NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D1.5.2 Implementation of Metadata Evolution

Deliverable Co-ordinator: **Diana Maynard**

Deliverable Co-ordinating Institution: **University of Sheffield (USFD)**

Other Authors: **Niraj Aswani (USFD); Wim Peters (USFD); Sofia Angeletou (OU); Mathieu d’Aquin (OU)**

This document describes the implementation of the approach to modelling some of the dynamics of (semantic) metadata that were described in D1.5.1 [MPD+07]. The deliverable comprises three main parts. First, we describe an implementation of the methodology for capturing changes to the metadata as a result of ontology evolution such as deleted concepts. We have developed an example client in GATE, known as the NeOnOntologyServiceLR, which implements the refined ontology API and which connects the GATE services with the NeOn ontology services. In the second and third parts, we describe the implementation of two methodologies for capturing changes to the ontology resulting from the metadata: SARDINE (an instance of the SAFE plugin) and a folksonomy application.

Document Identifier:	NEON/2008/D1.5.2/v1.0	Date due:	February 29, 2007
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	February 29, 2007
Project start date	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

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

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 11 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SL–1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier, France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield, United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Kolenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

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

- University of Sheffield
- Open University
- University of Karlsruhe

Change Log

Version	Date	Amended by	Changes
0.1	01-01-2008	Diana Maynard	First template
0.2	18-01-2008	Diana Maynard	Description of SARDINE and OntologyAPI
0.3	30-01-2008	Diana Maynard	Description of OU work on folksonomies
0.4	04-03-2008	Diana Maynard	Changes following reviewer's comments

Executive Summary

This document describes the implementation of the approach to modelling some of the dynamics of (semantic) metadata that were described in D1.5.1 [MPD$^+$07]. The deliverable comprises three main parts. First, we describe an implementation of the methodology for capturing changes to the metadata as a result of ontology evolution such as deleted concepts. We have developed an example client in GATE, known as the NeOnOntologyServiceLR, which implements the refined ontology API and which connects the GATE services with the NeOn ontology services. This is based on the change typology described in D1.5.1, the idea being that when changes are made to the ontology (e.g. new concepts added, deleted or moved), such changes should also be reflected in the metadata (instances) in order to preserve potentially valuable information. In the second and third parts, we describe the implementation of two methodologies for capturing changes to the ontology resulting from the metadata. The SARDINE application is an instance of the SAFE plugin which extracts relational information about fish from a set of texts and suggests new additions to the ontology, using information extraction techniques based on GATE. It makes use of linguistic patterns to find examples of relations such as hyponyms and synonyms between known fish names and unknown entities, which can then be added as new concepts in the ontology according to the relations proposed. This is realised as part of the SAFE plugin for the NeOn toolkit. The folksonomy application investigates how information gathered from folksonomies can be used to improve existing fisheries ontologies. The method is a complementary approach to the SARDINE method, but uses folksonomies rather than textual documents as a starting point, and makes use of lexical and conceptual information filtering to generate new potential changes to the ontology structure (such as new concepts to be added). Both of these two applications are based on the Fisheries Use Case in NeOn.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This document describes the implementation of the approach to modelling some of the dynamics of semantic metadata that were described in D1.5.1 [MPD$^+$07]. This is motivated by the need to propagate changes in an ontology to its metadata, and vice versa. For example, if a user deletes concepts from the ontology, it is important to have a mechanism for dealing with any associated metadata, in order not to lose vital information. If a user adds new concepts to the ontology, then it may be necessary to return to the text to check whether additional instances can be found which should be used to populate these new concepts in the ontology. Furthermore, not only are ontologies dynamic and subject to structural change, but so are the texts and instances from which the ontologies may be derived. If we get additional relevant textual material and/or find new instances in that text, it may be necessary to modify the ontology to take into consideration this new information (for example, adding new concepts or new relations between existing concepts in the ontology).

It is important to note carefully the distinction between semantic metadata and ontology metadata, as described in D1.5.1. Semantic metadata refers essentially to metadata describing *meaning* (in the texts), while ontology metadata refers to metadata describing the ontology and its *content*. So texts are annotated with semantic metadata which indicate, for example, that a certain phrase contains an instance of a concept in the ontology: this metadata may thus be used to populate the ontology. Here we can therefore generally understand (semantic) metadata as referring to instances in the ontology. In this deliverable we shall use the term metadata to refer exclusively to semantic metadata rather than ontology metadata. The overall goal of NeOn's work package 1 is to develop an integrated approach to the evolution process of networked ontologies and related metadata. Within this context, the more specific goal of T1.5 is to capture the evolution of metadata due to changed concepts, relations or related metadata in one of the networked ontologies, and to capture changes to the ontology caused by the metadata. In D1.5.1 we proposed a methodology to capture (1) the evolution of metadata and (2) changes to the ontology. In this deliverable, we describe the actual implementation of our approaches for evolution of metadata and evolution of ontologies.

Chapter 2 describes the mechanisms we have implemented in GATE to deal with changes to the ontology and the effect this has on the metadata. Chapters 3 and 4 describe the mechanisms implemented for dealing with changes to the ontology caused by the introduction of metadata: the first of these describes our SARDINE application which analyses fisheries texts and generates potential new concepts for the ontology; Chapter 4 describes some experiments with folksonomies to determine how changes in folksonomies can lead to changes in a related ontology.

Chapter 2

Mechanisms for dealing with ontology changes

2.1 Introduction to GATE and its Ontology API

The approach to handling the evolution of metadata is implemented in GATE [CMBT02]. GATE is the architecture (and framework for the IE components) for the SAFE plugin in NeOn. SAFE (Semantic Annotation Factory Environment) consists of various GATE Services (GaS) which perform annotation on text[1]. Figure 2.1 depicts the architecture of the relationship between the NeOn Toolkit (NTK), GATE, and SAFE. As shown in the diagram, one instantiation of SAFE in the toolkit is in the form of the SARDINE application, described in more detail in Section 3.1, which analyses text with respect to an existing ontology and makes suggestions for new concepts to be added to the ontology. Other SAFE applications will include ANNIE, a generic Information Extraction system for analysing textual documents [MTC$^+$02], and applications for annotating different kinds of information. GATE also contains mechanisms for creating, viewing and editing ontologies [BTMC04].

In addition to the generic OWLIM[2] Ontology Language Resource (LR) in GATE, we have created a new Ontology LR which is designed to take into account changes to the ontology in a way which preserves related data, as proposed in D1.5.1. In the following sections, we describe the GATE Ontology Services and the Ontology Event Model.

2.2 GATE Ontology Services

The GATE Ontology Service (GOS) is based on OWLIM [Kir06]. OWLIM stands for OWL In Memory and is a Semantic Repository developed in Java. It is packaged as a Storage And Inference Layer (SAIL) for the Sesame RDF database. The GOS allows users to upload a knowledge base to the server by creating a new repository for individual ontologies and provides various service methods to add, delete and modify data into the repository. Below we give a brief overview of the GATE Ontology Services and a client, known as NeONOntologyServiceLR, that has been developed for accessing the GATE Ontology Services.

There are more than 130 methods published as part of the GOS. These methods allow creating new repositories on the server running GOS and storing ontological data in them. These methods allow population and manipulation of the data in repositories. The web service definition file[3]. (WSDL) lists these methods and gives more details about the parameters that need to be provided with each method. Below we list some of these methods.

- createRepositoryFromUrl

[1]See http://www.gate.ac.uk/projects/neon/safe-plugin.html for more information
[2]http://www.ontotext.com/owlim/
[3]The WSDL file can be downloaded from http://gate.ac.uk/sale/gos/owlim.wsdl

