D2.3.8v2 Report and Prototype of Dynamics in the Ontology Lifecycle

Coordinator: Vít Nováček[1]

Loredana Laera[2], Siegfried Handschuh[1], Jan Zemánek[3], Max Völkel[4]

[1]DERI, National University of Ireland, Galway; [2]University of Liverpool, U.K.; [3]University of Economics, Prague, Czech Republic; [4]University of Karlsruhe, Germany

Abstract.
Deliverable D2.3.8v2 (WP2.3) presents a novel ontology integration technique that explicitly takes the dynamics and data-intensiveness of many practical application domains into account. This technique fully implements a crucial part of the dynamic ontology lifecycle scenario defined in D2.3.8v1. Changing and growing knowledge, possibly contained in unstructured natural language resources, is handled by application of cutting-edge Semantic Web technologies. We employ results recently achieved in the WP2.3, following the introduction of dynamic ontology lifecycle scenario in the previous version of this report. In particular, we describe semi-automatic integration of ontology learning results into a manually developed ontology. This integration bases on automatic negotiation of agreed alignments, inconsistency resolution, ontology diff computation and natural language generation methods. Their novel combination alleviates the end-user effort in the dynamic incorporation of new knowledge to large extent, thus conforming to the principles specified in D2.3.8v1. As such, it allows for a basic application of all the dynamic ontology lifecycle features we have proposed.

Keyword list: ontology, ontology dynamics, ontology lifecycle, ontology integration, inconsistency resolution, change operator implementation

Document Identifier	KWEB/2007/D2.3.8v2
Project	KWEB EU-IST-2004-507482
Version	v0.4
Date	November 16, 2007
State	draft
Distribution	public

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) - Coordinator
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polytechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

University of Aberdeen (UNIABDN)
Kings College
AB24 3FX Aberdeen
United Kingdom
Contact person: Jeff Pan
E-mail address: jpan@csd.abdn.ac.uk

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 writing parts of this document:

National University of Ireland Galway
University of Karlsruhe
University of Liverpool
University of Economics, Prague

Changes

Version	Date	Author	Changes
0.1	13.11.07	Vít Nováček	draft created
0.2	14.11.07	Vít Nováček	content related to DINO integration incorporated, DINO user manual drafted
0.3	15.11.07	Vít Nováček	Section 3.2 drafted, general editing
0.4	16.11.07	Vít Nováček	Chapter 7 drafted, Appendix A added, Executive Summary written
1.0	??.12.07	Vít Nováček	final editing, incorporation of the ??? University QC

Executive Summary

We present a report on implementation of the substantial part of dynamic ontology lifecycle scenario we have introduced in the previous version of this document [NHL$^+$06]. By this substantial part we mean semi-automatic dynamic ontology integration method. We show that the algorithms, applications and experimental results we describe within this report allow for practical following of the whole lifecycle scenario presented in [NHL$^+$06].

The work presented here was motivated by certain practical requirements:

1. the ability to process new knowledge (resources) automatically whenever it appears and when it is inappropriate for human users to incorporate it

2. the ability to automatically compare the new knowledge with a "master" ontology (that is manually maintained) and select the new knowledge accordingly

3. the ability to resolve possible major inconsistencies between the new and current knowledge, possibly favouring the assertions from presumably more complex and precise master ontology against the learned ones

4. the ability to automatically sort the new knowledge according to user-defined preferences and present it to them in a very simple and accessible way, thus further alleviating human effort in the task of knowledge integration

The technical core of the deliverable consists of description of the proposed semi-automatic ontology integration principles, algorithms and implementation. We provide basic user manuals for the GUI user interface and for programmatic API to the integration library (implemented in the Java programming language).

In order to show industrial relevance of our approach, we analyse several practical use cases from the e-health and biomedicine domains. We discuss the applicability of the implemented integration technique based on an experiment with respective real-world data-sets. We also show, how the presented ontology integration technique relates to the theoretical studies we provided in another deliverable [NHA$^+$07]. The report is concluded with explicit guidelines on how to practically apply the dynamic lifecycle scenario introduced in [NHL$^+$06], using the novel ontology integration research prototype presented here.

Contents

Chapter 1

Introduction

Ontologies on the Semantic Web, and especially in case of real world applications, are very likely subject to change given the dynamic nature of domain knowledge. Knowledge changes and evolves over time as experience accumulates – it is revised and augmented in the light of deeper understanding; new facts are getting known while some of the old ones need to be revised and/or retracted at the same time.

This holds especially for scientific domains – we have to incorporate newly discovered facts and possibly change the inappropriate old ones in the respective ontology as the scientific research evolves further. However, even virtually any industrial domain is dynamic – changes typically occur in product portfolios, personnel structure or industrial processes, which can all be reflected by an ontology in a knowledge management policy.

For instance, domains of e-health and biomedicine are both scientific (biomedical research) and industrial (clinical practice, pharmaceutics). The need for ontologies in biomedicine knowledge and data management has already been reflected in the community. They can serve as structured repositories giving a shared meaning to data and thus allowing to process and query them in more efficient and expressive manner. The shared meaning also results in facilitation of integration between different medical data formats once they are bound to an ontology. Moreover, the state of the art ontology-based techniques (like alignment or reasoning) can help to integrate the data even if they adhere to different ontologies. Therefore, the application domains introduced and investigated in this report are related to e-health and biomedicine scenarios, even though the general application potential of the delivered solutions is rather universal.

Large scale ontology construction is usually a result of collaboration (which involves cooperation among ontology engineers and domain experts) through a manual process of the extraction of the knowledge. However, it is not always feasible to process all the relevant data and extract the knowledge from them manually, since we might not have a sufficiently large committee of ontology engineers and/or dedicated experts at hand in order to process new data anytime it occurs. This implies a need for (partial) automation of ontology extraction and maintenance processes in dynamic and data-intensive environments. This can be achieved by automatic ontology learning. Therefore, a lifecycle of an ontology development process apt for universal application in the medicine domain should also support appropriate mechanisms for dealing with the large amounts of knowledge that are *dynamic* in nature.

The above features of an appropriate dynamic ontology lifecycle were already analysed in [NHL$^+$06]. This report describes a substantial step towards a full implementation of all the lifecycle features – a method and prototype of dynamic ontology integration. We present the integration with special emphasis put on its application in the e-health and biomedicine domains. However, it can easily be seen that the results presented here are applicable to any other dynamic knowledge engineering domain.

As an appendix of the document, we offer a report on ontology versioning survey results. The report analyses current status quo and requirements related to ontology dynamics, with opinions collected among significant representatives of the Semantic Web community. The report is not directly related to the primary content of the deliverable (therefore we present it as an isolated part). However, the survey results are a basis for further improvements of a versioning platform and other applications that have been developed within the WP2.3 research. Moreover, we reference it several times throughout this document, since some of the results are relevant in the context of public demand covered by certain features of our ontology integration method and lifecycle scenario.

1.1 Motivation

While there has been a great deal of work on ontology learning for ontology construction, e.g. [CBW02], as well as on manual or collaborative ontology development in [SEA$^+$02], relatively little attention has been paid to the user-friendly integration of both approaches within an ontology lifecycle scenario. By user-friendly we mean especially accessible to users who are not experts in ontology engineering (e.g., biomedicine researchers or practitioners). As a main contribution of this report, we introduce our framework for practical handling of dynamic and large data-sets in an ontology lifecycle, focusing particularly on dynamic integration of learned knowledge into manually maintained ontologies. However, the introduced integration mechanism is not restricted only to learned ontologies – arbitrary “external” ontology can be integrated into the primary ontology in question by the very same process.

The dynamic nature of knowledge is one of the most challenging problems in the current Semantic Web research – as can be seen in Section A.2.1 of the attached survey results report, the dynamics of ontologies in use is quite high at both schema and instance levels. Here we provide a solution for dealing with dynamics in large scale, based on properly developed connection of ontology learning and dynamic manual development. We do not concentrate on formal specification of respective ontology integration operators, we focus rather on implementation of them, following certain practical requirements:

1. the ability to process new knowledge (resources) automatically whenever it appears and when it is inappropriate for human users to incorporate it

2. the ability to automatically compare the new knowledge with a “master” ontology that is manually maintained and select the new knowledge accordingly

3. the ability to resolve possible major inconsistencies between the new and current knowledge, possibly favouring the assertions from presumably more complex and precise master ontology against the learned ones

4. the ability to automatically sort the new knowledge according to user-defined preferences and present it to them in a very simple and accessible way, thus further alleviating human effort in the task of knowledge integration

On one hand, using the automatic methods, we are able to deal with large amounts of changing data. On the other hand, the final incorporation of new knowledge is to be decided by the expert human users, repairing possible errors and inappropriate findings of the automatic techniques. The key to success and applicability is to let machines do most of the tedious and time-consuming work and provide people with concise and simple suggestions on ontology integration. Such an ontology integration method fits very well into the dynamic ontology lifecycle presented in [NHL+06]. Implementation of the method resolves one of the least researched and thus rather crucial parts of the dynamic lifecycle, constituting a substantial step towards its full deployment in practical applications.

1.2 Related Work

Within the Semantic Web research, several approaches and methodologies have been defined and implemented in the context of ontology lifecycle and integration. Recent overviews of the state-of-the-art in ontologies and related methodologies can be found in [SS04] and [GPFLC04]. However, none of them offers a direct solution to the requirements specified in Section 1.1.

A *Methontology* methodology by [FLGPJ97] was developed in the *Esperonto* EU project. It defines the process of designing ontologies and extends it towards evolving ontologies. It is provided with an ontology lifecycle based on evolving prototypes (see [FLGPR00]) and defines stages from specification and knowledge acquisition to configuration management. The particular stages and their requirements are characterised, but rather generally. The automatic ontology acquisition are considered in *Methontology*, however, its concrete incorporation into the whole lifecycle is not covered. The ODESeW and WebODE suite (see [CLCGP06]) projects base on Methontology and provide an infrastructure and tools for semantic application development/management, which is in the process of being extended for networked and evolving ontologies. However, they focus rather on the application development part of the problem than on the ontology evolution and dynamic ontology integration parts.

The methods and tools referenced above lack concrete mechanisms that would efficiently deal with the dynamics of realistic domains (so characteristic for instance for e-health and biomedicine). Moreover, the need for automatic methods of ontology acquisition in data-intensive environments is acknowledged, but the role and application of the automatic techniques is usually not clearly studied and implemented. Our approach [NHL$^+$06] offers a complex picture of how to deal with the dynamics in the general lifecycle scenario. The work we present here implements the fundamental semi-automatic dynamic integration component of the scenario.

There are more specific approaches similar to the one presented by our lifecycle framework. [DKMR$^+$06] incorporates automatic ontology extraction from a medical database and its consequent population by linguistic processing of corpus data. However, the mechanism is rather task-specific – the ontology is represented in RDF(S) format (see [BG04]) that is less expressive than the OWL language (see [BvHH$^+$04]), which we use. The extraction is oriented primarily at taxonomies and does not take the dynamics directly into account. Therefore the approach can hardly be applied in universal settings, which is one of our aims.

Protégé (see [GMF$^+$03]) and related PROMPT (see [NM02]) tools are designed for manual ontology development and semi-automatic ontology merging, respectively. PROMPT provides heuristic methods for identification of similarities between ontologies. The similarities are offered to the users for further processing. However, the direct connection to ontology learning, which we find important for dynamic and data-intensive domains is missing.

There are several works addressing directly the topic of ontology integration. [AHS05] and [CGL01] describe two approaches inspired mainly by database techniques of data mediation and query rewriting in order to provide integrated (global) view on several (local) ontologies. [HH00] present web ontology integration method using SHOE, a web-based knowledge representation language, and semi-automatically generated alignments. [DP06] implement a dynamic and automatic ontology integration technique in multi-agent environments, based on relatively simple graph ontology model inclusions and other operations. Again, none of the approaches tackles the requirements we specify in Section 1.1. Even though the methods propose solutions to the integration problem in general, there is no direct way how to integrate knowledge from unstructured resources, minimising human intervention. Furthermore, there is no emphasis on accessibility of the ontology integration to the laymen users. Our approach is distinguished by the fact that it pays special attention to these features, which we find essential for the application in dynamic domains.

1.3 Main Contribution

The main contributions of the presented work are as follows:

- proposal and implementation of a generic algorithm for dynamic integration of automatically learned knowledge into manually maintained ontologies (described in Chapters 2 and 3)

- analysis of requirements of particular realistic e-health and biomedicine use cases and identification of points which the proposed technique can contribute to in order to tackle related problems (Chapter 4)
- presentation of an example application of the implemented algorithm in a generic task of biomedical ontology extension by integrating knowledge automatically learned from textual domain resources, showing usability of the approach in the context of the presented use cases (Chapter 5)
- analysis of the general status quo, requirements and opinions concerning dynamics, in particular versioning, of ontologies within the Semantic Web community representatives (survey results report in Appendix A)

1.4 Position within the Project

This deliverable puts various existing technologies into one coherent and methodologically sound scenario of a dynamic ontology lifecycle. Within the WP 2.3, this is related to the versioning methodology and its implementation. Tasks T2.3.1. and T2.3.3.3 deal with RDF-based methodology and implementation of ontology versioning [VG06, VEK$^+$05, VKZ$^+$05] we use in the dynamic ontology lifecycle and (optionally) also in its integration part. Application of the alignment negotiation techniques within integration is an outcome of the task T2.3.7.

As we utilise argumentation-based negotiation and ontology alignment techniques within the integration, we relate to the research in WP 2.2 (Heterogeneity). Furthermore, we analyse concrete application scenarios from the bio-medicine domain. Therefore we also refer to industrial WP 1.1 – namely to the business case 2.16 (Integration of Biological Data) presented in [NM04]. Since we inherently aim at implementation of several parts of the Semantic Web framework (as proposed within D1.2.4), our work is related to the industry WP 1.2.

1.5 Structure of the Document

The rest of the report is organized as follows. Chapter 2 gives an overview of our ontology lifecycle scenario and framework, recalling the content of [NHL$^+$06] and bridging it with the progress reported here. Chapter 3 presents the new research on integration of manually designed and automatically learned ontologies in detail, forming the main technical contribution of the report. Chapter 4 discusses realistic e-health and biomedicine application domains, which our lifecycle framework can help in. In Chapter 5, we describe an example practical application of our integration technique, using real world input data (from the biomedicine research domain). Preliminary evaluation is present there as well. Moreover, respective lessons learned are discussed. Chapter 6 offers a basic user manual for the prototype API and user interface that implement a proof of concept of our ontology integration technique. The final Chapter 7 concludes the report and sums up our future work. Appendix A presents a report on the results of an ontology versioning survey we realised as a part of our research on ontology dynamics.

Chapter 2

Dynamic Ontology Lifecycle Principles and DINO

This report builds on the content of the report [NHL$^+$06], that introduced the basic principles of a dynamic ontology lifecycle scenario and suggested ways of its implementation. We recall this scenario here in Section 2.1. Section 2.2 provides a link between the lifecycle introduced in [NHL$^+$06] and the dynamic ontology integration platform presented in this report.

2.1 Recalling the Lifecycle Scenario

Figure 2.1 below depicts the scheme of the proposed dynamic and application-oriented ontology lifecycle we proposed in [NHL$^+$06].

Our ontology lifecycle builds on four basic phases: *creation* (comprises both manual and automatic ontology development and update approaches), *versioning*, *evaluation* and *negotiation* (comprises ontology alignment and merging as well as negotiation among different possible alignments). The four main phases are indicated by the boxes annotated by respective names. Ontologies or their instances in time are represented by circles, with arrows expressing various kinds of information flow. The A boxes present actors (institutions, companies, research teams etc.) involved in ontology development, where A_1 is zoomed-in in order to show the lifecycle's components in detail.

