflicts by analyzing interactions among different components, and covering the interactions between data sets and components.

- Chapter 4 was a central part to introduce the major components that were developed and tailored for FAO case study: (1) Core NeOn Toolkit, with (2) LabelTranslator, (3) ODEMapster, (4) OntoMap, (5) Oyster, (6) RaDON, (7) SAFE, (8) Text2Onto and (9) Watson. These plug- ins were integrated with Core NeOn Toolkit to support the fishery ontology lifecycle management.

- Chapter 5 provided four integrated use cases which contained two or three plug-ins to complete a normal ontology editing and management task. The successful and smooth integration of different components as NeOn Toolkit plug-ins proved that our approach for integration was well applicable for developing such a multiple component system from different parties.

- Chapter 6 outlined the next steps after this deliverable and corresponding first prototype.

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

7.2 Lessons Learned

During the work in T1.4 and the development of the first prototype of fishery ontology lifecycle management system, there are severable challenging situations and problems occurred. In this section, we will discuss these issues that are going to be noted in the future work.

1. Integrating different NeOn plug-ins requires multiplied efforts compare to developing several individual efforts.
2. The developed plug-in may go other way around to the original requirements from case study.
3. Compatibilities problem may occur once an individual component is being used by several other components.

The first challenge is often seen in integrating several individual components into a single system that consists of several plug-ins. As different partners may use different standards and procedures to elaborate their plug-ins, it is extremely different in coordinating all the actions in general. For example, software documentations are written in different manner so that other people may find the formats of documents are different and difficult to follow, while only the original developers are able to provide an accurate and correct documents. This is just one aspect of coordination, other aspects, like scheduling the development cycle, meeting deadlines, are also very challenging.

The second problem probably even more crucial. For example, the users had already provided the requirements to the developers in a face-to-face manner, and the developer also thought they had already realized the requirements correctly. However, in the end, the users reported the software delivered did not accurately meet the requirements. In cases like this, the only alternative to reworking the entire software at the very end of the development process, is to have frequent updates between developers and users.

In our integrating procedure, we have also found it is problematic to share some specific components in several other components. In particular, Oyster is running as a registry service, if several different components invoke Oyster simultaneously, the system will crash as only one Oyster instance is allow each time. This requires a central control of such publicly shared components, i.e., we need to build another component to control these share components separately for other components.

In a nutshell, this deliverable provided a first prototype for managing fishery ontology lifecycle by integrated different components from multiple parties. The integrated uses cases proved that our approach for integration was well applicable and could be followed on. The outcome of testing integrated use cases based on the FAO data sets showed that the first prototype had met the requirements from individual uses cases.

Bibliography

[BCM+03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[Car07] Caterina Caracciolo. D7.2.2 – revised and enhanced fisheries ontologies. Technical Report D7.2.2, FAO, September 2007.

[EW07] Michael Erdmann and Dirk Wenke. D6.6.1 – realisation & early evaluation of basic neon tools in neon toolkit v1. Technical Report D6.6.1, Ontoprise GmbH, September 2007.

[GAGW05] Jose Manuel Gómez, Carlos Buil Aranda, Oscar Munoz Garcí, and Walter Waterfeld. D6.1.1 – requirements on neon architecture. Technical Report D6.1.1, iSOCO, September 2005.

[HBP+07] Peter Haase, Saartje Brockmans, Raul Palma, Jérôme Euzenat, and Mathieu d'Aquin. Updated version of the networked ontology model. Technical Report D1.1.2, University of Karlsruhe, AUG 2007.

[HRW+06] Peter Haase, Sebastian Rudolph, Yimin Wang, Saartje Brockmans, Raul Palma, Jéróme Euzenat, and Mathieu d'Aquin. D1.1.1 networked ontology model. Technical Report D1.1.1, Universität Karlsruhe, NOV 2006.

[MGKI+07] Óscar Munoz-García, Soonho Kim, Marta Iglesias, Caterina Caracciolo, Andrew Bagdanov, Yimin Wang, Peter Haase, María del Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. D7.4.1 – software architecture for managing the fisheries ontologies lifecycle. Technical Report D7.4.1, UPM, September 2007.

[PHWd07] Raul Palma, Peter Haase, Yimin Wang, and Mathieu d'Aquin. D1.3.1 propagation models and strategies. Technical Report D1.3.1, Universidad Politécnica de Madrid, NOV 2007.

[WHP07] Yimin Wang, Peter Haase, and Raúl Palma. D1.4.1 prototypes for managing networked ontologies. Technical Report D1.4.1, Universität Karlsruhe, FEB 2007.

[WWH+06] Walter Waterfeld, Moritz Weiten, Peter Haase, Hamish Cunningham, Martin Dzbor, and Óscar Munoz García Raúl Palma. D6.2.1 – specification of neon reference architecture and neon apis. Technical Report D6.2.1, Software AG, March 2006.