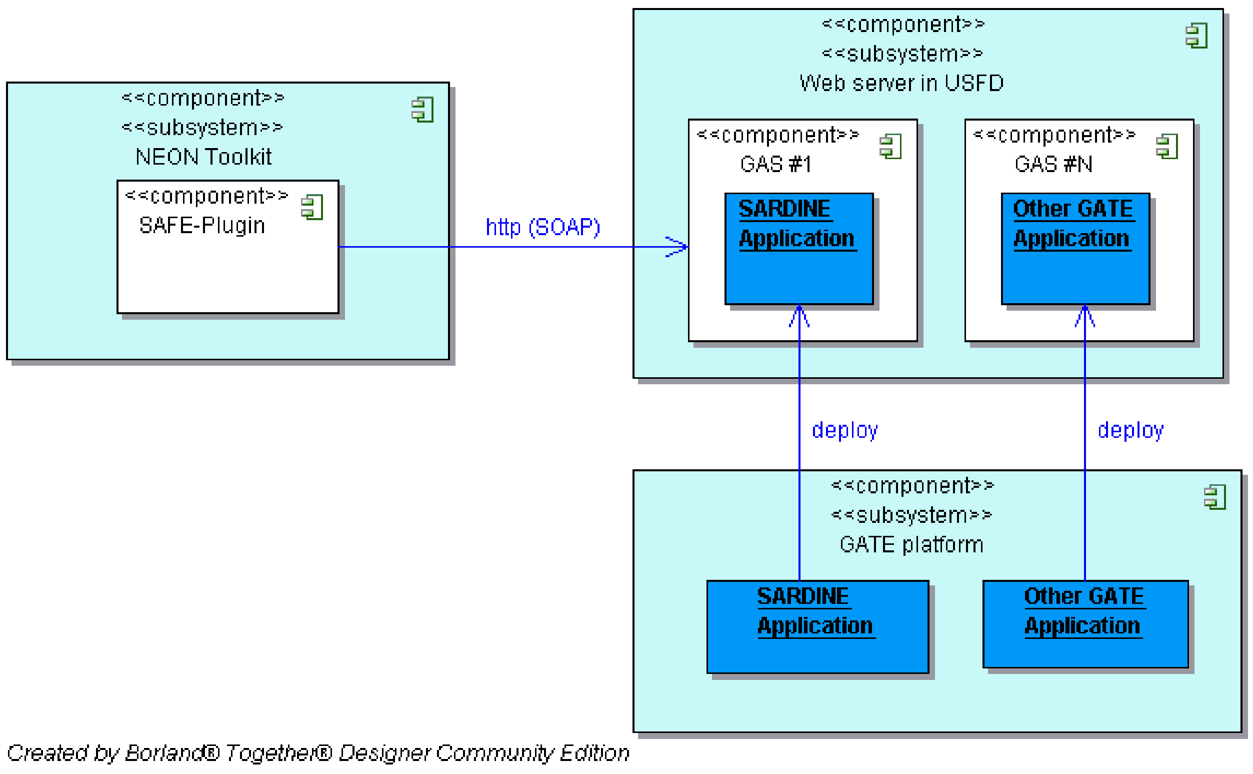


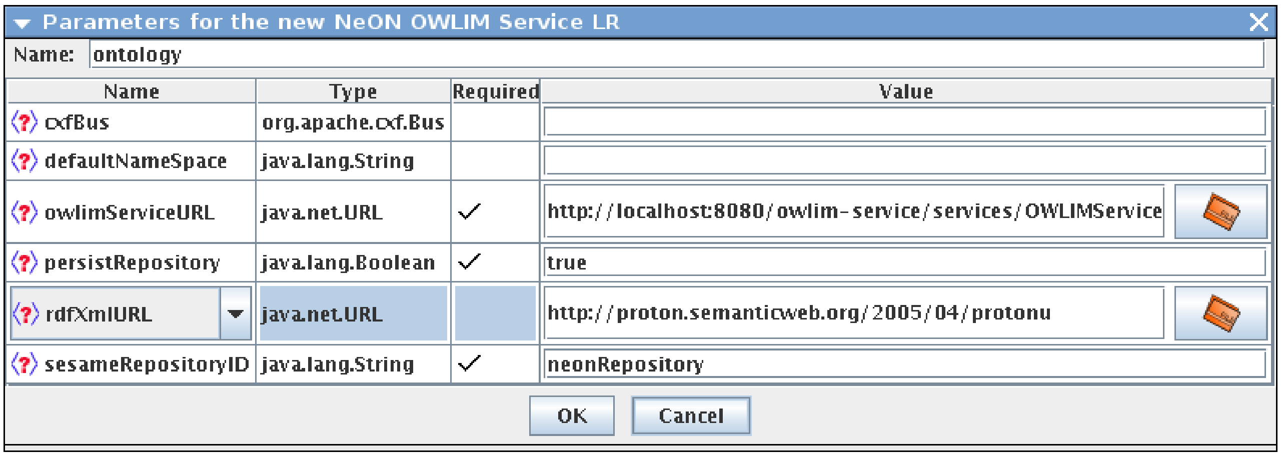
Figure 2.1: Relationship between GATE, SAFE and the NeOn Toolkit

- addClass
- isTopClass
- addSubClass
- getSubClasses
- addIndividual
- addAnnotationProperty
- addDatatypeProperty
- addObjectProperty
- addAnnotationPropertyValue
- removeObjectPropertyValues
- setEquivalentClassAs
- getDomain
- ... etc.

Implementation of the services is based on the the GATE Ontology API. More information on this can be found at http://gate.ac.uk/sale/tao/#chapt:ontologies. However, to meet the requirements of the NeON project, certain changes had to be made to the GATE Ontology Services. Below we list these NeON-specific changes.

2.2.1 NeONOntologyServiceLR

To illustrate a use of the GATE Ontology Services, we have developed an example client in GATE, known as the NeONOntologyServiceLR. It implements the GATE Ontology API but internally talks to the GATE Ontology Services. In other words, the client, when initiated, establishes the connection to the server running GOS and thereafter communicates with it to obtain various pieces of information stored on the server. Figure 2.2 shows a screenshot of the parameter window to establish connection with the GOS.



Name: ontology

Name	Type	Required	Value
cxfBus	org.apache.cxf.Bus		
defaultNameSpace	java.lang.String		
owlimServiceURL	java.net.URL	✓	http://localhost:8080/owlim-service/services/OWLIMService
persistRepository	java.lang.Boolean	✓	true
rdfXmlURL	java.net.URL		http://proton.semanticweb.org/2005/04/protonu
sesameRepositoryID	java.lang.String	✓	neonRepository

Figure 2.2: Parameters for establishing connection with GOS

This information is then displayed in the GATE Ontology Editor (see Figure 2.3) which allows users to manipulate data into the repository. It provides various functions including creating a new repository on the server by providing a URL of the ontology, saving the ontological data into a file on the local system and obtaining the change log (explained later) since the original state of the ontology. Users can save ontologies into various formats (rdf/xml, ntriples, n3 and turtle). The client also allows exporting the events log into a file by selecting the "Save Ontology Event Log" option from the options menu. Similarly, users can also load the exported event log and apply the changes on a different ontology by using the "Load Ontology Event Log" option. Figure 2.4 shows a screenshot of the options in the editor allowing users to save the ontology in different formats and to load and export the ontology event log.

2.2.2 Event logging

Ontologies stored on the server can be accessed by various people/applications from all over the world. Different people can introduce different changes to the ontology, and therefore we need to keep track of the changes made to the ontology as they arise. Each change made to the ontology is logged onto the server. Users of the GOS can then download these changes by using one of the service methods published by GOS. Any change made to the ontology can be described by a set of triples either added to or deleted from the repository. For example, when a new class is added to the ontology, the following statement is added to the repository:

```
<classURI> <rdf:type> <owl:Class>
```

Below we give illustrations of service methods and resulting event logs collected from the server running GOS. Each line in the event log entries is a triple describing the change made to the ontology.

```
// Adding a new instance "Rec1" of type "Recognized"
// 1st argument = name of the repository
```