The general dynamics of the lifecycle goes as follows. The community experts (or dedicated ontology engineers) develop a (relatively precise and complex) domain ontology (the *Community* part of the *Creation* component). They use means for continuous ontology *evaluation* and *versioning* to maintain high quality and manage changes during the development process. If the amount of data suitable for knowledge extraction (e.g. domain-relevant resources in natural language) is too large to be managed by the community, *ontology learning* takes its place. Its results are *evaluated* and partially (we take only the results with quality above a certain threshold into account) integrated into the more

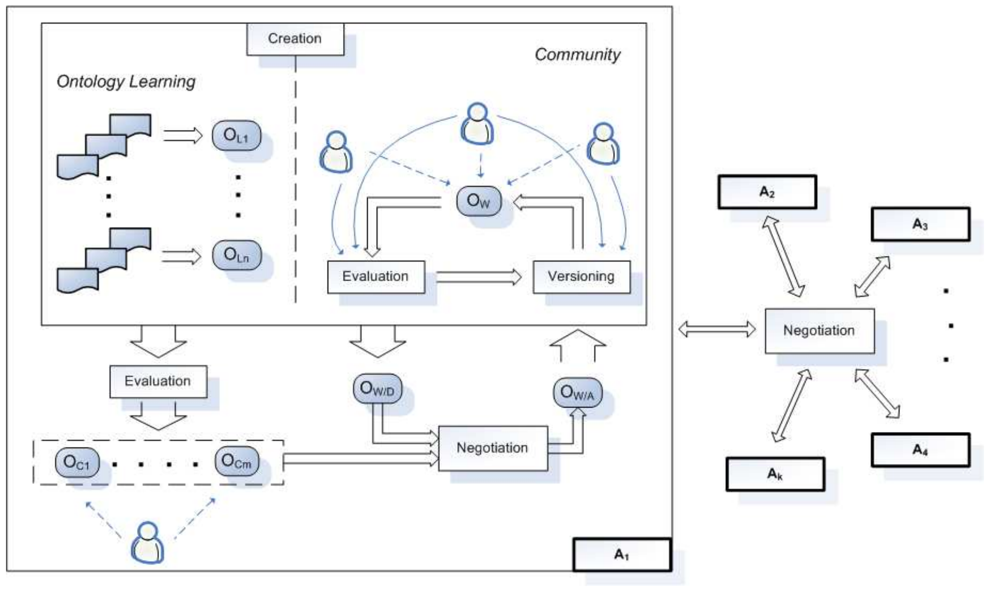


Figure 2.1: Dynamics in the ontology lifecycle

precise reference community ontology.

The integration in the scenario is based on alignment and merging covered by the *negotiation* component, complemented by inference, inconsistence resolution and diff computation. Its proposal, implementation principles and application in selected e-health use cases form the key contribution of this report (see Chapters 3 and 5 for details). The *negotiation* component takes its place also when interchanging or sharing the knowledge with other independent actors in the field. All the phases support ontologies in the standard OWL format [BvHH$^+$04]. In the following we will concentrate on the integration component. More information on other parts of the lifecycle can be found in [NHL$^+$06].

2.2 DINO and Dynamic Integration of Ontologies

DINO is an abbreviation of three key elements of our ontology lifecycle scenario and framework – *Dynamics*, *INtegration* and *Ontology*. However, the first two can also be *Data* and *INtensive*. All these features express the primary aim of our efforts – to make the knowledge efficiently and reasonably manageable in data-intensive and dynamic domains.

Since DINO can be read as Dynamic INtegration of Ontologies, too, we use the acronym in order to refer to both lifecycle and its integration part. However, this mixing of concepts and references is not essentially wrong or misguiding. As can be seen in Chapter 3 and Section 7.1.1, the dynamic integration – i.e., the DINO ontology integration framework – implements many of the essential features of the lifecycle given in Figure 2.1 in one coherent application. This has not been tackled by existing applications before.

Basically, the only phase not covered by DINO integration is the manual ontology editing and maintenance interface. However, this functionality could (and, in fact, should) be easily complemented by external state of the art tools, for instance Protégé [GMF$^+$03] and its appropriate plug-ins. We get back to this in more detail in the concluding Section 7.1.1.

Chapter 3

Dynamic Integration of Automatically Learned Knowledge

This chapter presents the technical core of the report, mainly in Section 3.1 that describes the tools and algorithms used in order to integrate an ontology. Section 3.2 offers a brief analysis of the DINO integration viewed as an ontology revision operator. This is done in line with the theoretical principles of ontology dynamics we introduced in [NHA+07].

3.1 Computing the Integration

The key novelty of the lifecycle scenario presented in [NHL+06] is its support for incorporation of changing knowledge in data-intensive domains, especially when unstructured data (i.e. natural language) is involved. This is achieved by implementation of a specific integration mechanism introduced in this section. The scheme of the integration process is depicted in Figure 3.1.

The integration scheme details the utilisation of generic lifecycle's components – mainly the (automatic) *creation* and *negotiation* – in the process of incorporation of learned ontologies into the collaboratively developed one (serving as the master model and source for stable ontology version deployment in the given settings). The master ontology – O_M circle in Figure 3.1 – is being developed within a dedicated external application (e.g., Protégé, see `http://protege.stanford.edu/`).

O_M presents a reference for integration with the O_L ontology resulting from the learning process. DINO provides user interfaces for controlling all the (semi)automatic phases of the integration process (e.g. for upload of the ontology learning resources or definition of user preferences). The final product of the integration process are natural language suggestions on the master ontology extension (see Section 3.1.6 for details). These form a base of a next version of the O_M ontology created after the integration. Note that during all phases of integration, we use the former O_M base namespace for all the other ontologies involved.

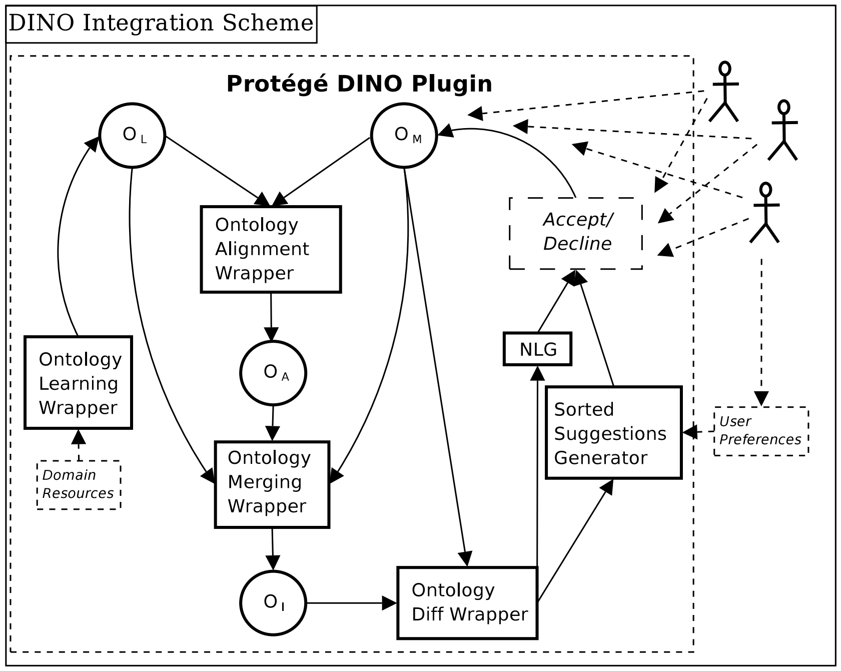


Figure 3.1: Dynamic integration scheme

We used Java^{TM} programming language to implement the algorithms presented here, employing primarily Jena 2 Ontology API[1] to handle and process the ontology models involved in the integration. Each of the phases of integration and their connections are described in detail in the following sections. See Chapter 6 for description of the delivered applications, implementing the DINO integration method.

3.1.1 Ontology Learning Wrapper

In this phase, machine learning and NLP methods are used for the processing of relevant resources and extracting knowledge from them (ontology learning). The ontology learning is realised using the Text2Onto framework (see [CV05]). Once an ontology is learned from the natural language resources uploaded via the DINO interface, it passed further to the alignment phase.

In the current implementation, only a restricted subset of possible OWL (DL) constructs is learned: `rdfs:subClassOf` axioms, class instances, named class assertions, `owl:disjointWith` axioms and `owl:ObjectProperty` assertions with `rdfs:domain` and `rdfs:range` properties specified.

Note that even an arbitrary external ontology can be integrated instead of the learned one, however, the integration results are not necessarily complete in case of more complex ontologies (e.g., containing complex restrictions and anonymous classes). This is due to

[1]See `http://jena.sourceforge.net/ontology/index.html`.

the fact that the current implementation is tailored specifically to the rather simple learned ontologies.

3.1.2 Ontology Alignment Wrapper

When the learned ontology O_L has been created, it has to be reconciled with master ontology O_M since they cover the same domain, but might be structured differently. The reconciliation of these ontologies depends on the ability to reach an agreement on the semantics of the terms used. The agreement takes the form of an alignment between the ontologies, that is, a set of correspondences (or mappings) between the concepts, properties, and relationships in the ontologies. However, the ontologies are developed in different contexts and under different conditions and thus they might represent different perspectives over similar knowledge, so the process by which to come to an agreement will necessarily only come through a negotiation process. The negotiation process is performed using argumentation-based negotiation that uses preferences over the types of correspondences in order to choose the mappings that will be used to finally merge the ontologies (see Section 3.1.3). The preferences depend on the context and situation. A major feature of this context is the ontology, and the structural features thereof, such as the depth of the subclass hierarchy and branching factor, ratio of properties to concepts, etc. The analysis of the components of the ontology is aligned with the approach to ontology evaluation, demonstrated in [DS06], and can be formalized in terms of feature metrics. Thus the preferences can be determined on the characteristics of the ontology. For example, we can select a preference for terminological mapping if the ontology is lacking in structure, or prefer extensional mapping if the ontology is rich in instances.

Thus, the alignment/negotiation wrapper interfaces two tools – one for the ontology alignment discovery and one for negotiation of agreed alignment. We call these tools *AKit* and *NKit*, respectively, within this section. For the former, we use the ontology alignment API (see [Euz04]) developed by INRIA Rhone-Alpes[2]. For the negotiation we use the framework described in [LTE$^+$06]. Both tools are used by the wrapper in order to produce O_A – an ontology consisting of axioms[3] merging classes, individuals and properties in the O_L and O_M ontologies. It is used in consequent factual merging and refinement in the ontology reasoning and management wrapper (see Section 3.1.3 for details).

The wrapper itself works according to the meta-code in Algorithm 1. The ontology alignment API offers several possibilities of actual alignment methods, which range from trivial lexical equality detection through more sophisticated string and edit-distance based algorithms to an iterative structural alignment by the OLA algorithm (see [ELTV04]). The ontology alignment API has recently been extended by a method for the calculation of a similarity metric between ontology entities, an adaptation of the SRMetric used in [VTW05]. We also consider a set of justifications, that explain why the mappings have been generated. This information forms the basis for the negotiation framework

[2]See `http://alignapi.gforge.inria.fr/` for up-to-date information on the API.

[3]Using constructs like *owl:equivalentClass, owl:sameAs, owl:equivalentProperty, rdfs:subClassOf* or *rdfs:subPropertyOf.*

Algorithm 1 Meta-algorithm of the alignment and negotiation

```
Require: $O_L, O_M$ — ontologies in OWL format
Require: $AKit, NKit$ — ontology alignment and alignment negotiation tools, respectively
Require: $ALMSET$ — a set of the alignment methods to be used
Require: $PREFSET$ — a set of alignment formal preferences corresponding to the $O_L, O_M$ ontologies (to be used in N-kit)

1: $S_A \leftarrow \emptyset$
2: for $method \in ALMSET$ do
3:   $S_A \leftarrow S_A \cup AKit.getAlignment(O_L, O_M, method)$
4: end for
5: $A_{agreed} \leftarrow NKit.negotiateAlignment(S_A, PREFSET)$
6: $O_A \leftarrow AKit.produceBridgeAxioms(A_{agreed})$
7: return $O_A$
```

that dynamically generates arguments, supplies the reasons for the mapping choices and negotiates an agreed alignment for both ontologies O_L and O_M.

3.1.3 Ontology Merging Wrapper

This wrapper is used for merging of the O_L and O_M ontologies according to the statements in O_A (each of the ontologies technically represented as a respective Jena ontology model). Moreover, the wrapper resolves possible inconsistencies caused by the merging – favouring the assertions in the O_M ontology, which are supposed to be more relevant. The resulting ontology O_I is passed to the ontology diff wrapper to be compared with the former O_M master ontology. The respective addition model forms a basis for the natural language suggestions that are produced as a final product of the integration (see Sections 3.1.4 and 3.1.5 for details).

Algorithm 2 describes the meta-code of the process arranged by the ontology merging and reasoning wrapper. We currently employ no reasoning in the $merge()$ function. How-

Algorithm 2 Meta-algorithm of the merging and inconsistency resolution

```
Require: $O_L, O_M, O_A$ — ontologies in OWL format
Require: $merge()$ — a function that merges the axioms from input ontologies, possibly implementing reasoning routines according
    to the ontology model used
Require: $C$ — set of implemented consistency restrictions; each element $r \in C$ can execute two functions $r.detect()$ and
    $r.resolve()$ that detect (and return) and resolve an inconsistency in the input ontology, respectively

1: $O_I \leftarrow merge(O_M, O_L, O_A)$
2: $inconsistencies \leftarrow \emptyset$
3: for $r \in C$ do
4:   $inconsistencies \leftarrow inconsistencies \cup r.detect(O_I)$
5:   $O_I \leftarrow r.resolve(O_I)$
6: end for
7: return $O_I, inconsistencies$
```

ever, sub-class subsumption (as implemented by the Jena framework) is used when detecting and resolving inconsistencies. The inconsistencies are constituted by user-defined restrictions. These restrictions are implemented as extensions of a generic inconsistency detector and resolver in the ontology merging wrapper. Thus we can implement either logical (in terms of Description Logics, see [BCM$^+$03]) inconsistencies, or custom-defined inconsistencies (i.e. cyclic definitions) according to requirements of particular practical applications.

The automatic inconsistency resolution itself is somewhat tricky. However, we can apply a sort of "greedy" heuristic, considering the assertions in the master O_M ontology to be more valid. Therefore we can discard axioms from O_L or O_A that are inconsistent with axioms in O_M – we call such axioms *candidate* in the text below. If there are more such axioms, we discard them one by one randomly until the inconsistency is resolved[4]. If all the conflicting axioms originated in O_M, we just report them without resolution.

We currently implement and resolve the following inconsistencies[5]:

- **sub-class** hierarchy **cycles**: these are resolved by cutting the cycle, i.e. removing a candidate *owl:subClassOf* statement;
- **disjointness-subsumption** conflicts: if classes are said to be disjoint and a sub-class relationship holds between them at the same time, a candidate conflicting assertion is removed;
- **disjointness-superclass** conflicts: if a class is said to be a sub-class of classes that are disjoint, a candidate conflicting assertion is removed;
- **disjointness-instantiation** conflicts (specialisation of the above): if an individual is said to be an instance of classes that are disjoint, a candidate conflicting assertion is removed.

Note that each element of the set of inconsistencies returned by Algorithm 2 (besides the integrated ontology itself) is associated with respective simple natural language description. The descriptions are presented for further examinations by human users in the DINO user interface.

3.1.4 Ontology Diff Wrapper

Possible extension of a master ontology O_M by elements contained in the merged and refined ontology O_I naturally corresponds to the differences between them. In particular, the possible extensions are equal to the additions O_I brings into O_M. The additions can be computed in several ways. Ontology diff wrapper in DINO offers a way how to uniformly interface the particular methods of addition computation. No matter which underlying method is employed, a respective Jena ontology model containing the respective additions is returned. Currently, the following methods are implemented within the wrapper:

[4]This is the currently implemented way, however, we plan to improve the selection of candidate axioms according to confidence ranking produced by the Text2Onto tool – similarly to the technique described in [HV05]. This is scheduled for the next version of the DINO integration library.

[5]If learned ontologies only are integrated, the resolution of these inconsistencies obviously handles all possible (logical) inconsistencies that can be introduced by integration due to restricted range of the learned axioms (see Section 3.1.1). However, this does not necessarily mean that all the inconsistencies possibly present in the master ontology will be resolved, too.

1. SemVersion-based diff computation – additions at the RDF (triple) level computed using the SemVersion library [VG06]
2. addition model computation by set operations on the underlying Jena RDF models
3. addition model computation by direct iterative querying of the former master ontology model, integrated model and alignment model for reference purposes (see Algorithm 3 for details on implementation)

For the practical experiments with ontologies, we have used the third method – mainly due to the fact that it computes the additions directly at the ontology level and not at the lower triple level (which means subsequent processing load when getting back to the ontology model again).

Algorithm 3 Meta-algorithm of the addition model computation (by direct model querying)

```
Require: $O_M, O_I, O_A$ — former master, integrated and alignment ontologies, respectively
Require: $copyResource()$ — a function that returns a copy of an ontology resource (e.g. class or property) including all relevant
    features that are bound to it (e.g. subclasses, superclasses, instances for a class or domain and range for a property)

1: $O_{added} \leftarrow \emptyset$
2: for $c \in O_I.getNamedOntologyClasses()$ do
3:     if not $O_M.contains(c)$ or $O_A.contains(c)$ then
4:         $O_{added} \leftarrow copyResource(c)$
5:     end if
6: end for
7: for $p \in O_I.getOntologyProperties()$ do
8:     if not $O_M.contains(p)$ or $O_A.contains(p)$ then
9:         $O_{added} \leftarrow copyResource(p)$
10:    end if
11: end for
12: return $O_{added}$
```

Note that the algorithm does not compute all differences between arbitrary ontologies in general. However, this is no drawback for the current implementation of DINO integration. We deal with learned ontology extending the master one. The extensions originating in automatically learned knowledge do not cover the whole range of possible OWL constructs, thus we do not need to tackle e.g. anonymous classes and restrictions in the addition model computation. Therefore the employed custom addition computation can be safely applied without any loss of information. The computed addition ontology model is passed to the suggestion sorter then (see Section 3.1.5 for details).

3.1.5 Sorted Suggestions Generator

The addition ontology passed to this component forms a base for the eventual extension suggestions for the domain experts. In order to reduce the effort in the final reviewing of the master ontology extensions, we create respective simple natural language suggestions that are associated with corresponding facts in the addition ontology model. The natural language suggestions are then presented to users – when a suggestion is accepted by the users, the associated fact is included into the master ontology model. Table 1

Table 3.1: Scheme of suggestion generation

Axiom pattern	NL suggestion scheme	Example
class c_1 is related by relation r to class c_2	The class $c_1.label()$ $f(r)$ the class $c_2.label()$.	The class ”difference_c” is disjoint with the class ”inclusion_c”.
individual i is a member of class c	The class $c.label()$ has the $i.label()$ instance.	The class ”the_cytoskeleton_organiser_c” has the ”centrosome_i” instance.
property p_1 with features features x is related to property p_2 by relation r	There is a $p_1.label()$ $g(x)$ property. It is $f(r)$ $p_2.label()$.	There is a ”contain_r” object property. Its range is the ”organ_c” class.
property p_1 with features x has domain/ range class c	There is a $p_1.label()$ $g(x)$ property. Its domain/range is the $c.label()$ class.	There is a ”contain_r” object property. It has the ”has_part_r” superproperty.

shows a scheme of the natural language (NL) suggestion generation. The r variable represents possible relations between classes or properties (e.g. `rdfs:subClassOf`, `rdfs:subPropertyOf` or `owl:disjointWith`), mapped by the function $f()$ to a respective natural language representation (e.g. *is a sub-class of*, *is a sub-property of* or *is disjoint with*). The x variable represents possible features of a property (e.g. `owl:ObjectProperty` or `owl:FunctionalProperty`, mapped by the function $g()$ to a respective natural language representation (e.g. *object* or *functional*).

In general, the number of suggestions originating from the addition ontology model can be quite large, so an ordering that takes a relevance measure of possible suggestions into account is needed. Thus we can for example eliminate suggestions with low relevance level when presenting the final set to the users (without overwhelming them with a large number of possibly irrelevant suggestions).

As a possible solution to this task, we have proposed and implemented a method based on string subsumption and Levenshtein distance [Lev66]. These two measures are used within relevance computation by comparing the lexical labels occurring in a suggestion with respect to two sets (S_p, S_n) of words, provided by users. The S_p and S_n sets contain preferred and unwanted words respectively, concerning the lexical level of optimal extensions. The general structure of the sorting function is given in Algorithm 4.

Algorithm 4 Meta-algorithm of relevance-based triple sorting

```
Require: $SUGGESTIONS$ — list of suggestions
Require: $PREF = \{S_p, S_n\}$ — user preferences

1: $HASH = \{\}$
2: for $T \in SUGGESTIONS$ do
3:     $HASH[getScore(T, S_p, S_n)] \leftarrow T$
4: end for
5: return $sort(HASH)$
```

The $getScore()$ function is crucial in the sorting algorithm. It is given by the formula:

$$getScore(T, S_p, S_n) = rel(T, S_p) - rel(T, S_n),$$

where $rel(T, S)$ is a function measuring the relevance of the suggestion T with respect to the words in the set S. The higher the value, the more relevant the triple is. We develop the relevance function in detail in Algorithm 5.

Algorithm 5 The relevance function

```
Require: $S_t$ — a set of (possibly multiword) lexical terms occurring in the suggestion
Require: $S$ — set of words
Require: $\rho \in (0, 1)$ influences the absolute value of relevance measure
Require: $t$ — integer constant; maximal allowed distance
Require: $levDist(s_1, s_2)$ — Lev. distance implementation

1: for $elem \in S_t$ do
2:   $R_{elem} \leftarrow 0$
3: end for
4: for $elem \in S_t$ do
5:   if $elem$ is a substring of or equals to any word in $S$ or vice versa then
6:     $R_{elem} \leftarrow 1$
7:   else
8:     $d \leftarrow \infty$
9:     for $v \in S$ do
10:      if $levDist(elem, v) < d$ then
11:        $d \leftarrow levDist(elem, v)$
12:      end if
13:    end for
14:    if $d \leq t$ then
15:      $R_{elem} \leftarrow (1 - \frac{d}{t+1})$
16:    else if $elem$ is a multiword term then
17:      $L \leftarrow$ set of single terms in the $elem$ label expression
18:      $EXP \leftarrow 0$
19:      for $u \in L$ do
20:        if $u$ is a substring of or equals to any word in $S$ or vice versa then
21:          $EXP \leftarrow EXP + 1$
22:        else
23:          $d \leftarrow \infty$
24:          for $v \in S$ do
25:            if $levDist(u, v) < d$ then
26:              $d \leftarrow levDist(u, v)$
27:            end if
28:          end for
29:          if $d \leq t$ then
30:            $EXP \leftarrow EXP + (1 - \frac{d}{t+1})$
31:          end if
32:        end if
33:      end for
34:      if $EXP = 0$ then
35:        $R_{elem} \leftarrow 0$
36:      else
37:        $R_{elem} \leftarrow \rho^{\frac{1}{EXP}}$
38:      end if
39:    end if
40:  end if
41: end for
42: return $\frac{\sum_{elem \in S_t} R_{elem}}{|S_t|}$
```

The function naturally measures the "closeness" of the labels occurring in the suggestion to the set of terms in S. The value of 1 is achieved when the label is a direct substring of or equal to any word in S or vice versa. When the Levenshtein distance between the label and a word in S is lower than or equal to the defined threshold t, the relevance decreases from 1 by a value proportional to the fraction of the distance and t. If this is not the case (i.e. the label's distance is greater than t for each word in S), a similar principle is applied for possible word-parts of the label and the relevance is further proportionally decreased (the minimal possible value being 0).

Note that the complexity of sorting itself mostly contributes to the overall complexity of the relevance-based sorting of suggestions. As can be found out from Algorithm 5, the complexity is in $O(cmnl^2 + m \log m)$ (c – maximal number of terms occurring in a suggestion, thus a constant; m – number of suggestions; n – number of words in the preference sets; l – maximal length of a word in suggestion terms, basically a constant), which gives $O(m(n+\log m))$. As the size of the sets of user preferences can be practically bounded by a constant[6], we obtain the $O(m \log m)$ complexity class with respect to the number of suggestions, which is feasible for most practical applications.

3.1.6 Natural Language Generation (NLG) Component

The DINO framework is supposed to be used primarily by users who are not experts in ontology engineering. Therefore the suggestions are produced in a form of very simple natural language statements, as seen in the previous section. Moreover, we automatically create a natural language representation of the whole addition model, interfacing the framework described in [TPCB06]. This is meant to further support laymen users by readable representation of the whole addition model in order to give them an overall impression of the changes.

The single suggestions are still bound to the underlying statement in the addition ontology model. Therefore a user can very easily add the appropriate OWL axioms into the new version of the O_M master ontology without actually dealing with the intricate OWL syntax itself. Concrete examples of both suggestions and continuous natural language representation of the addition model are given in Chapter 5.

3.2 Integration as an Ontology Revision Operator

In [NHA$^+$07], Chapter 2, we define several postulates for rational DL-based ontology change operators (namely for *contraction* and *revision*, i.e. for removing and adding of axioms into an ontology). Since we support OWL DL ontologies in the integration process described above, we fit into this theoretical framework. Moreover, the integration process in fact consists of axiom addition into the master ontology.

Since the final integration step is done by human users, the whole process can be hardly covered by a formal definition of an ontology revision operator (due to the non-determinism of the human involvement – human users may possibly decide to include modified set of axioms, using the automatically offered revision set as only a kind of “inspiration”). However, we can restrict the situation a bit by, (1), taking only integration of learned ontologies into account and, (2), considering all the automatically generated extension axioms as a base for the revision.

In the simplified case, we have only a restricted subset of possible OWL DL constructs in the revision set generated by the integration process (due to simple nature of the learned

[6] In theory, this constant can be quite large, however, in practical scenarios, users usually do not define infeasibly large sets of preferences for particular integration iterations.

ontologies, see Section 3.1.1). Moreover, the content of the revision set is precisely determined. Every possible inconsistency is resolved by default in this case restricted only to learned ontologies (see Section 3.1.3).

Let us go through the postulates **(O+1)** to **(O+4)** in [NHA+07] now, showing that the revision operator of restricted DINO integration conforms to them. We do not consider the postulate **(O+5)**, as it involves contraction operator that is not implemented in DINO. In the postulates, we use the same notation as in [NHA+07] – i.e., O stands for master ontology in our context, X for the revision set, $+$ for the revision operator (integration), Cn for a Tarski-like deductive closure of an ontology, L for a set of all possible axioms of a given ontology language an $\cong$ for semantic ontology equivalence. The postulates assume that the O ontology is consistent, which is generally not the case for the real world ontologies that can form a master ontology for the integration process. However, if we further restrict our situation and take only consistent master ontologies into account (which is not that harmful, considering the fact that ontologies *should* be consistent), we can show the conformance of DINO integration to the postulates as follows:

- **(O+1)** $X \subseteq O+X$. The inclusion of axioms associated with suggestions, generated as described in Section 3.1.5, is based on a set union with the master ontology axioms, therefore the postulate holds for the restricted DINO integration.
- **(O+2)** If $Cn(O \cup X) \neq L$ then $O + X = O \cup X$. Combining the discussion of postulates **(O+1)** and **(O+3)**, we can see that this postulate obviously holds, too.
- **(O+3)** If $Cn(X) \neq L$ then $Cn(O + X) \neq L$. We know that X is consistent (all the inconsistencies are resolved in our restricted situation). The integration is based on the set union, as stated above. Union of two consistent sets of axioms does not necessarily have to be consistent. However, the inconsistencies in the revision set are resolved after the *mapping* with the master ontology (see Section 3.1.2). Therefore, all inconsistencies possibly originating from the trivial set union merge of learned and master ontologies are already resolved concerning the X set. Thus the postulate holds.
- **(O+4)** If $X \cong Y$ then $O + X \cong O + Y$. Using the postulate **(O+2)**, we can assume that $O + X = O \cup X$ and $O + Y = O \cup Y$. Therefore, if $X \cong Y$, then obviously $O \cup X \cong O \cup Y$.

Chapter 4

Selected Application Domains

The application domains are discussed according to the use case areas identified in [Eic06] within the EU IST 6th Framework project RIDE. The areas are rather broad, however, we can track the needs that can be at least partially covered by an appropriate ontology lifecycle framework. We do this for five selected domains here:

- *Longitudinal Electronic Health Record* – Section 4.1
- *Epidemiological Registries* – Section 4.2
- *Public Health Surveillance* – Section 4.3
- *Management of Clinical Trials* – Section 4.4
- *Genomics and Proteomics Research* – Section 4.5

The DINO ontology lifecycle framework can serve as a substantial part of the respective semantics-enabled solutions in all of the presented application domains, since it provides complete framework for ontology creation, maintenance and mediation in data-intensive dynamic environments.

Note that there is one generic way of DINO application possibly appropriate and desired throughout all the presented use cases. In practice, particular institutions and/or companies may very often want to extend a standard upper biomedical ontologies by their custom domain-specific knowledge. This knowledge can typically be present within large amount of natural language resources. Application of DINO is straightforward in such cases – ontology learned from the textual resources is semi-automatically integrated into a master ontology, i.e. the upper ontology to be extended. This way of DINO utilisation is further described more concretely for two selected application domains in Chapter 5.

4.1 Longitudinal Electronic Health Record

The main topic here is development of standards and platforms supporting creation and management of long-term electronic health records of particular patients. These should be able to integrate various sources of data coming from different medical institutions a patient may have been treated in during his whole life.

Need for integration of different data sources imposes need for respective, possibly automatised, technologies able to facilitate this task. Common abstract conceptual structure of the electronic health record needs to be populated and/or extended by concrete data, present very often in unstructured natural language form. The electronic health record should also be opened to efficient and expressive querying.

Ontologies bound to patient data resources in particular institutions can very naturally support integration of respective data into longitudinal electronic health records. Once there is an ontology describing the underlying data, we can directly use the integration mechanism presented here in order to manage the needed integration semi-automatically. Moreover, the DINO framework can serve for easy and laymen-oriented ontology development already at the particular institutions' side. Support for ontology learning directly facilitates the population/extension. Querying of ontology-enabled electronic health records is straightforward in our framework, since it is possible using the state of the art OWL reasoning tools.

4.2 Epidemiological Registries

Epidemiology is concerned with events occurring in population – diseases, their reasons, statistical origins and their relation to a selected population sample's socioeconomic characteristic. Epidemiological registries should be able to reasonably store and manage data related to population samples and their medical attributes in order to support efficient processing of the respective knowledge by the experts.