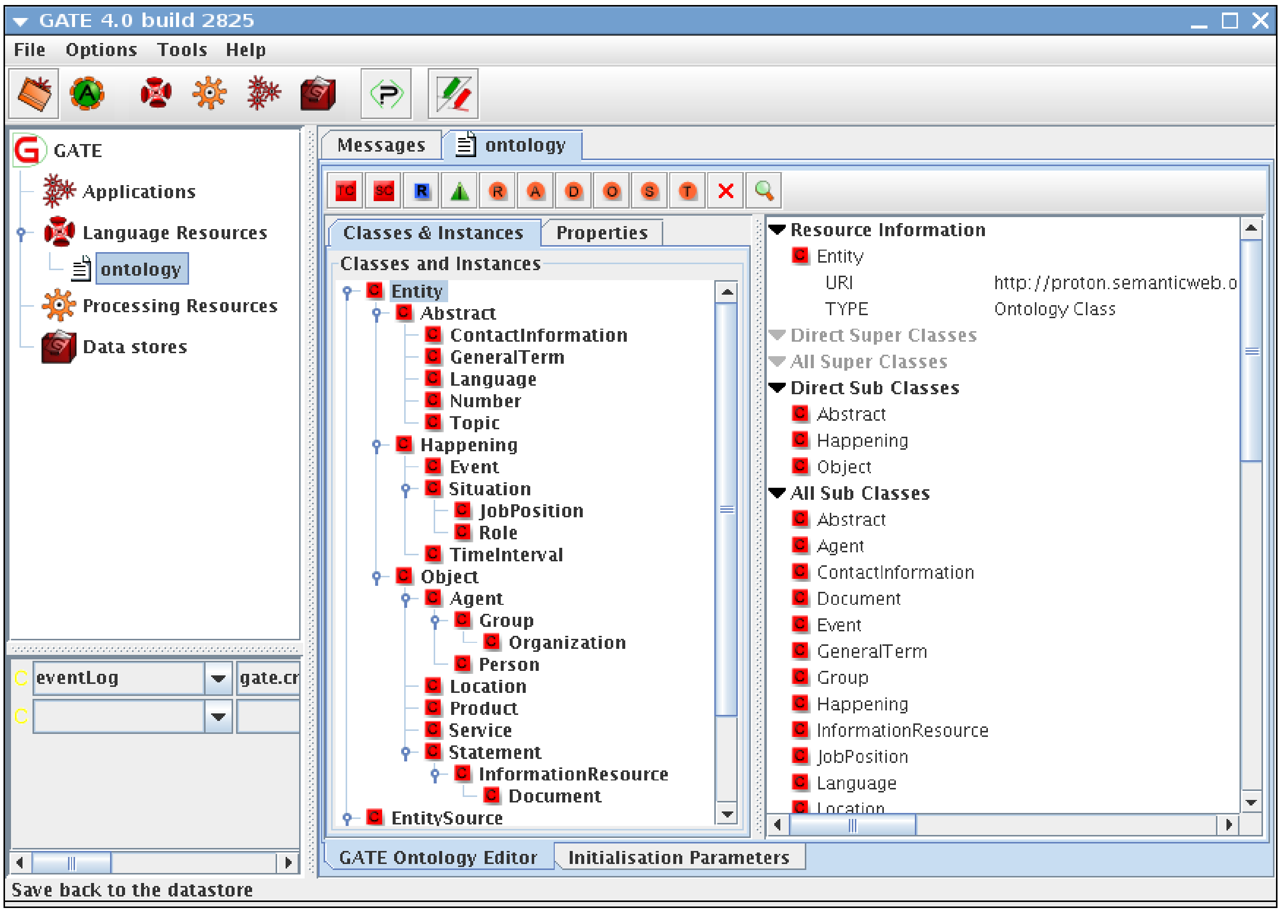


Figure 2.3: GATE Ontology Editor

```
// 2nd argument = URI of the concept
// 3rd argument = URI of the new instance
service.addIndividual("neonRepository",
"http://proton.semanticweb.org/2005/04/protons#Recognized",
"http://proton.semanticweb.org/2005/04/protons#Rec1");

Resulting Event Log:
====================
// Here + indicates the addition
+ <http://proton.semanticweb.org/2005/04/protons#Rec1>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://proton.semanticweb.org/2005/04/protons#Recognized>

// Adding a label (annotation property) to the instance
// with value "Rec Instance"
// 1st argument = name of the repository
// 2nd argument = URI of the instance
// 3rd argument = URI of the RDFS.label
// 4th argument = label
service.addAnnotationPropertyValue("neonRepository",
           "http://proton.semanticweb.org/2005/04/protons#Rec1",
  "http://www.w3.org/2000/01/rdf-schema#label",
```

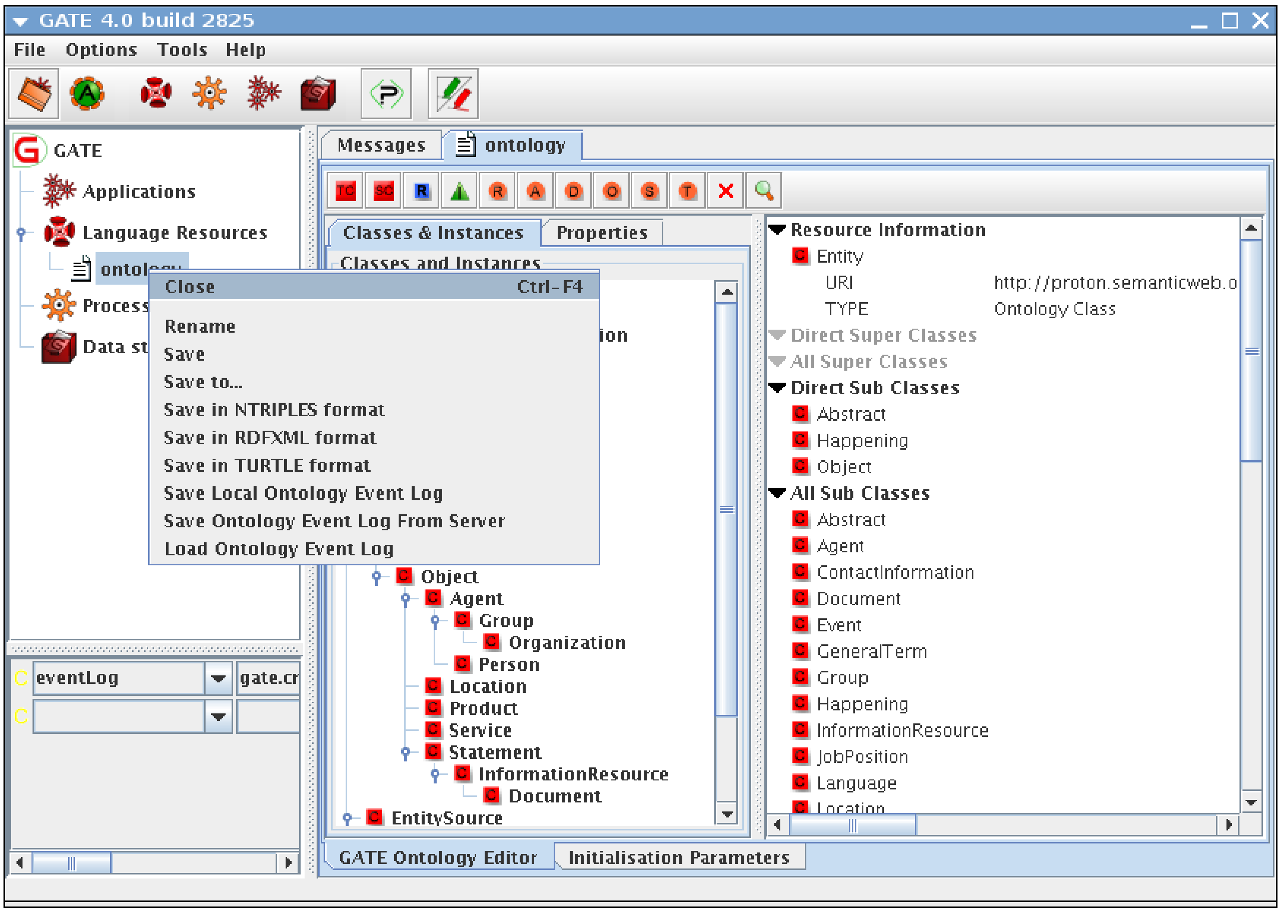


Figure 2.4: Options allowing users to load and export event log

```
  "Rec Instance");

Resulting Event Log:
====================
+ <http://proton.semanticweb.org/2005/04/protons#Rec1>
<http://www.w3.org/2000/01/rdf-schema#label>
<Rec Instance>
<http://www.w3.org/2001/XMLSchema#string>

// Adding a new class called TrustSubClass
// 1st argument = name of the repository
// 2nd argument = URI of the new concept
// 3rd argument = 0 indicates OWL CLASS
service.addClass("neonRepository", "http://proton.semanticweb.org/2005/04/TrustSub

Resulting Event Log:
====================
+ <http://proton.semanticweb.org/2005/04/protons#TrustSubClass>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Class>

// TrustSubClass is a subClassOf the class Trusted
// 1st argument = name of the repository
// 2nd argument = URI of the super class
// 3rd argument = URI of the sub class
service.addSubClass("neonRepository",
  "http://proton.semanticweb.org/2005/04/protons#Trusted",
          "http://proton.semanticweb.org/2005/04/protons#TrustSubClass");

Resulting Event Log:
====================
+ <http://proton.semanticweb.org/2005/04/protons#TrustSubClass>
<http://www.w3.org/2000/01/rdf-schema#subClassOf>
<http://proton.semanticweb.org/2005/04/protons#Trusted>

// Deleting a property called hasAlias and all relevant statements
// 1st argument = name of the repository
// 2nd argument = URI of the property to be deleted
// 3rd argument = indicates whether the sub properties should be deleted
// return an array of URIs that are deleted because of this operation
String[] deletedResources =
service.removePropertyFromOntology("neonRepository",
        "http://proton.semanticweb.org/2005/04/protons#hasAlias",
false);

Resulting Event Log:
====================
// Here - indicates the deletion
// * indicates any value in place
- <http://proton.semanticweb.org/2005/04/protons#hasAlias> <*> <*>
- <*> <http://proton.semanticweb.org/2005/04/protons#hasAlias> <*>
- <*> <*> <http://proton.semanticweb.org/2005/04/protons#hasAlias>

// Deleting a label set on the instance Rec1
// 1st argument = name of the repository
// 2nd argument = URI of the instance
// 3rd argument = URI of the annotation property LABEL
// 4th argument = value of the label
service.removeAnnotationPropertyValue("neonRepository",
  "http://proton.semanticweb.org/2005/04/protons#Rec1",
  "http://www.w3.org/2000/01/rdf-schema#label",
  "Rec Instance");

Resulting Event Log:
====================
- <http://proton.semanticweb.org/2005/04/protons#Rec1>
<http://www.w3.org/2000/01/rdf-schema#label>
<Rec Instance>
<http://www.w3.org/2001/XMLSchema#string>

// Resetting the entire ontology (Deleting all statements)
// 1st argument = name of the repository
service.cleanOntology("neonRepository");
```

```
Resulting Event Log:
====================
- <*> <*> <*>
```