The needs of this application domain can be seen as an extension of the needs in Section 4.1. Again, we have to integrate various sources of patient data, however, this time we would rather like to gather knowledge from the electronic health records to create population-wise repositories. Furthermore, when studying relations between diseases and population samples, global drug efficiency measures, etc., we need efficient mechanisms of dealing with classes and their attributes while querying the stored data.

Once there are ontology-enabled electronic health records (as described in Section 4.1), we can easily integrate them within another instance of "epidemiology" ontology developed in the DINO framework. The ontology representation of data in an epidemiology repository can add additional dimension to usual statistical processing of population data. Using DL-based reasoning on the data semantics expressed by the respective OWL ontologies, we could obtain additional qualitatively different (symbolic) valuable results.

4.3 Public Health Surveillance

Public health surveillance presents ongoing collection, analysis, interpretation and dissemination of health-related data in order to facilitate a public health action reducing mortality and/or improving health [Eic06]. It has several public health functions, including estimating the impact of a disease, determining the distribution and spread of illness, outbreak detection or evaluating prevention and control measures.

The needs are similar to Section 4.2. However, there are important differences, as the active public health functions (e.g. outbreak detection) directly require efficient dynamic processing of newly coming data. Moreover, the need for tools able to automatically process free natural language text is explicitly emphasised in this application domain concerning the dynamic knowledge processing.

The basic design principles of DINO directly conform to the needs here. Ontologies created and dynamically extended by or confronted with newly coming critical data can efficiently support expert decisions in risk management tasks. Continuous integration of less critical data from various sources can back the study of public health issues in long term perspective at the same time.

4.4 Management of Clinical Trials

Briefly put, clinical trials are studies of the effects of newly developed drugs on selected sample of real patients. They are essential part of approval of new drugs for normal clinical use and present an important bridge between medical research and practice.

A need for electronic representation of clinical trials data is emphasised. However, even if the data are electronically represented, problems with their heterogeneity and integration occur as there are typically several different institutions involved in a single trial. Efficient querying is demanded, stating it can reduce the overall cost of clinical trials significantly.

Once again, ontologies developed and/or mediated using the DINO framework can facilitate the integration problems. Universal formal OWL representation allows unified querying of different clinical trial data then.

4.5 Genomics and Proteomics Research

Similarly to Section 4.4, this application domain is related to translational medicine and to bridging the research and clinical practice. Genomics and proteomics research studies genes, proteins, their effects, mutual influences and interactions within human organism. It covers both basic and applied medical and pharmaceutical research.

Integration of various knowledge repositories is needed when pursuing study in a particular sub-domain of genomics and proteomics. We may need to integrate specific knowledge e.g. in GO or UMLS controlled dictionaries[1] and in clinical reports on drug compounds and their effects in practice. Merits of efficient querying of the knowledge are obvious even in this case.

The ontology development and integration services, together with OWL-based formalised support for efficient reasoning, cover the needs even in this application domain to some extent. Unfortunately, there are practical limitations mainly in the lack of formal structure of genomics and proteomics knowledge bases. Their transformation into a formal ontology is thus not trivial. However, after development/adaptation and implementation of a certain methodology and rules of this translation, the semi-automatic relevance-guided integration proposed in DINO can help in this task even if the translation itself would not perform very well.

[1]See `http://www.ebi.ac.uk/ego` and `http://umlsinfo.nlm.nih.gov`, respectively.

Chapter 5

Sample Experiment with the DINO Integration

We applied the integration technique described in Section 3 in the context of data typical for biomedical research. This example application is similar to the situation described in the *genomics and proteomics research* use case (see Section 4.5). However, the typical way of exploiting the DINO integration technique reported in this section is rather general, since it aims at cost-efficient extension of a master ontology by knowledge learned from empirical data. Thus, a similar deployment of the integration can actually help to tackle needs of any other use case we have analysed.

5.1 Characteristics of the Experiment

Real world data for the master ontology and ontology learning sources were used. More specifically, we employed resources from CO-ODE biomedicine ontology fragment repository[1] and data from relevant Wikipedia topics, respectively.

Rigorous evaluation of the whole process of integration is a complex task involving lot of open problems as its sub-problems (for instance, there is no standard ontology evaluation process applicable in general – see [HSG$^+$05, DS06]). Moreover, there is an emphasis on the human-readable and laymen oriented form of the integration process results. This dimension forms a primary axis of the evaluation, however, its realisation involves logistically demanding participation of a broader (biomedicine) expert community.

Accomplishing the above tasks properly is a part of our future work. Nonetheless, there are several aspects that can be assessed and reported even without devising an optimal ontology evaluation method (which may be impossible anyway) and/or getting involved large representative sample of domain experts:

- features of the learned ontology (e.g. size or complexity)

[1] See `http://www.co-ode.org/ontologies`.

- mappings established by alignment
- basic assessment of the quality and correctness of suggestions and their sorting according to defined preferences

These factors of integration are analysed and discussed within an experimental application described in Section 5.2.

The negotiation component has recently been evaluated separately as a stand-alone module, using the Ontology Alignment Evaluation Initiative test suite[2] and experiments on the impact the argumentation approach has over a set of mappings. A comparison wrt. current alignment tools is presented in [LBT$^+$07]. The preliminary results of these experiments are promising and suggest that the argumentation approach can be beneficial and an effective solution to the problem of dynamically aligning heterogeneous ontologies. This justifies also the application of the implemented technique in the ontology integration task.

5.2 Evaluating Integration of Biomedical Research Knowledge

In order to show the basic features of our novel integration technique in practice, we tested the implementation using knowledge resources from biomedicine domain[3]. In particular, we combined fragments of GO cellular component description and eukaryotic cell description[4] to form the master ontology. In the example scenario, we wanted to extend this master ontology using content of Wikipedia entries on `Cells_(biology)` and `Red_blood_cell`. These resources were passed to the ontology learning DINO component and respective ontology was learned. Both master and learned ontology samples are displayed in Figure 5.1 (on the left-hand and right-hand side, respectively). Note that these master and learned ontologies correspond to the O_M, O_L ontologies displayed in Figure 3.1, Chapter 3. The names in learned ontology have specific suffixes (i.e. "*_c*"). This is due to naming conventions of the ontology learning algorithm we use. We keep the suffixes in suggestions, since they help to easily discriminate what comes from empirical data and what from the master ontology. However, we filter them out when generating the text representing the whole extension model (see below for examples).

Table 2 compares metric properties of the master and learned ontologies, as computed by the Protégé tool. The meaning of the column headers is as follows:

[2]See `http://oaei.ontologymatching.org/`.

[3]Should the reader be interested, all relevant resources used and/or created during the described experiment are available at `http://smile.deri.ie/resources/2007/08/31/dino_exp_data.zip`

[4]Samples downloaded from the CO-ODE repository, see `http://www.co-ode.org/ontologies/bio-tutorial/sources/GO_CELLULAR_COMPONENT_EXTRACT.owl` and `http://www.co-ode.org/ontologies/eukariotic/2005/06/01/eukariotic.owl`, respectively.

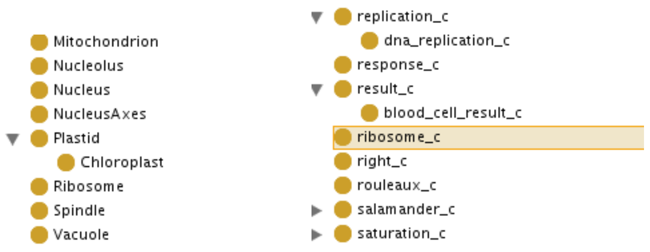


Figure 5.1: Sample from master and learned ontology

Table 5.1: Metrics of master and learned ontologies

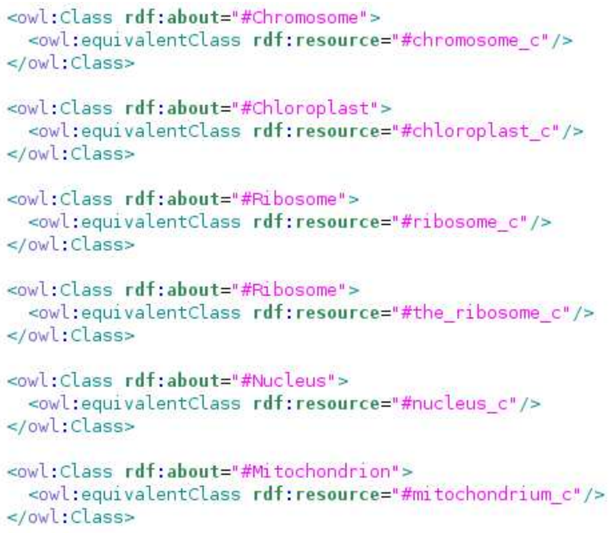
Ontology	Named classes (all/prim./def.)	Par. (mn./ md./max)	Sibl. (mn./ md./max)	Anonym. classes	Properties (all/obj./dt.)	DL expr.
Learned	391 / 379 / 12	3 / 1 / 5	7 / 1 / 16	0	13 / 13 / 0	$\mathcal{ALC}(D)$
Master	40 / 36 / 4	2 / 1 / 2	5 / 1 / 15	16 (restr.)	1 / 1 / 0	$\mathcal{ALCN}$

1. ontology type
2. number of named classes (all/primitive/defined)
3. number of parents (mean/median/maximum)
4. number of siblings (mean/median/maximum)
5. number of anonymous classes (restrictions)
6. number of properties (all/object/datatype)
7. Description Logics expressivity

The learned ontology has higher ratio of primitive classes, moreover, it contains no restriction class definitions. There are some simple object properties with both domains and ranges defined. Its DL expressivity allows concept intersection, full universal and existential quantification, atomic and complex negation and datatypes. The expressivity of the master ontology does not involve datatypes, however, it allows numeric restrictions. Summing up, the master ontology contains several complicated constructs not present in the learned ontology, however, the ontology learned only from two simple and relatively small resources is much larger.

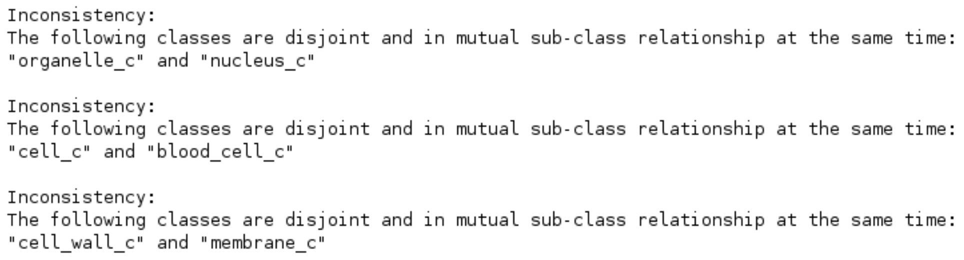
When computing the negotiated alignment (the O_A ontology as given in Figure 3.1, Chapter 3) between master and learned ontology, 207 mappings were produced and among them, 16 were accepted. A sample from the alignment ontology is displayed in Figure 5.2.

Merging of the learned and master ontologies according to the computed alignments results in several inconsistencies – the report generated by DINO is displayed in Figure 5.3. Two of these three inconsistencies are resolved correctly (according to human



```
<owl:Class rdf:about="#Chromosome">
  <owl:equivalentClass rdf:resource="#chromosome_c"/>
</owl:Class>

<owl:Class rdf:about="#Chloroplast">
  <owl:equivalentClass rdf:resource="#chloroplast_c"/>
</owl:Class>

<owl:Class rdf:about="#Ribosome">
  <owl:equivalentClass rdf:resource="#ribosome_c"/>
</owl:Class>

<owl:Class rdf:about="#Ribosome">
  <owl:equivalentClass rdf:resource="#the_ribosome_c"/>
</owl:Class>

<owl:Class rdf:about="#Nucleus">
  <owl:equivalentClass rdf:resource="#nucleus_c"/>
</owl:Class>

<owl:Class rdf:about="#Mitochondrion">
  <owl:equivalentClass rdf:resource="#mitochondrium_c"/>
</owl:Class>
```

Figure 5.2: Sample alignment



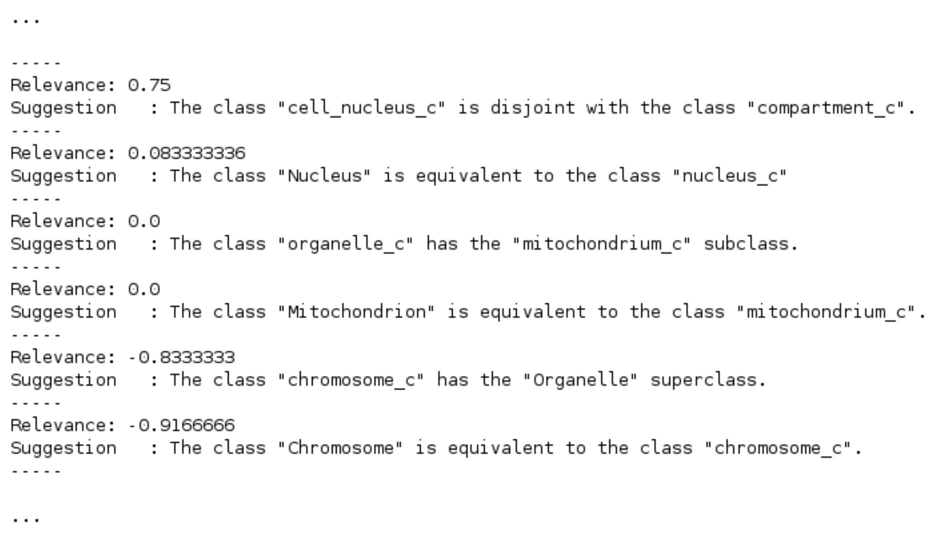
```
Inconsistency:
The following classes are disjoint and in mutual sub-class relationship at the same time:
"organelle_c" and "nucleus_c"

Inconsistency:
The following classes are disjoint and in mutual sub-class relationship at the same time:
"cell_c" and "blood_cell_c"

Inconsistency:
The following classes are disjoint and in mutual sub-class relationship at the same time:
"cell_wall_c" and "membrane_c"
```

Figure 5.3: Report on inconsistencies

intuition) by the algorithm, forming an integrated ontology O_I, as displayed in Figure 3.1, Chapter 3.

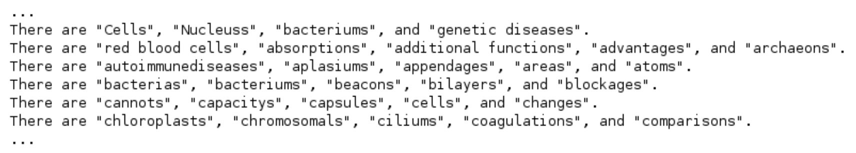
After resolving the inconsistencies and generating the addition model, natural language suggestions (associated with respective OWL axioms) are produced. Sample suggestions associated with respective relevance measures are displayed in Figure 5.4. A portion of the continuous text generated by the NLG component that is corresponding to the addition model is displayed in Figure 5.5. This text is presented to users in the DINO GUI interface (after the necessary post-processing, parsing, filtering and highlighting of the ontology terms, which is currently still work in progress). It provides users with additional source of lookup when deciding which suggestions to accept into the next version of the master ontology.