2.2.3 Ontology Event Model

Event logging enables a user to know that the ontology has been modified by another user (possibly at some remote location) and to know how it differs (e.g. concept X is new to the ontology). For a deleted or modified concept or instance, we might want to check that nothing else is adversely affected. For an additional concept in the ontology, we might need to find new instances of this concept from our text (either manually or automatically). In other words, the overall idea here is to log these events as they occur. The GATE Ontology API has its own event model which notifies the registered listeners about the different events related to ontologies (the Ontology interface provides methods to get registered to listen these events). These events are fired when a resource is added, modified or deleted from the ontology. An interface called *OntologyModificationListener* is created with five methods (see below) that need to be implemented by the listeners of ontology events. Whilst the server running GOS is responsible for recording all the changes made by all the users connected to the GOS, sometimes a user may want to know about the changes he/she introduced to the ontology. In order to record user specific changes, each instance of the NeOnOntologyServiceLR client registers itself as a litener of these events. As a result, whenever user makes any change to the ontology, apart from it being logged on the server, it is also stored locally. The GUI also provides an option to export such a local event log to a file.

```
public void resourcesRemoved(Ontology ontology, String[] resources);
```

This method is invoked whenever an ontology resource (a class, property or instance) is removed from the ontology. Deleting one resource can also result into the deletion of other dependent resources (more details on how deletion works are explained later). The second parameter, an array of strings, provides a list of URIs of resources deleted from the ontology.

```
public void resourceAdded(Ontology ontology, OResource resource);
```

This method is invoked whenever a new resource is added to the ontology. The parameters provide references to the ontology and the resource being added to it.

```
public void ontologyRelationChanged(Ontology ontology, OResource resource1,
OResource resource2, int eventType);
```

This method is invoked whenever a relation between two resources (e.g. OClass and OClass, RDFProperty and RDFProperty etc) is changed. Example events are the addition or removal of a subclass or a subproperty, two classes or properties being set as equivalent or different and two instances being set as same or different. The first parameter is the reference to the ontology, the next two parameters are the resources being affected and the final parameters is the event type. Please refer to the list of events specified below for different types of events.

```
public void resourcePropertyValueChanged(Ontology ontology,
OResource resource, RDFProperty property,
Object value, int eventType);
```

This method is invoked whenever any property value is added or removed to a resource. The first parameter provides a reference to the ontology in which the event took place. The second provides a reference to the resource affected, the third parameter provides a reference to the property for which the value is added or removed, the fourth parameter is the actual value being set on the resource and the fifth parameter identifies the type of event.

```
public void ontologyReset(Ontology ontology)
```

This method is called whenever the ontology is reset, i.e. when all the resources in the ontology are deleted.

Below we list specific event types that the ontology event methods (as discussed above) capture and notify to the users.

```
OCLASS_ADDED_EVENT
ANONYMOUS_CLASS_ADDED_EVENT
CARDINALITY_RESTRICTION_ADDED_EVENT
MIN_CARDINALITY_RESTRICTION_ADDED_EVENT
MAX_CARDINALITY_RESTRICTION_ADDED_EVENT
HAS_VALUE_RESTRICTION_ADDED_EVENT
SOME_VALUES_FROM_RESTRICTION_ADDED_EVENT
ALL_VALUES_FROM_RESTRICTION_ADDED_EVENT
SUB_CLASS_ADDED_EVENT
SUB_CLASS_REMOVED_EVENT
EQUIVALENT_CLASS_EVENT
ANNOTATION_PROPERTY_ADDED_EVENT
DATATYPE_PROPERTY_ADDED_EVENT
OBJECT_PROPERTY_ADDED_EVENT
TRANSITIVE_PROPERTY_ADDED_EVENT
SYMMETRIC_PROPERTY_ADDED_EVENT
ANNOTATION_PROPERTY_VALUE_ADDED_EVENT
DATATYPE_PROPERTY_VALUE_ADDED_EVENT
OBJECT_PROPERTY_VALUE_ADDED_EVENT
RDF_PROPERTY_VALUE_ADDED_EVENT
ANNOTATION_PROPERTY_VALUE_REMOVED_EVENT
DATATYPE_PROPERTY_VALUE_REMOVED_EVENT
OBJECT_PROPERTY_VALUE_REMOVED_EVENT
RDF_PROPERTY_VALUE_REMOVED_EVENT
EQUIVALENT_PROPERTY_EVENT
OINSTANCE_ADDED_EVENT
DIFFERENT_INSTANCE_EVENT
SAME_INSTANCE_EVENT
RESOURCE_REMOVED_EVENT
RESTRICTION_ON_PROPERTY_VALUE_CHANGED
SUB_PROPERTY_ADDED_EVENT
SUB_PROPERTY_REMOVED_EVENT
```

An ontology is responsible for firing various ontology events. Objects wishing to listen to the ontology events must implement the methods above and must be registered with the ontology using the following method.

```
addOntologyModificationListener(OntologyModificationListener oml);
```

The following method cancels the registration.

```
removeOntologyModificationListener(OntologyModificationListener oml);
```

2.2.4 What happens when a resource is deleted?

Resources in an ontology are connected with one another, for example, one class can be a sub or superclass of the other. A resource can have multiple properties attached to it. Taking these various relations into account, changes in one resource can affect other resources in the ontology. Below we describe what happens when a resource is deleted.

- **When a class is deleted,** all subclasses and instances of this class are shifted to the superclass of this class. If there is no superclass, the immediate subclasses become top classes and the instances are deleted. If there is any property where this class is assigned as Domain or Range, the property is deleted. All the statements are also removed from the repository where there is a mention of this class. This ensures that the class is completely deleted from the repository.
- **When an instance is deleted,** all the statements are removed from the repository where there is a mention of this instance. This ensures that the instance is completely deleted from the repository.
- **When a property is deleted,** all subproperties of this property are shifted to the superproperty of this property. If there is no super property, the immediate subproperties become top properties. All the statments are also removed from the repository where there is a mention of this property. This ensures that the property is completely deleted from the repository.

Chapter 3

Bottom-up approach to ontology evolution: experiments with fish

In this chapter, we describe our work on SARDINE which generates potential new concepts for the ontology, based on analysis of the text. This work is carried out in conjunction with WP6 and WP7. In WP6, we develop the SAFE plugin for the NeOn toolkit, of which SARDINE is one application within the plugin. In WP7, we develop methods for ontology population and acquisition as part of the Fisheries case study.

SARDINE aims to find new mentions of fish[1] from a corpus, and adds them to the ontology as new instances or concepts (or makes suggestions about where to add them). It operates as a GATE Annotation Service (GaS), taking as input the text to be processed and producing as output an OWL file. This will be discussed more fully in Section 3.4.

3.1 SARDINE

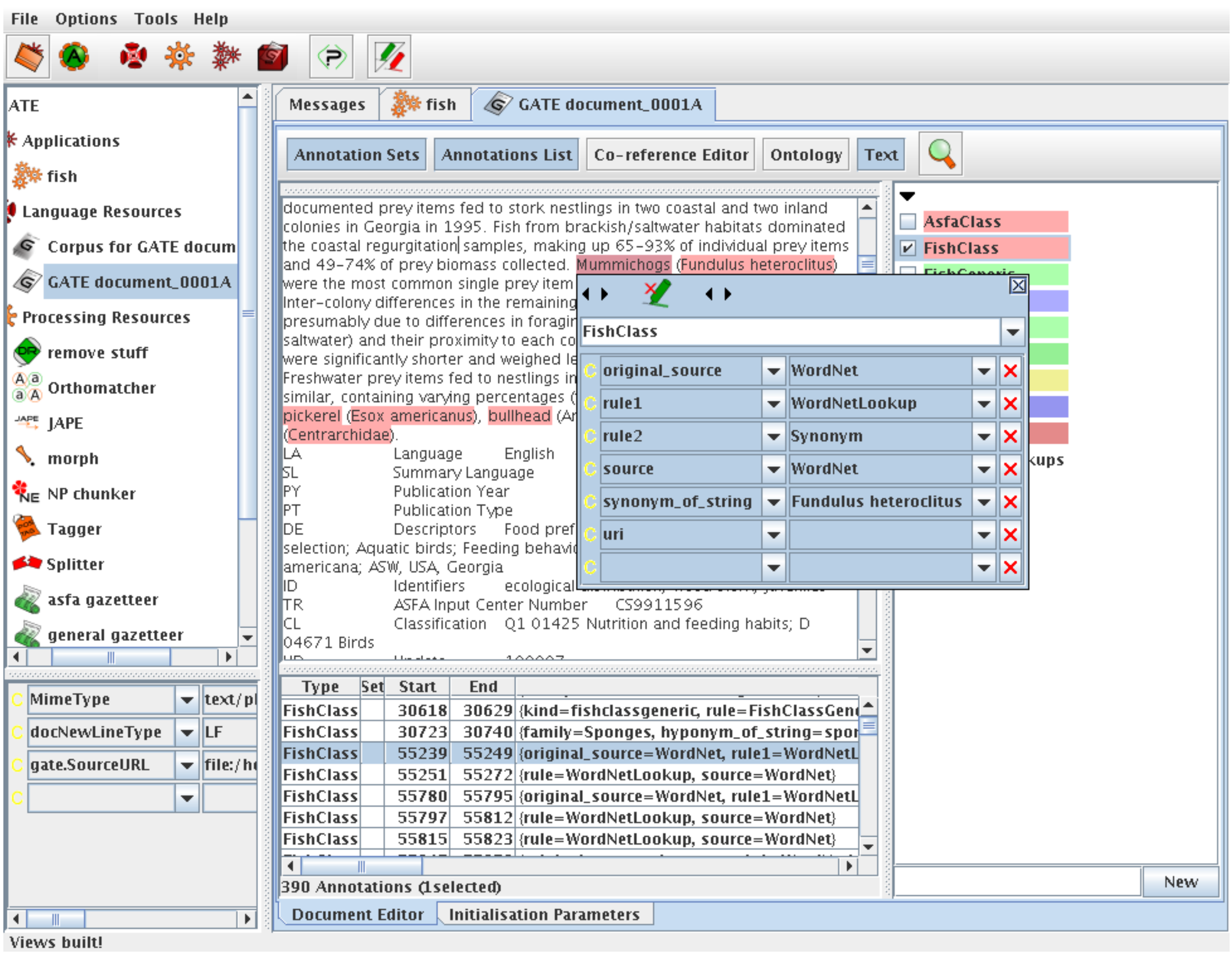
The idea behind SARDINE is to identify new mentions of fish species from text. This includes both finding known fish names that are listed in the ontology, and also identifying potential new fish names not listed in the ontology, and their relationship with existing known fish (and potentially, other entities within the fisheries domain). The GATE application used by SARDINE finds instances of fish in the text and annotates them with information from the ontology (in this case, we use Figis[2]). We also find instances of fish using other lexical resources such as WordNet[ME90] and Agrovoc[3]. For the new fish names found, SARDINE attempts to classify them in the ontology, based on linguistic information such as synonyms and hyponyms of existing fish, as explained in the following section.

3.2 Pattern matching rules for ontology population and acquisition

The GATE application for finding fish names uses ontology-based information extraction techniques based on pattern matching (see for example [MBC03, MLP07, CV05]). We first recognise known fish names in the text, in order to help us find new kinds of fish. The application essentially uses linguistic and contextual information to find (guess) relations between fish, such as synonyms and hyponyms. This enables us to add links between existing instances, and to suggest new instances to be included in the ontology, based on their relation with existing instances.

For example, "rainbow trout" is an example of a known fish that is listed in Figis. When we find an example of it in the text, we extract the following information:

[1]When we talk about fish in this document, we refer to all kinds of marine life.
[2]figis: http://www.fao.org/aims/aos/fi/species_v1.0.owl
[3]See http://www.fao.org/aims/ag_intro.htm for more information



```
documented prey items fed to stork nestlings in two coastal and two inland
colonies in Georgia in 1995. Fish from brackish/saltwater habitats dominated
the coastal regurgitation samples, making up 65-93% of individual prey items
and 49-74% of prey biomass collected. Mummichogs (Fundulus heteroclitus)
were the most common single prey item
Inter-colony differences in the remaining
presumably due to differences in foragi
saltwater) and their proximity to each co
were significantly shorter and weighed le
Freshwater prey items fed to nestlings in
similar, containing varying percentages (
pickerel (Esox americanus), bullhead (A
(Centrarchidae).
LA          Language        English
SL          Summary Language
PY          Publication Year
PT          Publication Type
DE          Descriptors     Food pref
selection; Aquatic birds; Feeding behavi
americana; ASW, USA, Georgia
ID          Identifiers     ecological
TR          ASFA Input Center Number     CS9911596
CL          Classification  Q1 01425 Nutrition and feeding habits; D
04671 Birds
```

FishClass

original_source	WordNet
rule1	WordNetLookup
rule2	Synonym
source	WordNet
synonym_of_string	Fundulus heteroclitus
uri	

Type	Set	Start	End	
FishClass		30618	30629	{kind=fishclassgeneric, rule=FishClassGen
FishClass		30723	30740	{family=Sponges, hyponym_of_string=spo
FishClass		55239	55249	{original_source=WordNet, rule1=WordNetL
FishClass		55251	55272	{rule=WordNetLookup, source=WordNet}
FishClass		55780	55795	{original_source=WordNet, rule1=WordNetL
FishClass		55797	55812	{rule=WordNetLookup, source=WordNet}
FishClass		55815	55823	{rule=WordNetLookup, source=WordNet}

390 Annotations (1selected)

MimeType	text/pl
docNewLineType	LF
gate.SourceURL	file:/h

Figure 3.1: Annotation of a synonym in GATE

```
name = rainbow trout
family = Salmonids_nei
source = Figis
uri = http://www.fao.org/aims/aos/fi/species_v1.0.owl#31005_2934
```

We then use a set of heuristics to find synonyms and hyponyms of known fish, described below.

3.2.1 Finding hyponyms

The application extracts candidate subclasses of existing elements from the species ontology, using the following heuristics:

- Noun phrases extending existing identified labels from the species ontology, e.g. "Japanese flounder", "Suberites sponges" (where "flounder" and "sponge" are existing labels);
- Hearst patterns [Hea92]: "especially", "such as", "including", "and other", "or other". Basically here we look for patterns of the type "X such as Y", where Y is frequently found to be a hyponym of X.

The application then annotates the new (proposed) instance of a fish with the following information:

```
name = Suberites Sponges
```

```
is_hyponym = true
original_source = Figis
hyponym_of_string = sponges
uri = http://www.fao.org/aims/aos/fi/species_v1.0.owl#31005_2951
rule = NounFishClass
```

This tells us the string annotated, the fact that it is a hyponym, the ontology which lists the hypernym (the previously known fish), the name of the hypernym (in this case "sponges", the URI of the hypernym, and the rule used by our application to find the new fish.

3.2.2 Finding synonyms

The application extracts candidate synonyms of existing elements from the species ontology (i.e. new instances of the same class), using the following heuristics:

- apposition: here we find for example bracketed mentions of recognized fish names, e.g. "Mummichogs (Fundulus Heteroclitus) were the most common single prey item" where "Mummichogs" and "Fundulus Heteroclitus" are synonyms. Usually these are instances of the English name and the Latin name for the same fish.

- list membership: here we find new instances among lists of recognised instances of fish, e.g. "Tuna, clams and herring", where we know that tuna and herring are both fish, but know nothing about clams: we can predict that clams are also fish.

The application then annotates the new (proposed) instance of a fish with the following information:

```
name = Mummichogs
original_source = WordNet
synonym_of_string = Fundulus heteroclitus
uri = http://www.fao.org/aims/aos/fi/species_v1.0.owl#31005_2960
rule = WordNetLookup
```

Figure 3.1 depicts a screenshot of the annotation of a synonym (mummichogs) in GATE.

3.3 Analysis of Results

We performed some experiments using the FIGIS factsheets described earlier, to investigate how many new concepts had been found. We found no occurrences of synonyms in these documents, although we did find many occurrences in other kinds of documents about fish. Future work will focus on expanding the patterns in order to rectify this. However, we found 2160 instances of potential new hyponyms. Note that we refer to instances because some terms occurred more than once, so the number of distinct new hyponyms is less. Table 3.1 shows some statistical information about them. Additionally, we found 574 instances of potential new hyponyms matching non-classified concepts. These were new concepts proposed by the system that were not based on any relation with existing concepts (for example, a phrase containing an adjective modifying the word "fish"). However, many of these would probably not be included as new concepts as they were spurious. Table 3.2 shows some examples of hyponyms found, together with their hypernym. On the left we see hyponyms that would be accepted as new concepts; on the right we see examples of spurious hyponyms that would not be accepted. Many of these spurious examples could be eliminated by using a better stoplist (for example, preventing certain adjectives occurring before a known fish name to be included in the pattern).