The suggestions are the ultimate output of the integration algorithm. Their main purpose is to facilitate laymen effort in incorporation of new knowledge from unstructured resources into an ontology. Therefore we performed basic evaluation of several parame-



```
...

-----
Relevance: 0.75
Suggestion   : The class "cell_nucleus_c" is disjoint with the class "compartment_c".
-----
Relevance: 0.083333336
Suggestion   : The class "Nucleus" is equivalent to the class "nucleus_c"
-----
Relevance: 0.0
Suggestion   : The class "organelle_c" has the "mitochondrium_c" subclass.
-----
Relevance: 0.0
Suggestion   : The class "Mitochondrion" is equivalent to the class "mitochondrium_c".
-----
Relevance: -0.8333333
Suggestion   : The class "chromosome_c" has the "Organelle" superclass.
-----
Relevance: -0.9166666
Suggestion   : The class "Chromosome" is equivalent to the class "chromosome_c".
-----

...
```

Figure 5.4: Sample suggestions



```
...
There are "Cells", "Nucleuss", "bacteriums", and "genetic diseases".
There are "red blood cells", "absorptions", "additional functions", "advantages", and "archaeons".
There are "autoimmunediseases", "aplasiums", "appendages", "areas", and "atoms".
There are "bacterias", "bacteriums", "beacons", "bilayers", and "blockages".
There are "cannots", "capacitys", "capsules", "cells", and "changes".
There are "chloroplasts", "chromosomals", "ciliums", "coagulations", and "comparisons".
...
```

Figure 5.5: Sample from the generated continuous text

ters that influence actual applicability of the suggestions. We ran the integration algorithm on the same data with four different suggestion-preference sets, simulating four generic trends in the preference definition:

- specification of rather small number of preferred terms, no unwanted terms
- specification of rather small number of preferred and unwanted terms
- specification of larger number of preferred terms, no unwanted terms
- specification of larger number of preferred and unwanted terms

Table 3 gives an overview of the four iterations, the particular preferred and unwanted terms and distribution of suggestions into relevance classes. The terms were set by a human user arbitrarily, reflecting general interest in clinical aspects of the experimental domain knowledge. The terms in preference sets reflect possible topics, which would the users like to be covered by the automatic extension of their current ontology (that has been covering these topics insufficiently so far). S_+, S_0 and S_- are classes of suggestions with relevance greater, equal and lower than zero, respectively ($S = S_+ \cup S_0 \cup S_-$).

Table 5.2: Iterations – the preference sets and sizes of the resulting suggestion classes

Iteration	**Preferred**	**Unwanted**	$\lvert S_+\rvert$	$\lvert S_0\rvert$	$\lvert S_-\rvert$	$\lvert S\rvert$
I_1	cell; autoimmune disease; transport; drug; gene; DNA	$\emptyset$	310	429	0	739
I_2	cell; autoimmune disease; transport; drug; gene; DNA	bacteria; prokaryotic; organelle; wall; chromosome; creation	250	344	145	739
I_3	cell; autoimmune disease; transport; drug; gene; DNA eukaryotic; organ; function; part; protein; disease; treatment; cell part immunosuppression; production	$\emptyset$	485	254	0	739
I_4	cell; autoimmune disease; transport; drug; gene; DNA eukaryotic; organ; function; part; protein; disease; treatment; cell part immunosuppression; production	bilayer; bacteria; prokaryotic; additional function; organelle; macromollecule; archaeon; vessel; wall; volume; body; cell nucleus; chromosome; erythrocyte; creation	314	292	133	739

For each of the relevance classes induced by one iteration, we randomly selected 20 suggestions and computed two values on this sample:

- $P_x, x \in \{+, 0, -\}$ – ratio of suggestions correctly placed by the sorting algorithm into an order defined by a human user for the same set (according to the interest defined by the particular preferences)

- $A_x, x \in \{+, 0, -\}$ – ratio of suggestions that are considered appropriate by a human user according to his or her knowledge of the domain (among all the suggestions in the sample)

The results are summed up in Table 4. More details on interpretation of all the experimental findings are given in consequent Section 5.3.

Table 5.3: Evaluation of random suggestion samples per class

Iteration	P_+	A_+	P_0	A_0	P_-	A_-
I_1	0.45	0.75	0.90	0.60	-	-
I_2	0.45	0.75	1.00	0.80	0.60	0.70
I_3	0.70	0.80	0.95	0.75	-	-
I_4	0.55	0.75	0.70	0.85	0.50	0.85

5.3 Discussion of the Presented Results

The DINO integration library allows users to submit the resources containing knowledge they would like to reflect in their current ontology. The only thing that is needed is to specify preferences on the knowledge to be included using the sets of preferred and unwanted terms. After this, sorted suggestions on possible ontology extensions (after resolution or reporting of possible inconsistencies) can be produced and processed in minutes, whereas the purely manual development and integration of respective ontology would take hours even for relatively simple natural language resources. Moreover, it would require a certain experience with knowledge engineering, which does not have to be true for biomedicine domain experts.

In Section 5.2 we described application of our integration technique to an extension of biomedical research ontology fragment. The analysed results show that the suggestions produced are mostly correct (even though rather simple and sometimes obvious) with respect to the domain in question, ranging from 50% to 85% among the algorithm iterations. The relevance-based sorting according to preferences is more appropriate in case of irrelevant (zero relevance) suggestions, ranging from 70% to 100% of correctly placed suggestions. Its precision in case of suggestions with positive and negative relevance is relatively lower, ranging from 45% to 70%. More terms in the preference sets cause better sorting performance (the ratio of appropriate suggestions being independent on this fact). Thus, the best discrimination in terms of presenting the most relevant suggestions first is achieved for larger preference sets. However, even the discrimination for relatively smaller sets is fair enough (as seen in Table 3 in the previous section).

The automatically produced natural language suggestions can be very easily browsed and assessed by users who are not familiar with ontology engineering at all. Since the respective axioms are associated to the suggestions, their inclusion into another version of the master ontology is pretty straightforward once a suggestion is followed by a user. The DINO integration technique still needs to be evaluated with a broader domain expert audience involved, however, even the preliminary results presented here are very promising in the scope of the requirements specified in Section 1.1.

Chapter 6

Basic User Manual for DINO Applications

This chapter presents a basic user manual for the software implementing the DINO ontology integration functionalities. You can download all the related materials, source code and applications at `http://smile.deri.ie/resources/2007/dino`. The user manual consists of three parts:

1. Section 6.1 – general comments on prerequisites of the DINO ontology integration applications
2. Section 6.2 – description of a GUI user interface to the DINO ontology integration library, namely comments on installation, execution and typical actions (associated by respective screenshots)
3. Section 6.3 – description of an API programmatic interface to the DINO ontology integration library, namely comments on its installation and sample code, referencing respective detailed JavaDoc API whenever needed

6.1 Prerequisites

General prerequisite is a machine with Java SE platform installed. For both API and GUI interfaces, the Java virtual machine (JVM) should be launched with 768MB or more of dedicated heap memory in order to ensure smooth performance (lower amounts of memory will do, too, however, it may reasonably slow down or even disable certain phases of the ontology learning or integration). You can set the Java heap memory for instance using the `-XmsINIT_SIZEm` and `-XmxMAX_SIZEm` parameters of the `java` command in order to set the initial heap size to `INIT_SIZE` and maximum heap size to `MAX_SIZE` megabytes, respectively.

Required 3rd party applications are covered in the following paragraphs of this section.

GATE - NLP and IE framework GATE [CMBT02] is a general architecture for text engineering with wide range of functions and possible applications. DINO uses GATE API for several tasks – mainly for natural language text preprocessing in the ontology learning phase and for the natural language generation component. Therefore, it needs to be installed on your machine before you can start to use the DINO applications.

The DINO framework has been tested with GATE versions 3.1, 3.2 and 4 (available at `http://gate.ac.uk/`). However, there may be (rather unlikely) settings (e.g., when working with the DINO API interface in some versions of Eclipse on certain platforms) hampering using DINO with the official GATE versions. If this is the case, you may try to use a tested alpha-version available at `http://smile.deri.ie/resources/2007/dino/download/gate4a.zip`.

Text2Onto - ontology learning tool and library Text2Onto is an ontology learning library and GUI-enabled application framework aimed at ontology learning from a natural language text corpora. We interface the Text2Onto API in the ontology learning component of DINO. The tool is available at `http://ontoware.org/projects/text2onto/`. Versions 130607 and 180607 have been tested; any future version should work fine with DINO.

See Chapter 2 in [VS05] (available at `http://www.sekt-project.org/rd/deliverables/` under ID 3-3-1) in order to figure out how to properly configure the Text2Onto library.

6.2 DINO GUI

The DINO GUI interface is available at `http://smile.deri.ie/resources/2007/dino/download.html`. Note that the GUI version 0.1 is a public alpha testing version, not intended for production use as such.

6.2.1 Notes on Installation and Configuration

Recommended GATE and Text2Onto installation location After downloading the DINO GUI package, extract its content into a directory on your machine (this directory is referred to as `DINO_GUI_HOME` in the following text). It is recommended to install/extract the GATE and Text2Onto tools into the `DINO_GUI_HOME` directory as `DINO_GUI_HOME/gate4a` and `DINO_GUI_HOME/text2onto18` directories, respectively. You can also create respective symbolic links from your custom installation locations to these recommended directories.

Set the `DINO_GUI_HOME/run.bat` start-up script up If you have installed/extracted GATE and Text2Onto into `DINO_GUI_HOME/gate4a` and `DINO_GUI_HOME/-`

`text2onto18` directories, you will only need to set up a value of `DINO` variable to point to your `DINO_GUI_HOME` directory:

```
set DINO=<DINO_GUI_HOME>
```

In case you have not used the recommended locations for GATE and Text2Onto, you will need to change `GATE` and `T2O` variables to point to their home directories as well:

```
set DINO=<DINO_GUI_HOME>
...
set T2O=<Text2Onto_HOME>

set GATE=<GATE_HOME>
```

If you intend to use DINO GUI on another platform (i.e., non-Windows system), you can launch it using the `java` or equivalent command directly, with the command line parameters set according to the Windows start-up script included in the package.

Set up the `Text2Onto_HOME/text2onto.properties` file Modify the `Text2Onto_HOME/text2onto.properties` file according to the following:

```
language=english
gate_dir=<Text2Onto_HOME>/3rdparty/gate/
gate_app=application.gate
jape_main=main.jape
stop_file=stopwords.txt
creole_dir=<Text2Onto_HOME>/3rdparty/gate/
jwnl_properties=<Text2Onto_HOME>/3rdparty/jwnl/file_properties.xml
temp_corpus=<Text2Onto_HOME>/temp
icons=<Text2Onto_HOME>/icons/
datastore=serial
tagger_dir=f:/treetagger/bin/
spanish_wn_dir=f:/wordnet_es/
```

where `<Text2Onto_HOME>` is your Text2Onto home directory.

Important note: After setting up the `Text2Onto_HOME/text2onto.properties` file you have to copy it into the `DINO_GUI_HOME` directory (otherwise Text2Onto will not see it).

Set the `GATE_HOME` directory in the DINO interface After you have launched DINO GUI (using either the start-up script or direct invocation by `java` command), you have to set up the GATE home directory in the *Settings* menu item of the interface. Select the item `Set paths` there and put a path to your `GATE_HOME` directory into the configuration window that pops up.

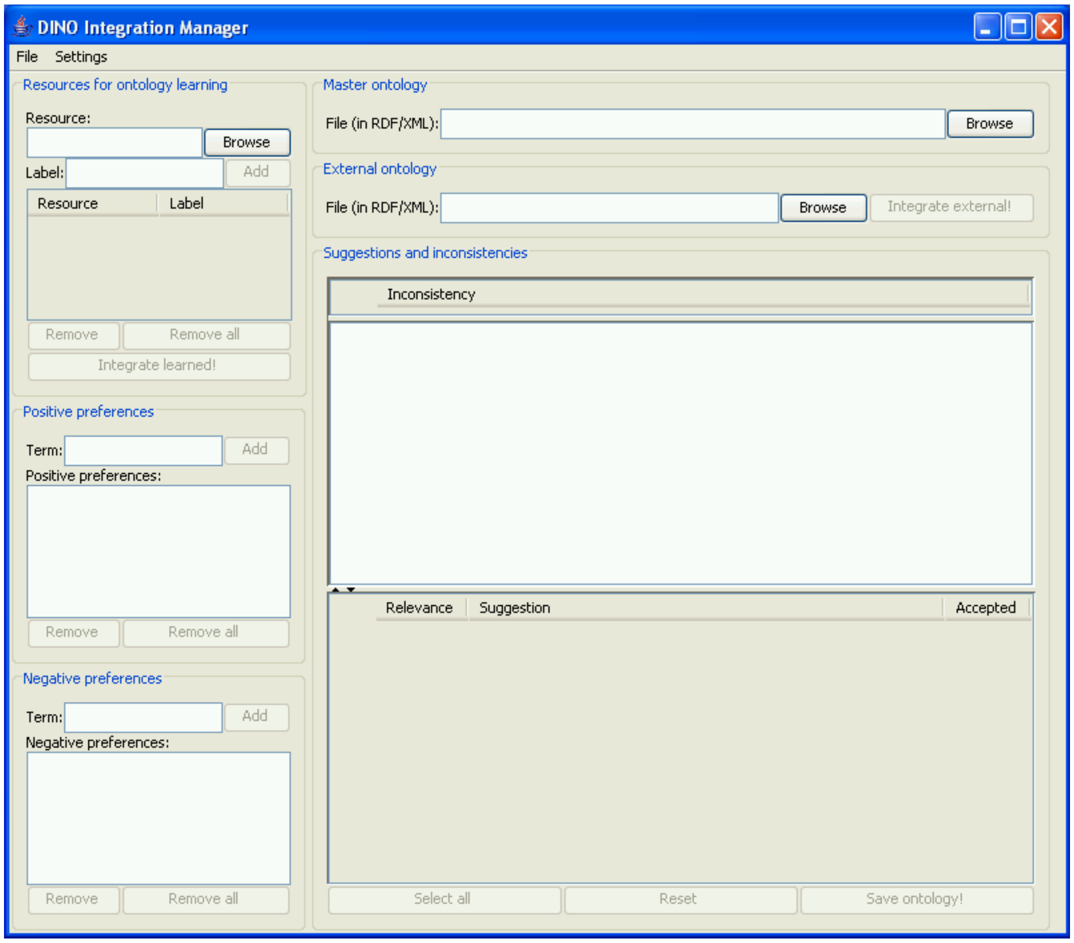


Figure 6.1: Launching the DINO interface

6.2.2 Working with DINO GUI Step by Step

In the paragraphs below, we describe typical actions performed when working with DINO GUI interface step by step. Note that only simple explanatory data are used in the given examples – for a practical use you have to pay much more attention for instance to setting the preference terms is you want to achieve reasonable results in the eventual suggestion sorting.

Launching

The DINO GUI interface after launching is displayed in Figure 6.1.

Besides the menu (its essential items are described in the following text and in Section 6.2.1), several fields are present in the interface:

- ***Resources for ontology learning*** – corpus of natural language texts can be created here; the corpus is then used for ontology learning, if a learned ontology is to be integrated

- ***Master ontology*** – the master ontology to be used within the integration can be specified and uploaded here
- ***External ontology*** – an external ontology can be specified and uploaded here; either external (if present), or learned ontology can be integrated into the master one using the DINO GUI interface – see below for details
- ***Positive preferences*** – positive preferences (i.e., the words or expressions that are preferred as labels for integrated ontology elements) can be specified here
- ***Negative preferences*** – negative preferences (i.e., the words or expressions that are unwanted as labels for integrated ontology elements) can be specified here
- ***Suggestions and inconsistencies*** – the integration output is displayed in this field; see below for details

Selecting a master ontology

If you press the *Browse* button in the ***Master ontology*** field, the file selection window pops up, as showed in Figure 6.2.

After selecting the master ontology file (in RDF/XML OWL syntax), it is ready to be uploaded as a master ontology in the integration process. You can also edit the file path directly in the respective field, as can be seen in Figure 6.3.

Creating a text corpus

If you want to integrate an ontology automatically created from natural language resources, you have to upload the respective resources (in plain text format) first. You can choose a file to be added to the corpus using the *Browse* button in the ***Resources for ontology learning*** field, as showed in Figure 6.4.

You can also specify the path to the file directly in the respective field, as can be seen in Figure 6.5.

After specifying the file to be added to the corpus, you can associate a label with it using the *Label* text field, as showed in Figure 6.6.

After pressing the *Add* button in the ***Resources for ontology learning*** field, the selected and labelled text file is added into the ontology learning corpus (see Figure 6.7). You can use the *Remove* or *Remove all* buttons in the same field in order to get rid of some or all documents, respectively, from the corpus.

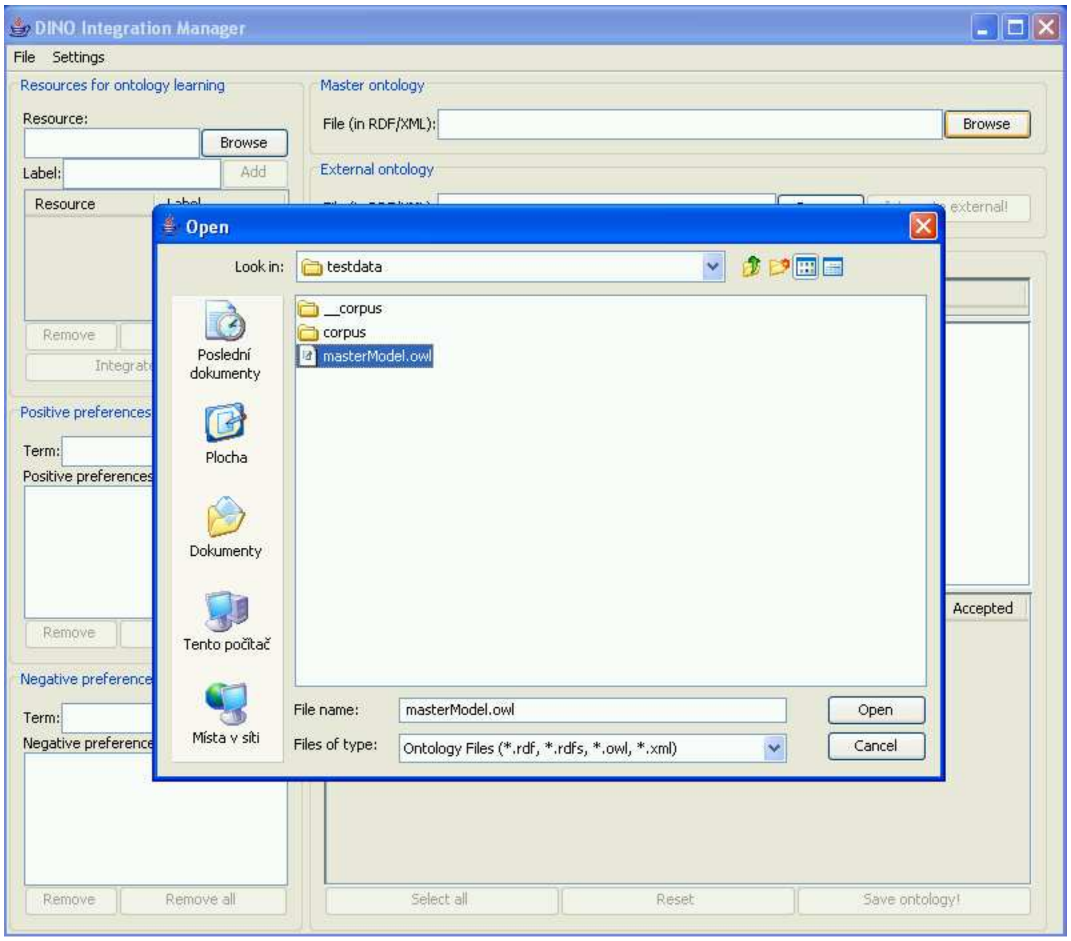
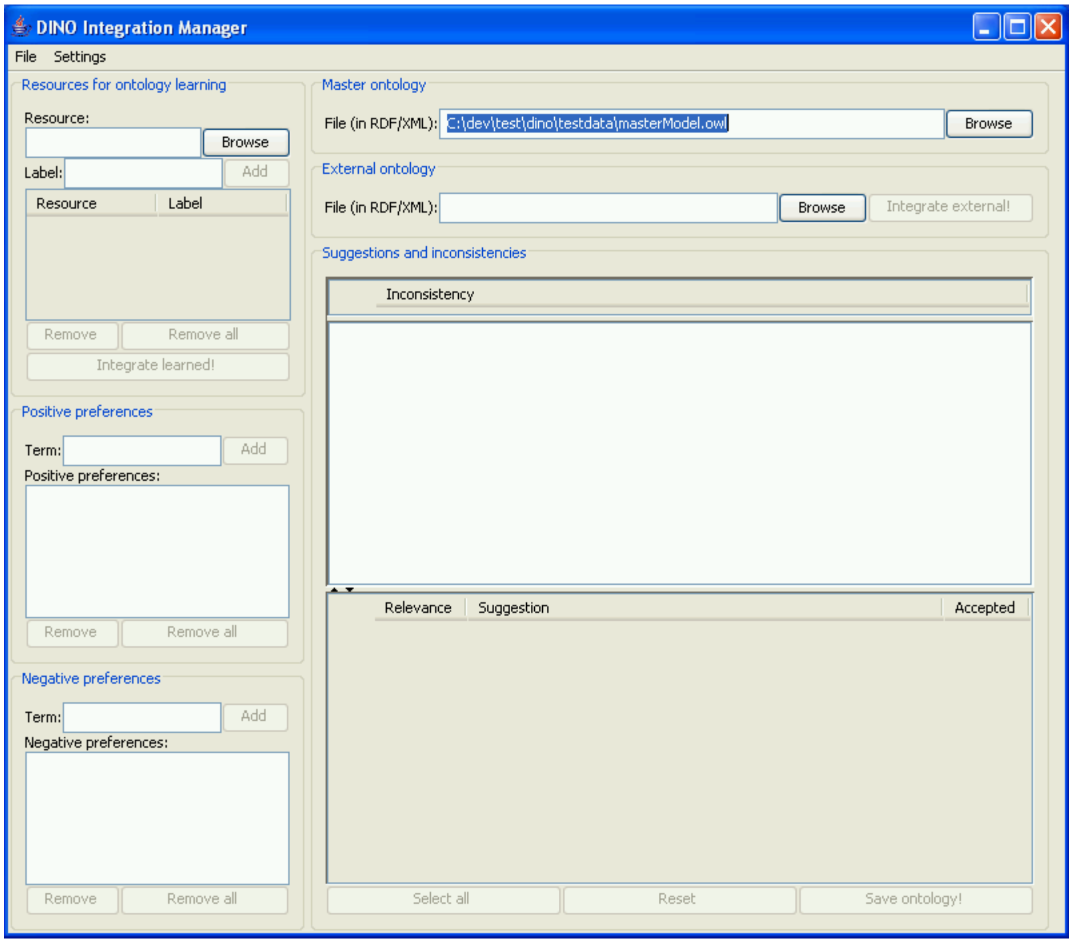
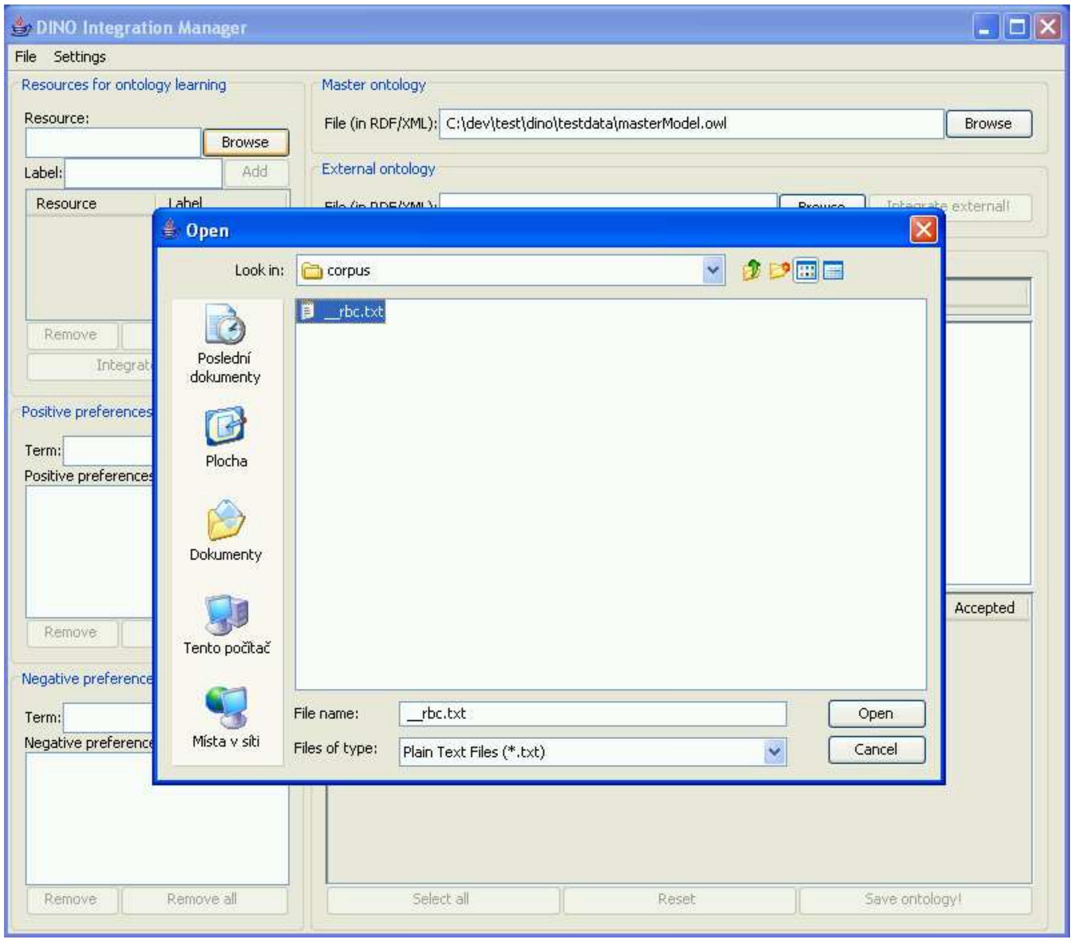


Figure 6.2: Choosing a master ontology



File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

Figure 6.3: Loading the master ontology



File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

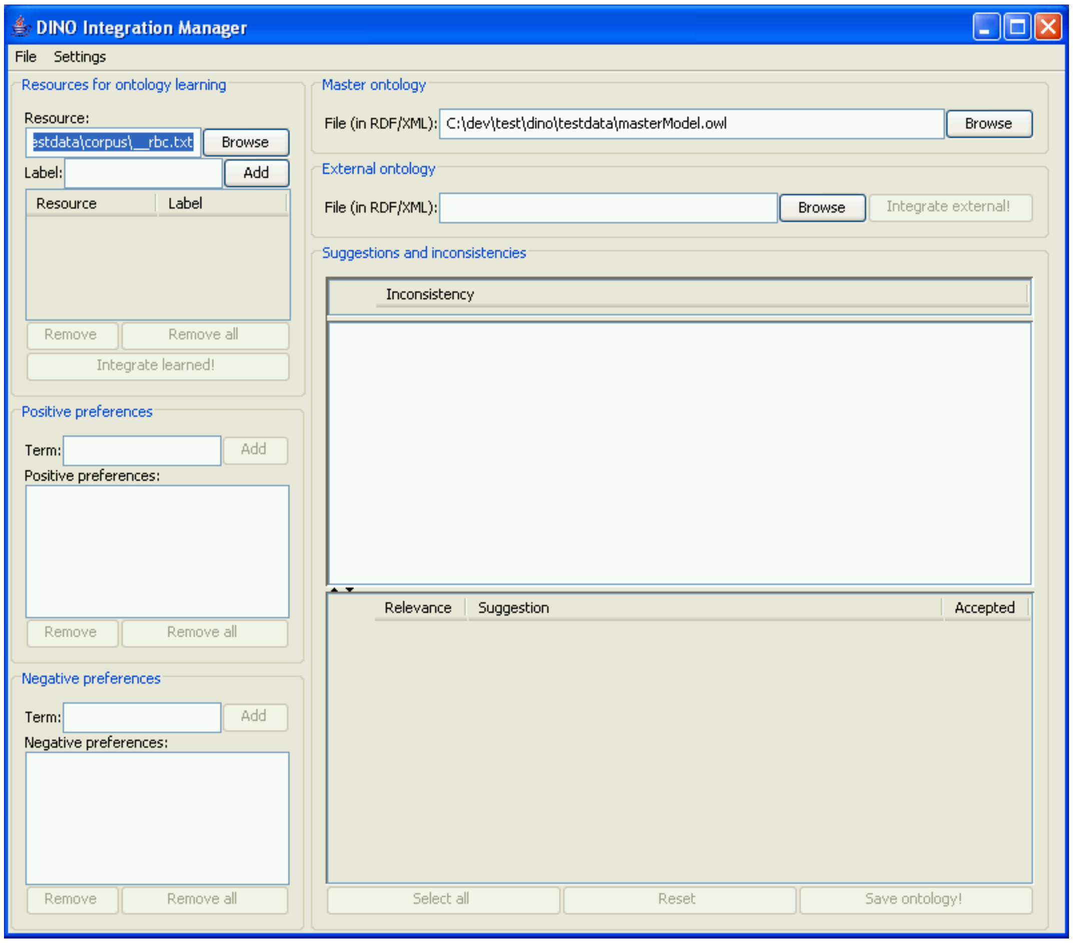
Look in: corpus

__rbc.txt

File name: __rbc.txt

Files of type: Plain Text Files (*.txt)