Number of documents	1384
Total new hyponyms found	2160
Total new hyponyms of existing concepts	1586
New hyponyms of concepts in Figis	150
New hyponyms of concepts in WordNet	783
New hyponyms of concepts in Agrovoc	653

Table 3.1: Statistical information about new concepts proposed

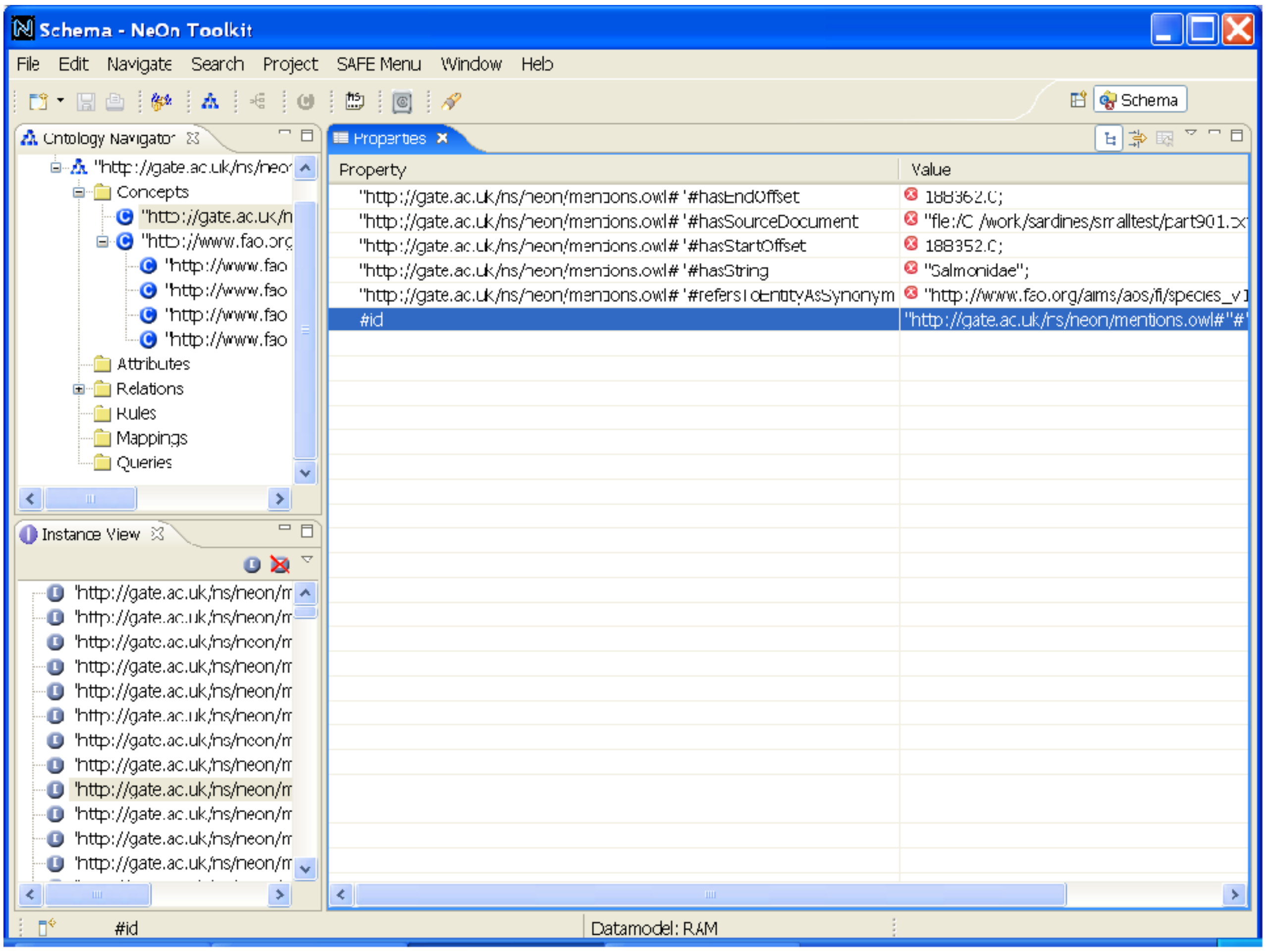
New concept	Hypernym	Spurious concept	hypernym
Gulf sturgeon	sturgeon	dorsal rays	ray
Tropical tunas	tuna	traditional tuna	tuna
scomber vernalis	scomber	Suborder Clupeidae	clupeidae
golden carp	carp	Cape Cod	cod
horse mackerel	mackerel	facilitated blue mussel	mussel

Table 3.2: Examples of new hyponyms found

3.4 Implementation of SARDINE

As mentioned earlier, SARDINE is a GATE Annotation Service (GaS). It takes as input the text to be processed: in this case we use corpora from the fisheries domain, such as factsheets about fish from the FAO. The output is a set of OWL files containing mentions and any newly created candidate entities. One OWL file is produced for each document. The reason for producing separate OWL files is that it is rather unwieldy to produce one huge OWL file for the whole corpus, and it is easier to manipulate this way, because each generated file can be integrated (or not) into a larger ontology. The newly generated concepts are only suggestions, and should therefore be verified by a domain expert before integration takes place.

With the SAFE plugin loaded into the NeOn toolkit, the user simply clicks on the SAFE button, and will be prompted for a directory containing the texts to process, and a directory for the output (this would normally be a new blank directory). SARDINE calls the GaS once for each document to be processed. The generated OWL file is placed in a corresponding file in the output directory. The new ontology can be viewed in the NeOn Toolkit in the usual way. Figure 3.2 depicts one such ontology in the toolkit.



Property	Value
"http://gate.ac.uk/ns/neon/mentions.owl#"#hasEndOffset	188362.0;
"http://gate.ac.uk/ns/neon/mentions.owl#"#hasSourceDocument	"file:/C:/work/sardines/smalltest/part901.tx
"http://gate.ac.uk/ns/neon/mentions.owl#"#hasStartOffset	188352.0;
"http://gate.ac.uk/ns/neon/mentions.owl#"#hasString	"Salmonidae";
"http://gate.ac.uk/ns/neon/mentions.owl#"#refersToEntityAsSynonym	"http://www.fao.org/aims/aos/fi/species_v1
#id	"http://gate.ac.uk/ns/neon/mentions.owl#"#"

Figure 3.2: New ontology generated by SARDINE

Chapter 4

Experiments with folksonomies

4.1 Introduction

This section describes the methodology and the results of our investigation of folksonomy-based ontology evolution. Folksonomies are popular Web2.0 ecosystems enjoying high interest from web users. Their basic entities are resources, users and tags. Users upload and annotate resources with their personally selected tags. Due to the usability and strong social dimension of folksonomies, they are updated very frequently, i.e. users upload and annotate new resources with new tags daily or even several times a day. Unlike folksonomies, ontologies evolve much more slowly and under the influence of a much more restricted community (i.e. ontology engineers). As a result, they often lag behind folksonomies in terms of content. This constitutes a motivation for our work: we wish to leverage the dynamic nature of folksonomies in order to evolve the content of existing ontologies.

The aim of this study is to approach the ontology evolution problem in a bottom-up manner, initiating the process at a low level with user contributed content from folksonomies. The study of folksonomy-based ontology evolution and the experimentation with real data sets aim at identifying alternative or complementary ways to perform ontology evolution.

More specifically, the exploitation of statistical information from folksonomies can provide helpful insights into the possible knowledge gaps of ontologies. For example, the high co-occurrence rate between two terms (tags) in folksonomies indicates with a high probability that they are related [SM07, BKS06], even though they may not belong to the same ontology or even if they don't both belong to any ontology. This means that it may be necessary to introduce a new relation between them in the ontology (in case they belong to the same ontology) or to add one of them as a new candidate entity (class, property or individual) for the ontology.

The main objective of this work is to experiment with FAO resources and real datasets from folksonomies, and to draw some conclusions about the entire procedure of bottom-up folksonomy-based ontology evolution.

4.2 Methodology

As previously mentioned, our goal is to exploit the rapid evolution of folksonomy content in order to perform ontology evolution. The core idea of our approach is to use folksonomies for identifying tags that often co-occur with the entities of a given ontology. These related tags can potentially indicate new - candidate entities that could be added to the ontology. Although the nature of the two sources (folksonomies and ontologies) is diverse and their alignment is a complex procedure, the basic steps of the process are depicted in Figure 4.1.