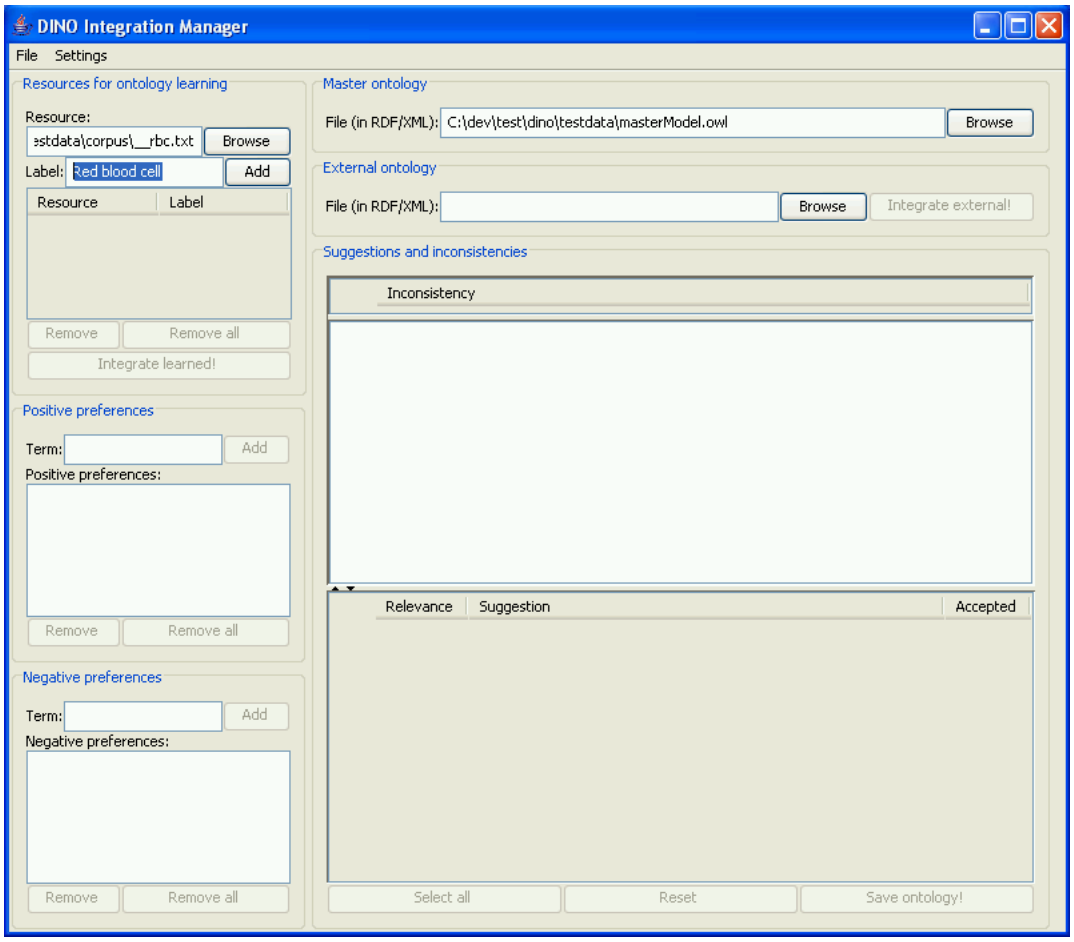
Figure 6.4: Choosing a text corpus file



Resource: estdata\corpus\__rbc.txt

File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

Figure 6.5: The text corpus file selected

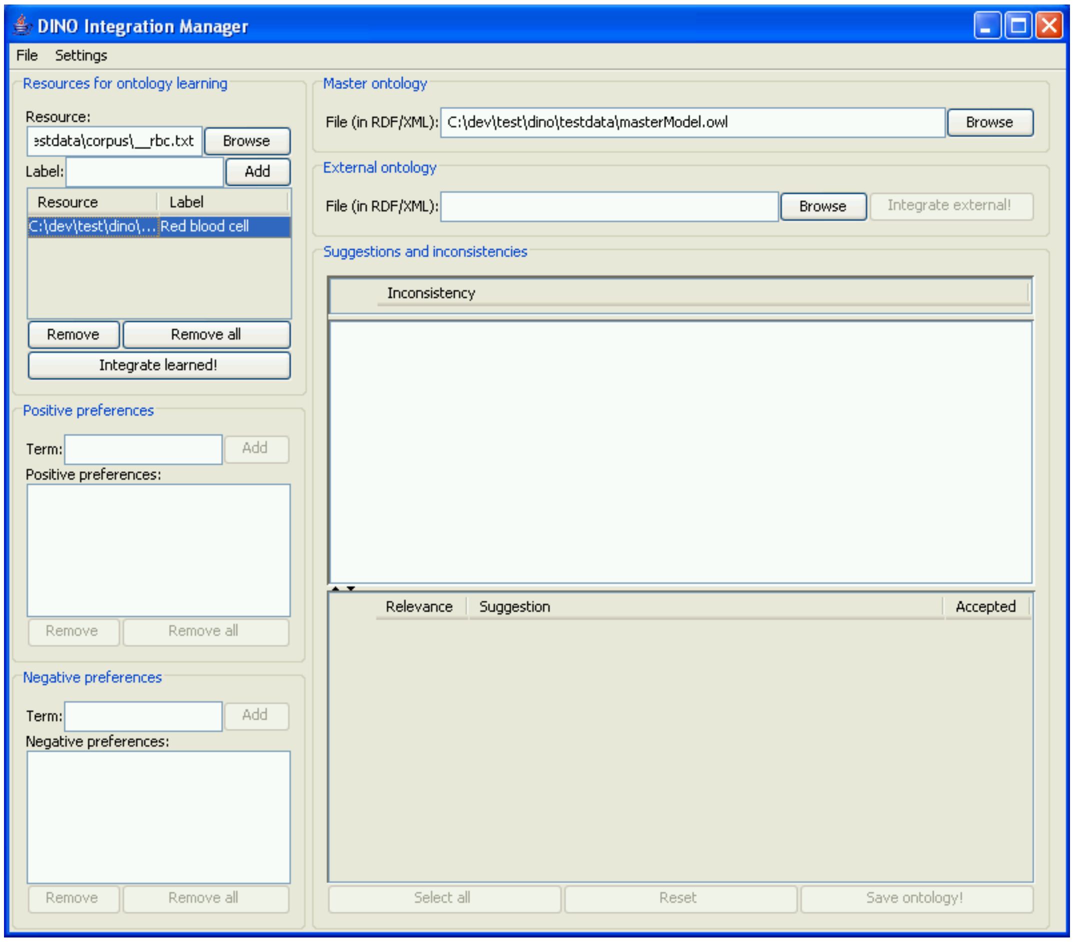


Resource: ɘstdata\corpus\__rbc.txt

Label: Red blood cell

File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

Figure 6.6: Labelling the text corpus file



Resource: ɘstdata\corpus\__rbc.txt

Resource	Label
C:\dev\test\dino\...	Red blood cell

File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

Figure 6.7: The text corpus file added

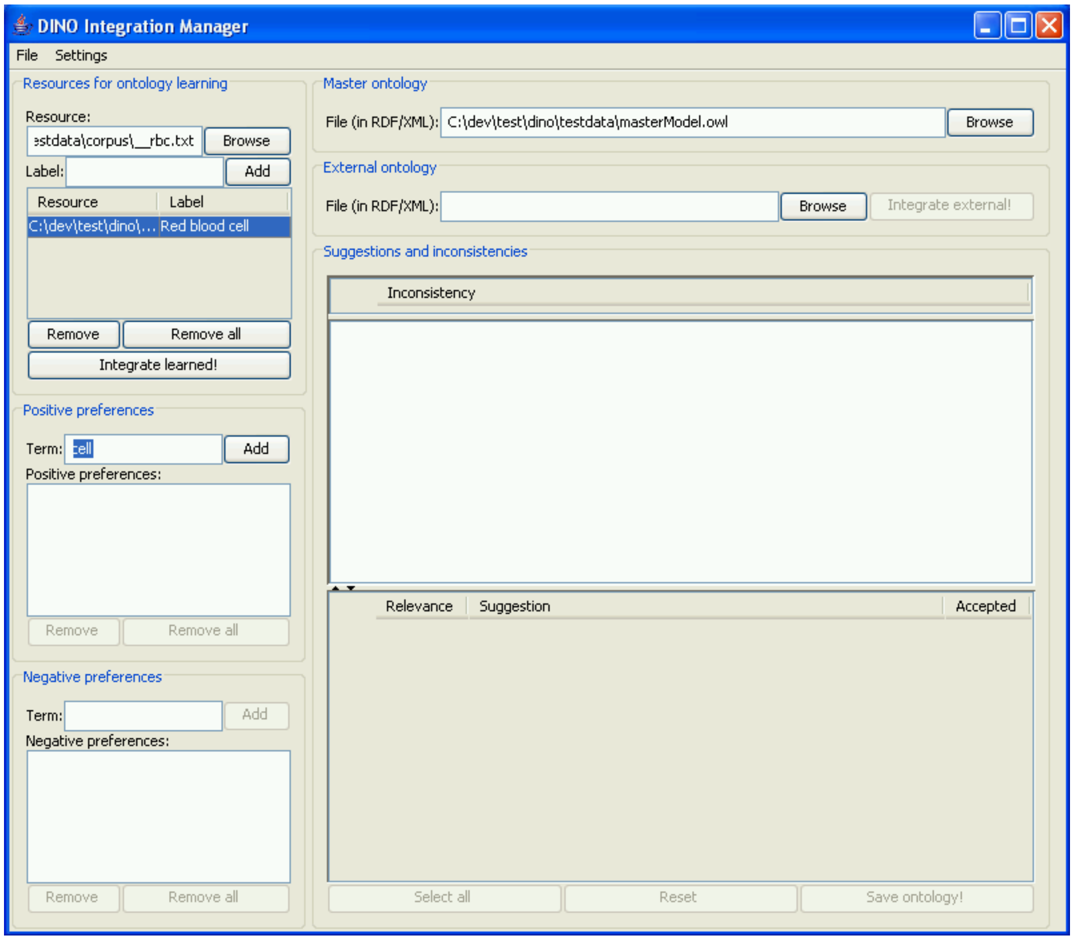


Figure 6.8: Setting preferences – typing a preferred term in

Preference settings

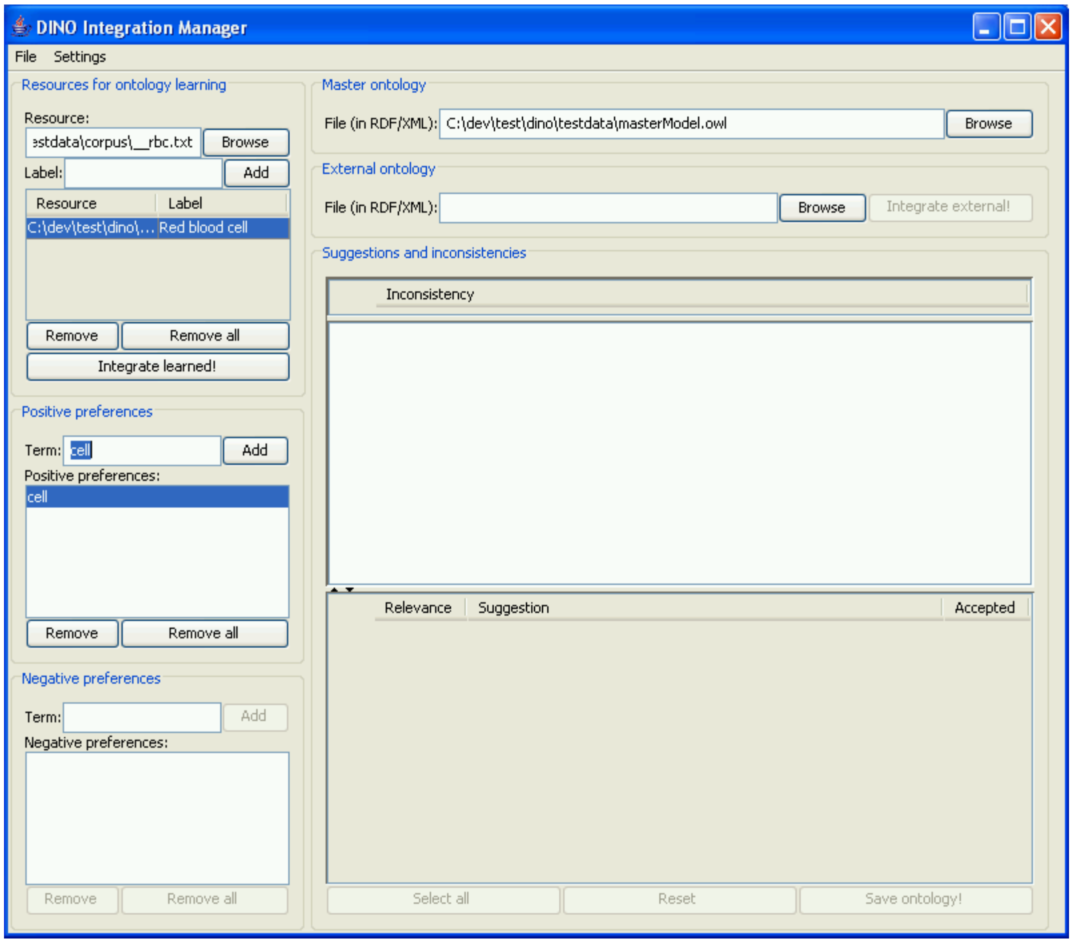
The preferred and unwanted terms (used by the suggestion sorting algorithm, see Section 3.1.5 for details) can be defined using the ***Positive preferences*** and ***Negative preferences*** fields, respectively. Figure 6.8 shows how to type in a positive preference term.

After pressing the *Add* button in the respective field, the defined preference is recorded, as can be seen in Figure 6.9. Note that exactly same procedure is to be applied when defining a negative preference, it only has to be done using the ***Negative preferences*** field.

Executing the integration

The integration can be executed in two different ways:

1. **integration of a learned ontology** – launched by pressing the *Integrate learned!* button in the ***Resources for ontology learning*** field; note that at least the master



Resource: estdata\corpus\__rbc.txt

Label:

Resource	Label
C:\dev\test\dino\...	Red blood cell

Term: cell

Positive preferences:

cell

Term:

Negative preferences:

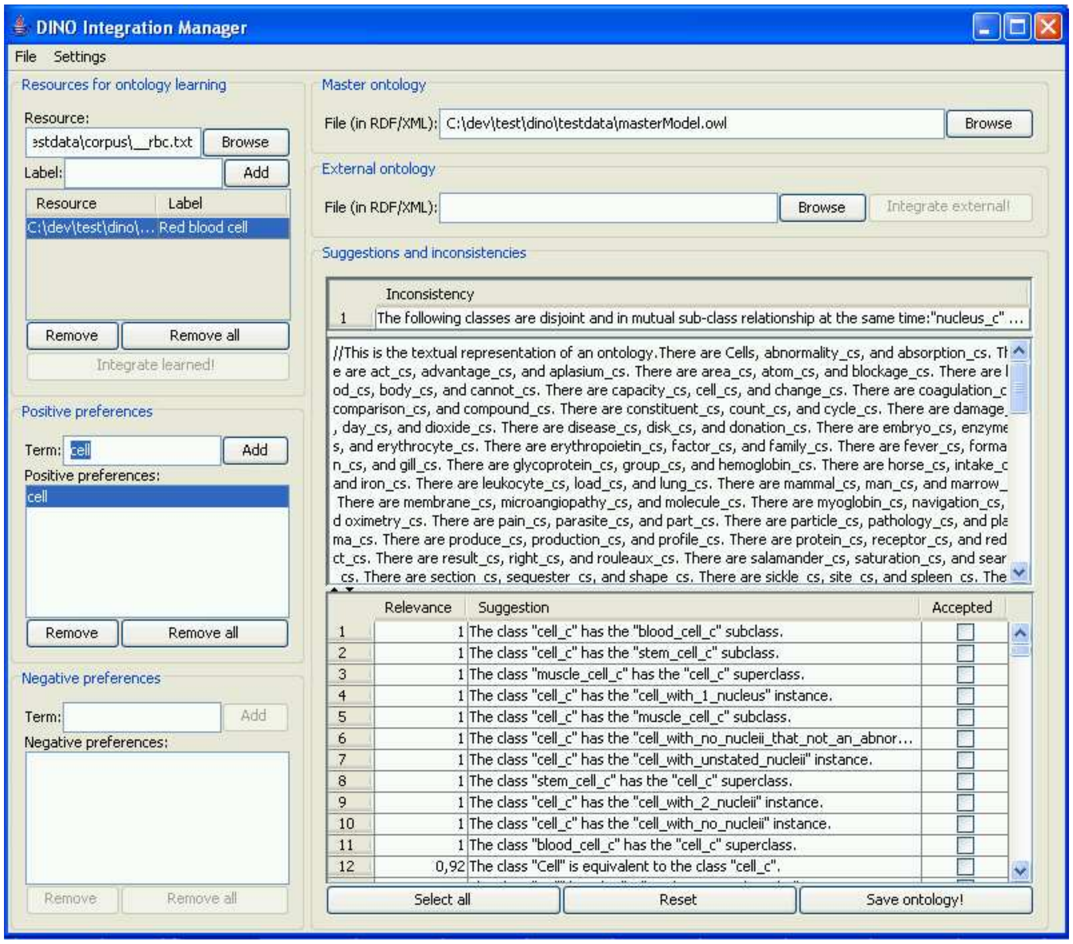
File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