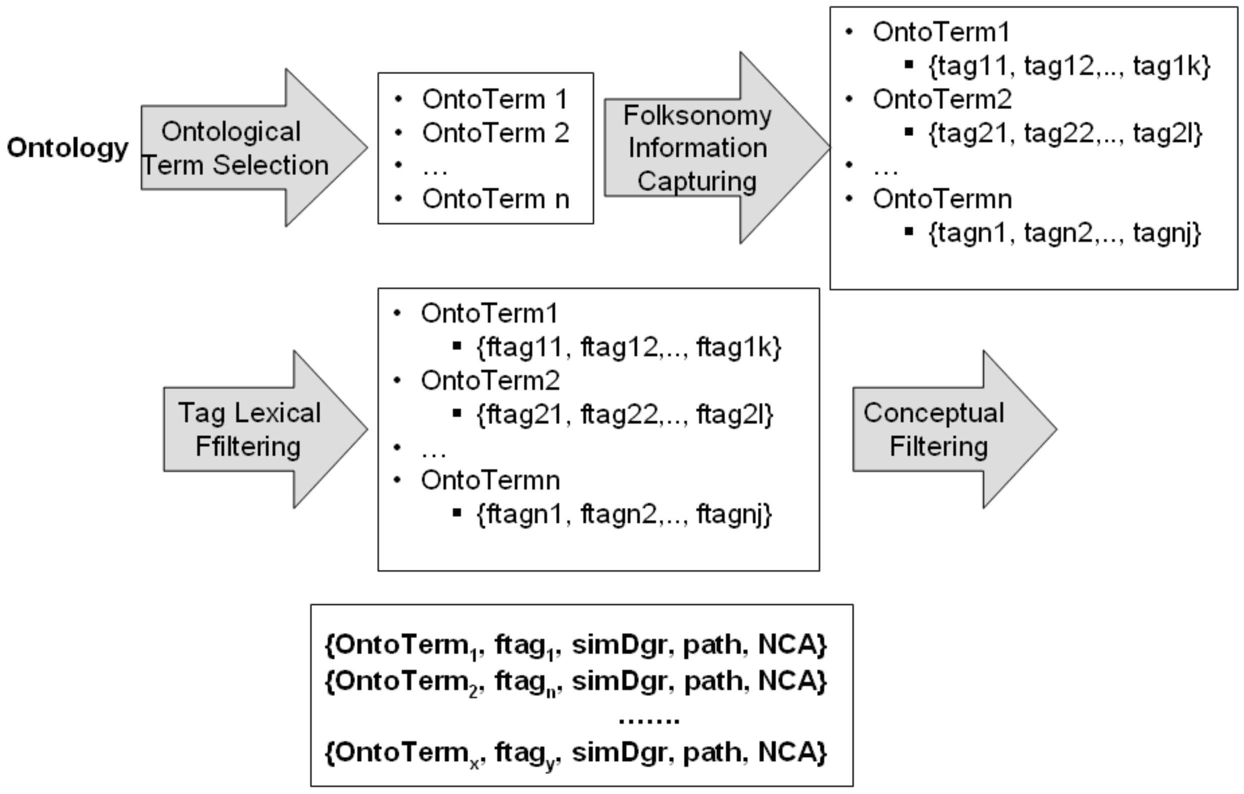

Ontology
Ontological Term Selection
OntoTerm 1
OntoTerm 2
…
OntoTerm n
Folksonomy Information Capturing
OntoTerm1
{tag11, tag12,.., tag1k}
OntoTerm2
{tag21, tag22,.., tag2l}
…
OntoTermn
{tagn1, tagn2,.., tagnj}
Tag Lexical Ffiltering
OntoTerm1
{ftag11, ftag12,.., ftag1k}
OntoTerm2
{ftag21, ftag22,.., ftag2l}
…
OntoTermn
{ftagn1, ftagn2,.., ftagnj}
Conceptual Filtering
{OntoTerm1, ftag1, simDgr, path, NCA}
{OntoTerm2, ftagn, simDgr, path, NCA}
…….
{OntoTermx, ftagy, simDgr, path, NCA}

Figure 4.1: Methodology for folksonomy-based ontology evolution

4.2.1 Ontological Entity selection

Although the ontology evolution is meant to be bottom-up, i.e. initiated from a low level, the involvement of ontological entity names at this level is the domain/case-defining step. Thus the selection of entities, classes, properties and individuals depending on the ontology and the required level of evolution is performed first. The local names or labels of the entities are further used to query folksonomies for related tags as described below.

4.2.2 Capturing folksonomy information

As folksonomies are diverse and data intensive, the information they contain covers a wide range of topics. Thus the first sub-goal of this work is to extract from them only the necessary information. As previously mentioned, the goal is to identify related tags for the ontological entities. Related studies [SM07, BKS06] have shown that the co-occurrence of tags in various resources indicates relatedness amongst them. Various statistical algorithms have thus been proposed for identifying clusters of related tags based on their co-occurrence. We rely on such algorithms to identify tags that are potentially related to the ontological terms acquired in Step 1. The clustering of folksonomy tags can be repeated at regular time intervals in order to reveal new related tags as the folksonomy evolves.

4.2.3 Lexical filtering of tags

One of the main characteristics of folksonomies is the unrestricted way in which users can tag their resources. This increases the user contribution but also increases the noise in the system. Although all the tags tend to be important to the users, not all of them are important in the ontology evolution procedure. Therefore, it is necessary to perform lexical filtering operations in order to exclude the tags that introduce noise in the process, thereby increasing the performance. More specifically, in this experiment we remove non-English tags and all non-alphabetical tags (e.g. those containing special characters or numbers). We also normalise the tags by replace all plurals with their singular equivalents. The output of this step is a set of related tags for each concept, lexically filtered.

4.2.4 Conceptual Filtering

Once the related tags have been lexically processed, the last step of our approach is to perform a conceptual filtering before suggesting candidate entities to be added into the original ontology. The reason for that is to increase the accuracy of suggestions. As described in more detail in the following case study, the related tags returned through the clustering algorithms from folksonomies are frequently too general (and usable in several different domains, e.g., Object) and therefore do not necessarily contribute any significant knowledge to the domain of the original ontology.

In order to avoid the suggestion of many candidate entities with low relevance, we perform conceptual filtering on the related tag sets exploiting the IS-A hierarchy of WordNet[ME90]. We use the similarity formula of Wu and Palmer [WP94] to calculate the conceptual distance between two WordNet terms. This is a function of the path length between the two terms, and number of common ancestors in the conceptual hierarchy of WordNet. Using this approach, we are able to obtain not only a similarity degree between two terms, but also the connecting path between them in WordNet.

Applying this strategy pairwise on each ontological concept and each of its related tags, we are able to identify the following cases:

- The related tag is not contained in the ontology. In this case, provided that the similarity is higher than a given threshold, we can suggest (1) that the tag is added as a new entity in the ontology and (2) also the way it should be connected to the rest of the ontology (i.e., the entity to which it relates and the type of relation).

- The related tag is contained in the ontology already. In this case, we propose the connecting path given by WordNet to be a candidate relation between the two entities.

These two cases can cover the following possible changes in ontologies according to the specifications made in D 1.5.1 [MPD$^+$07]:

```
add superclass C
add subclass C
add equivalent class C
```

In the following case study, we describe the method in more detail and give some examples.

4.3 Case Study: Evolving the FAO biological entity ontologies using Flickr

Flickr[1], the folksonomy we used to acquire real data sets, is one of the fastest growing and most popular folksonomies among web users. An additional reason for selecting Flickr is the wide range of domains covered by its photos. This can provide more confidence on the diverse and rich vocabulary with which the

[1] http://www.flickr.com

users annotate their photos. The ontology to be evolved was the biological entity ontology of FAO which contains 10604 classes describing biological species[2].

4.3.1 Step 1

First, we selected the 8108 entities - classes with a non null "hasNameEN" data property, out of the 10604 total classes[3].

4.3.2 Step 2

To obtain the related folksonomy tags for each entity, we used the "related tags" function of the Flickr API[4]. This function returns the related tags of a tag in Flickr using statistical measures based on the co-occurrence of tags. Looking for related tags in Flickr, we found that only 1216 out of the 8108 classes (15%) were used by Flickr users to annotate their photos. Furthermore, only 123 out of the 1216 (10%), which equates to 2% of the initial dataset of 8108, were found to return related tags in Flickr. This is possibly due to the following reasons:

- Many of these tags were found to describe one or two photos in Flickr, thus there is the possibility that these photos are only tagged with these entities. As a result, there are no co-occurring tags for them in the system.
- The Flickr API characterises as related those tags that co-occur at least a certain number of times, thus the entities that fall below this threshold are excluded.

The reduced data set consists of 123 ontological entities from the biological entity ontology, each with a set of related tags of variable cardinality between 3 and 74.

4.3.3 Step 3

As discussed above, the tags retrieved from folksonomies require a certain amount of lexical processing prior to looking for them in WordNet. The reason for this is to minimise the processing time. We remove the redundant tags which are the non English tags, tags that appear in both singular and plural forms or tags that contain special characters (such as "?", "*") or numbers.

4.3.4 Step 4

Using the strategy described above for each of the ontological entities, we looked for the WordNet-based similarity between 2305 pairs of terms (ontological_entity, related_tag). Out of these pairs, 600 were found to be related in WordNet with similarities varying from 0.2 to 1.0 (where a similarity of 1 indicates a synonym). Experimenting with different similarity thresholds, we discovered that the highest number of relevant relations is achieved with similarity higher or equal to 0.75. For this threshold, we acquired 159 relations which correspond to 159 candidate changes for the ontology. It is important to point out that we only worked with ontological classes as opposed to properties and individuals as the conceptual filtering methodology is only able to support subsumption relations. Current work is focusing on alternative conceptual filtering methods that can support all types of ontological entities such as properties and individuals.

[2]http://www.fao.org/aims/aos/fi/species_v1.0.owl
[3]See [Car07] for more information
[4]http://www.flickr.com/services/api

Ontological Entity	Related Tag	Path	SimDgr	NCA
Albacore	salmon	albacore, long-fin_tunny → tuna → **food_fish** ← salmon	0.87	**food_fish**
Albacore	eel	albacore, long-fin_tunny → tuna → scombroid → percoid_fish → spiny-finned_fish → **teleost_fish** ← soft-finned_fish ← eel	0.76	**teleost_fish**
Bat	fish	batfish → spiny-finned_fish → teleost_fish → bony_fish → **fish**	0.82	**fish**
Striped bass	rockfish	striped_bass, striper, Roccus_saxatilis, rockfish	1.00	synonyms
Swordfish	tuna	swordfish → **scombroid** ← tuna	0.93	**scombroid**
Humpback whale	cetacean	humpback_whale → baleen_whale → whale → **cetacean**	0.88	**cetacean**
Scallop	crab	scallop → **shellfish** ← crab	0.86	**shellfish**

Table 4.1: Candidate concepts and relations

4.3.5 Results

Below we demonstrate some representative results from the above case study. Table 4.1 contains examples of how certain ontological entities were extended with tags retrieved from Flickr. For example, the first row indicates that, according to Flickr usage, the term "salmon" often co-occurs with (and therefore might be related to) the term "Albacore" (corresponding to an ontology entity with such a label). In WordNet these two terms are siblings for the concept of food_fish, which is their nearest common ancestor. This relation translates in the corresponding path and also in a high degree of similarity when computed with Wu and Palmer's formula.

We can see that there are various types of connections between ontological entities and related tags. "Striped bass" is related to rockfish with the relation *sameAs*, while "Humpback whale" is a *subClassOf* "cetacean", which does not belong to the ontology. Also "tuna" and "swordfish" are both subclasses of "scombroid" which exists in the ontology, while "Albacore" and "eel" both belong to the ontology but connect through "teleost_fish" which does not belong to the ontology. These new entities and their corresponding relations to existing ontology entities are suggested as additions to the FAO ontology, thus leading to its evolution. Out of the 159 candidate entities, 83 (52%) were valid relation suggestions and 21 of these suggested that the corresponding tag should be added as a new class (*superclass* or *subclass*).

Table 4.2 indicates another interesting phenomenon, which is the 48% of the relations with high similarity but which are invalid recommendations for the ontology. This happens because the ontological terms are polysemous, and probably their non-fisheries related meanings are more popular in folksonomies. As a result, an important outcome of this study is the necessity for disambiguation of the terms involved, either from ontological sources or from folksonomies in order to avoid this phenomenon.

It is also interesting to analyse the results that returned low similarity. These are either generic concepts in the domain of fisheries (e.g. animal) or concepts which do not belong to the domain of fisheries (e.g. asparagus). It should be noted that the related tags were obtained by user-contributed annotations, thus

Ontological Entity	**Related Tag**	**SimDgr**	**NCA**
Banjo	drum	0.77	**musical_instrument**
Torpedo	tube	0.75	**device**
Thresher	machine	0.87	**machine**

Table 4.2: Irrelevant results due to ontological term ambiguity

Ontological Entity	**Related Tag**	**SimDgr**	**NCA**
Scallop	asparagus	0.57	**food**
Ling	buddhist	0.47	**organism**
Roach	graffiti	0.40	**object**

Table 4.3: Low similarity relations

total conceptual correctness cannot be guaranteed. Table 4.3 shows some results for relations with low similarity. We can see here that users in folksonomies have annotated images of "Scallop" with the term "asparagus", although the conceptual relation between these terms is quite low. Also the ambiguous tags Ling and Roach connect with non-fisheries related tags through very generic and high level concepts, thus their similarity is low.

4.4 Summary

The experiments we have performed give us valuable insights into the bottom-up approach to ontology evolution and also show where the limitations lie. Our next steps will therefore be to explore ways to overcome these limitations and resolve the issues that hinder our methodology. These are: the disambiguation of ontological entities (e.g. Banjo) prior to looking for related tags in folksonomies, the exclusion of very generic tags as related entities depending on the domain of interest (e.g. animal) and the discovery of alternative ways to retrieve related tags for ontological entities (2% of the 8108 ontological concepts were found to have related tags in folksonomies). Also we are currently working on identifying a methodology to support conceptual filtering for all types of ontological entities (classes, properties, individuals) using more sources of background knowledge such as ontologies already available on the web using SCARLET [5] and WATSON [6] or Wikipedia http://www.wikipedia.org. Finally, the ongoing work includes building appropriate I/O interfaces for our tool so that it can accept OWL ontologies as input and return the enriched ontology as OWL.

[5]http://scarlet.open.ac.uk/
[6]http://watson.kmi.open.ac.uk/

Chapter 5

Conclusions

In this deliverable, we have described the implementation of the methodologies proposed in D1.5.1 for dealing with ontology and metadata change. The top-down approach enables changes made to the ontology to be propagated to the metadata so that as little information as possible is lost. For example, when a concept is deleted from the ontology, usually any metadata (instances) belonging to that concept would be deleted with it. However, this is not always desirable behaviour because often we would prefer to reclassify the instance at a more general level. Klein and Noy [KN03] proposed a framework which captured a set of ontology changes, which we used as a starting point in order to guide us in the creation of a more appropriate methodology for our needs (as described in [MPD$^+$07]), and base the implementation of our ontology change typology on. D1.5.1 described how we used Klein and Noy's framework in order to develop our own set of ontology changes relevant to our needs; in the first part of this deliverable we have reported on the implementation of these changes. The resulting implementation in SAFE also enables a user to be aware of changes to the ontology made by another person at a distributed location, thereby aiding collaborative annotation.

The work described in the first part of this document is strongly related with the work carried out in T1.3 and described in [PHWD08]. This deliverable is concerned with a framework for networked ontology change management. Currently changes made to an ontology in the NeOn toolkit cannot be propagated to the GATE ontology management system and vice versa: this is one of the tasks planned for future work. The idea is to link the change log created by GATE to the change log in the NeOn toolkit, such that changes made to an ontology in GATE can be propagated back to the toolkit and vice versa, in order to aid collaborative distributed ontology management.

The two applications, which implement the bottom-up approach both work on the basis of finding new entities and relations in texts and proposing new concepts for an existing ontology. The first application, SARDINE, uses linguistic techniques to find relations between known and new entities, and outputs the result as a new ontology section. The second application uses techniques based on tag relatedness in folksonomies. The current work is focusing on implementing appropriate interfaces to accept OWL ontologies as input and return the enriched ontology as OWL. Both approaches provide valuable insights into bottom up ontology evolution.

Bibliography

[BKS06] Grigory Begelman, Philipp Keller, and Frank Smadja. Automated tag clustering: Improving search and exploration in the tag space. In *Proceedings of the Collaborative Web Tagging Workshop at the WWW 2006*, pages 22–26, Edinburgh, Scotland, May 2006.

[BTMC04] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3/4):349—373, 2004.

[Car07] Caterina Caracciolo. Revised and enhanced fisheries ontologies. Technical Report D7.2.2, NeOn Project Deliverable, 2007.

[CMBT02] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

[CV05] P. Cimiano and J. Voelker. Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, Alicante, Spain, 2005.

[Hea92] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Conference on Computational Linguistics (COLING'92)*, Nantes, France, 1992. Association for Computational Linguistics.

[Kir06] Atanas Kiryakov. OWLIM: balancing between scalable repository and light-weight reasoner. In *Proc. of WWW2006*, Edinburgh, Scotland, 2006.

[KN03] M. Klein and N. F. Noy. A Component-Based Framework for Ontology Evolution. In *Workshop on Ontologies and Distributed Systems, IJCAI*, 2003.

[MBC03] D. Maynard, K. Bontcheva, and H. Cunningham. Towards a semantic extraction of named entities. In *Proceedings of RANLP-03*, pages 255–261, 2003. `http://gate.ac.uk/sale/ranlp03/ranlp03.pdf`.

[ME90] G. A. Miller (Ed.). WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–312, 1990.

[MLP07] Diana Maynard, Yaoyong Li, and Wim Peters. Nlp techniques for term extraction and ontology population. In P. Buitelaar and P. Cimiano, editors, *Bridging the Gap between Text and Knowledge - Selected Contributions to Ontology Learning and Population from Text*. IOS Press, 2007.

[MPD$^+$07] D. Maynard, W. Peters, M. D'Aquin, M. Sabou, and N. Aswani. Dynamics of metadata. Technical Report D1.5.1, NeOn Project Deliverable, 2007.

[MTC$^+$02] D. Maynard, V. Tablan, H. Cunningham, C. Ursu, H. Saggion, K. Bontcheva, and Y. Wilks. Architectural Elements of Language Engineering Robustness. *Journal of Natural Language Engineering – Special Issue on Robust Methods in Analysis of Natural Language Data*, 8(2/3):257–274, 2002.

[PHWD08] R. Palma, P. Haase, Y. Wang, and M. D'Aquin. Propagation models and strategies. Technical Report D1.3.1, NeOn Project Deliverable, 2008.

[SM07] Lucia Specia and Enrico Motta. Integrating folksonomies with the semantic web. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *Proceedings of the European Semantic Web Conference (ESWC2007)*, volume 4519 of *LNCS*, pages 624–639, Berlin Heidelberg, Germany, July 2007. Springer-Verlag.

[WP94] Zhibiao Wu and Martha Palmer. Verb semantics and lexical selection. In *32nd. Annual Meeting of the Association for Computational Linguistics*, pages 133–138, Morristown, NJ, USA, 1994.