File (in RDF/XML):

Inconsistency

Relevance	Suggestion	Accepted

Figure 6.9: Setting preferences – the preferred term added



Master ontology — File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

Resource	Label
C:\dev\test\dino\...	Red blood cell

Positive preferences: cell

	Inconsistency
1	The following classes are disjoint and in mutual sub-class relationship at the same time:"nucleus_c" ...

```
//This is the textual representation of an ontology.There are Cells, abnormality_cs, and absorption_cs. Th
e are act_cs, advantage_cs, and aplasium_cs. There are area_cs, atom_cs, and blockage_cs. There are l
od_cs, body_cs, and cannot_cs. There are capacity_cs, cell_cs, and change_cs. There are coagulation_c
comparison_cs, and compound_cs. There are constituent_cs, count_cs, and cycle_cs. There are damage_
, day_cs, and dioxide_cs. There are disease_cs, disk_cs, and donation_cs. There are embryo_cs, enzyme
s, and erythrocyte_cs. There are erythropoietin_cs, factor_cs, and family_cs. There are fever_cs, forma
n_cs, and gill_cs. There are glycoprotein_cs, group_cs, and hemoglobin_cs. There are horse_cs, intake_c
and iron_cs. There are leukocyte_cs, load_cs, and lung_cs. There are mammal_cs, man_cs, and marrow_
 There are membrane_cs, microangiopathy_cs, and molecule_cs. There are myoglobin_cs, navigation_cs,
d oximetry_cs. There are pain_cs, parasite_cs, and part_cs. There are particle_cs, pathology_cs, and pla
ma_cs. There are produce_cs, production_cs, and profile_cs. There are protein_cs, receptor_cs, and red
ct_cs. There are result_cs, right_cs, and rouleaux_cs. There are salamander_cs, saturation_cs, and sear
 cs. There are section_cs, sequester_cs, and shape_cs. There are sickle_cs, site_cs, and spleen_cs. The
```

	Relevance	Suggestion	Accepted
1	1	The class "cell_c" has the "blood_cell_c" subclass.	☐
2	1	The class "cell_c" has the "stem_cell_c" subclass.	☐
3	1	The class "muscle_cell_c" has the "cell_c" superclass.	☐
4	1	The class "cell_c" has the "cell_with_1_nucleus" instance.	☐
5	1	The class "cell_c" has the "muscle_cell_c" subclass.	☐
6	1	The class "cell_c" has the "cell_with_no_nucleii_that_not_an_abnor...	☐
7	1	The class "cell_c" has the "cell_with_unstated_nucleii" instance.	☐
8	1	The class "stem_cell_c" has the "cell_c" superclass.	☐
9	1	The class "cell_c" has the "cell_with_2_nucleii" instance.	☐
10	1	The class "cell_c" has the "cell_with_no_nucleii" instance.	☐
11	1	The class "blood_cell_c" has the "cell_c" superclass.	☐
12	0,92	The class "Cell" is equivalent to the class "cell_c".	☐

Figure 6.10: After launching the DINO integration

ontology has to be selected and respective corpus has to be created before you can launch this mode of integration!

2. **integration of an external ontology** – launched by pressing the *Integrate external!* button in the ***External ontology*** field; note that at least the master and external ontologies has to be selected before you can launch this mode of integration! Also note that results of integration of more complex external ontologies (e.g. containing restrictions or complex anonymous classes) are not necessarily ideal nor complete, since the current implementation is tuned in order to support rather (less complex) learned ontology integration.

Sample results of integration are displayed in Figure 6.10.

In the three parts of the ***Suggestions and inconsistencies*** field, you can see the following (from the top to the bottom):

- *detected inconsistencies* – these are resolved by default; you can check the ontology elements involved in these inconsistencies using an ontology editor later on and possibly adjust the integrated ontology concerning the inconsistencies found

- *textual representation of the addition ontology* – automatically generated natural language text, representing the statements that are to be added to the master ontology as a result of the integration process[1]

- *sorted suggestions* – the main DINO integration output; the suggestions are presented in natural language, sorted according to their lexical similarity to the set of defined preferences and associated with the underlying ontology axioms – you can browse and process them in order to generate the final integrated ontology, as described in the following paragraph

After the integration

A suggestion can be accepted by ticking the respective box, as displayed in Figure 6.11.

You can also use the *Select all* or *Reset* buttons in the ***Suggestions and inconsistencies*** field in order to select or de-select all suggestion, respectively. After selecting all accepted suggestions, you can eventually save the integrated ontology using the *Save ontology!* button. When pressing this button, the axioms corresponding to the accepted suggestions are included into the former master ontology model and a file-save window pops up, as showed in Figure 6.12.

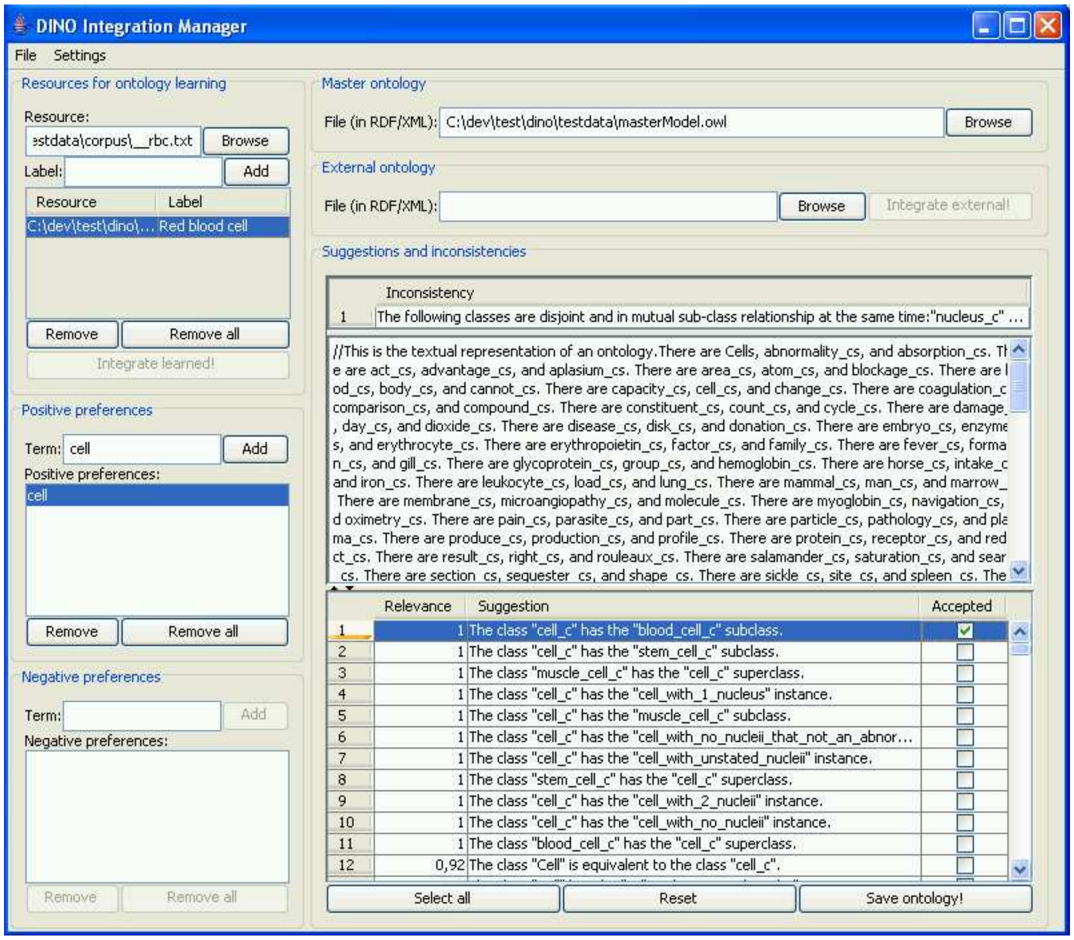
You can select the file which the integrated ontology will be saved into either using the *Browse* button in the file-save window (see Figure 6.13), or by typing the respective path directly into the appropriate field (see Figure 6.14).

The ontology is saved in the selected location by pressing the *Save* button in the file-save dialog window, as showed in Figure 6.14. Note that the ontology is saved in RDF/XML OWL syntax.

6.3 DINO API

You can download the DINO API at `http://smile.deri.ie/resources/2007/dino/download.html`. Note that the API version 0.1 is a public alpha testing version, not intended for production use as such. The most current JavaDoc API documentation is available at `http://smile.deri.ie/resources/2007/dino/documentation/`.

[1]Note that implementation of appropriate post-processing of rather distracting form of this output is currently in progress as one of the major DINO improvements planned for the near future.



Resource: ɘstdata\corpus\__rbc.txt

Resource	Label
C:\dev\test\dino\...	Red blood cell

Term: cell

Positive preferences: cell

Master ontology — File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

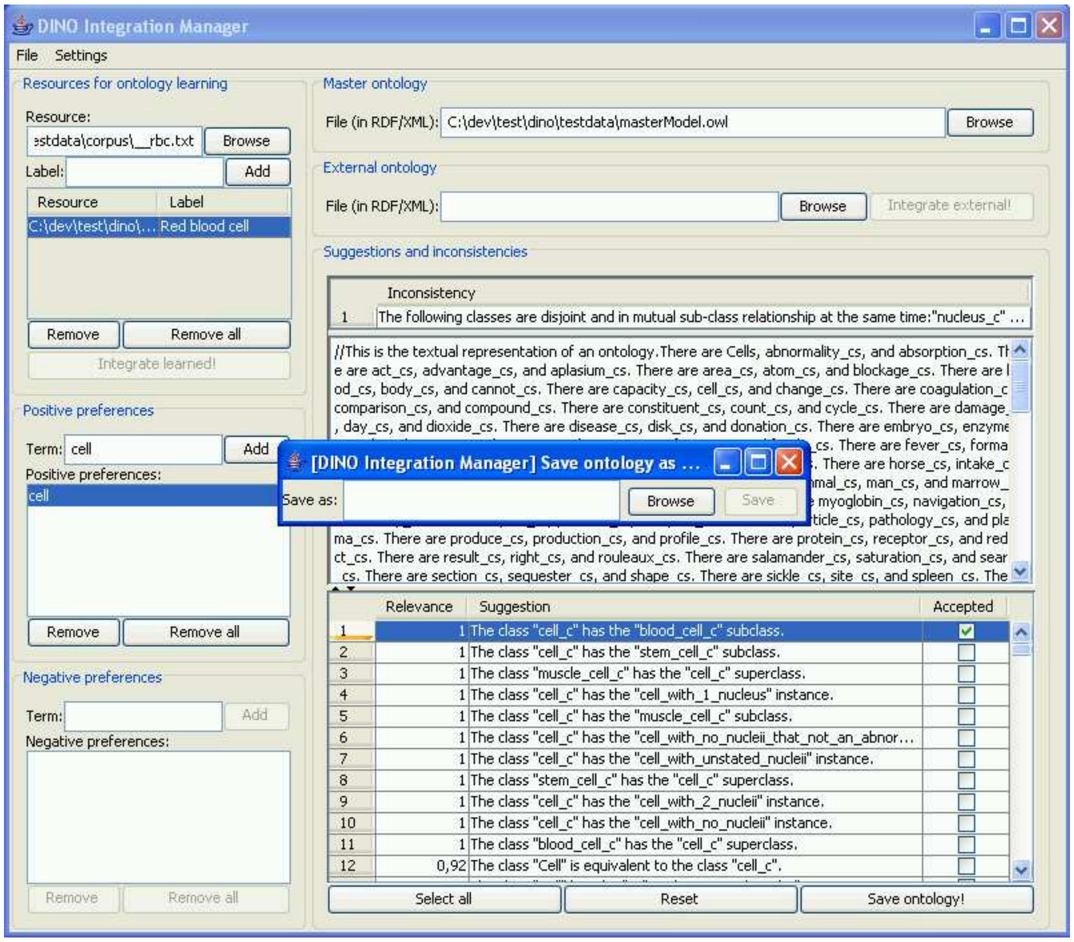
External ontology — File (in RDF/XML):

	Inconsistency
1	The following classes are disjoint and in mutual sub-class relationship at the same time:"nucleus_c" ...

```
//This is the textual representation of an ontology.There are Cells, abnormality_cs, and absorption_cs. Th
e are act_cs, advantage_cs, and aplasium_cs. There are area_cs, atom_cs, and blockage_cs. There are l
od_cs, body_cs, and cannot_cs. There are capacity_cs, cell_cs, and change_cs. There are coagulation_c
comparison_cs, and compound_cs. There are constituent_cs, count_cs, and cycle_cs. There are damage_
, day_cs, and dioxide_cs. There are disease_cs, disk_cs, and donation_cs. There are embryo_cs, enzyme
s, and erythrocyte_cs. There are erythropoietin_cs, factor_cs, and family_cs. There are fever_cs, forma
n_cs, and gill_cs. There are glycoprotein_cs, group_cs, and hemoglobin_cs. There are horse_cs, intake_c
and iron_cs. There are leukocyte_cs, load_cs, and lung_cs. There are mammal_cs, man_cs, and marrow_
 There are membrane_cs, microangiopathy_cs, and molecule_cs. There are myoglobin_cs, navigation_cs,
d oximetry_cs. There are pain_cs, parasite_cs, and part_cs. There are particle_cs, pathology_cs, and pla
ma_cs. There are produce_cs, production_cs, and profile_cs. There are protein_cs, receptor_cs, and red
ct_cs. There are result_cs, right_cs, and rouleaux_cs. There are salamander_cs, saturation_cs, and sear
 cs. There are section cs, sequester cs, and shape cs. There are sickle cs, site cs, and spleen cs. The
```

	Relevance	Suggestion	Accepted
1	1	The class "cell_c" has the "blood_cell_c" subclass.	☑
2	1	The class "cell_c" has the "stem_cell_c" subclass.	☐
3	1	The class "muscle_cell_c" has the "cell_c" superclass.	☐
4	1	The class "cell_c" has the "cell_with_1_nucleus" instance.	☐
5	1	The class "cell_c" has the "muscle_cell_c" subclass.	☐
6	1	The class "cell_c" has the "cell_with_no_nucleii_that_not_an_abnor...	☐
7	1	The class "cell_c" has the "cell_with_unstated_nucleii" instance.	☐
8	1	The class "stem_cell_c" has the "cell_c" superclass.	☐
9	1	The class "cell_c" has the "cell_with_2_nucleii" instance.	☐
10	1	The class "cell_c" has the "cell_with_no_nucleii" instance.	☐
11	1	The class "blood_cell_c" has the "cell_c" superclass.	☐
12	0,92	The class "Cell" is equivalent to the class "cell_c".	☐

Figure 6.11: Accepting a suggestion



Resource: ɘstdata\corpus\__rbc.txt

Resource	Label
C:\dev\test\dino\...	Red blood cell

Term: cell

Positive preferences:
cell

Term:

Negative preferences:

Master ontology — File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

External ontology — File (in RDF/XML):

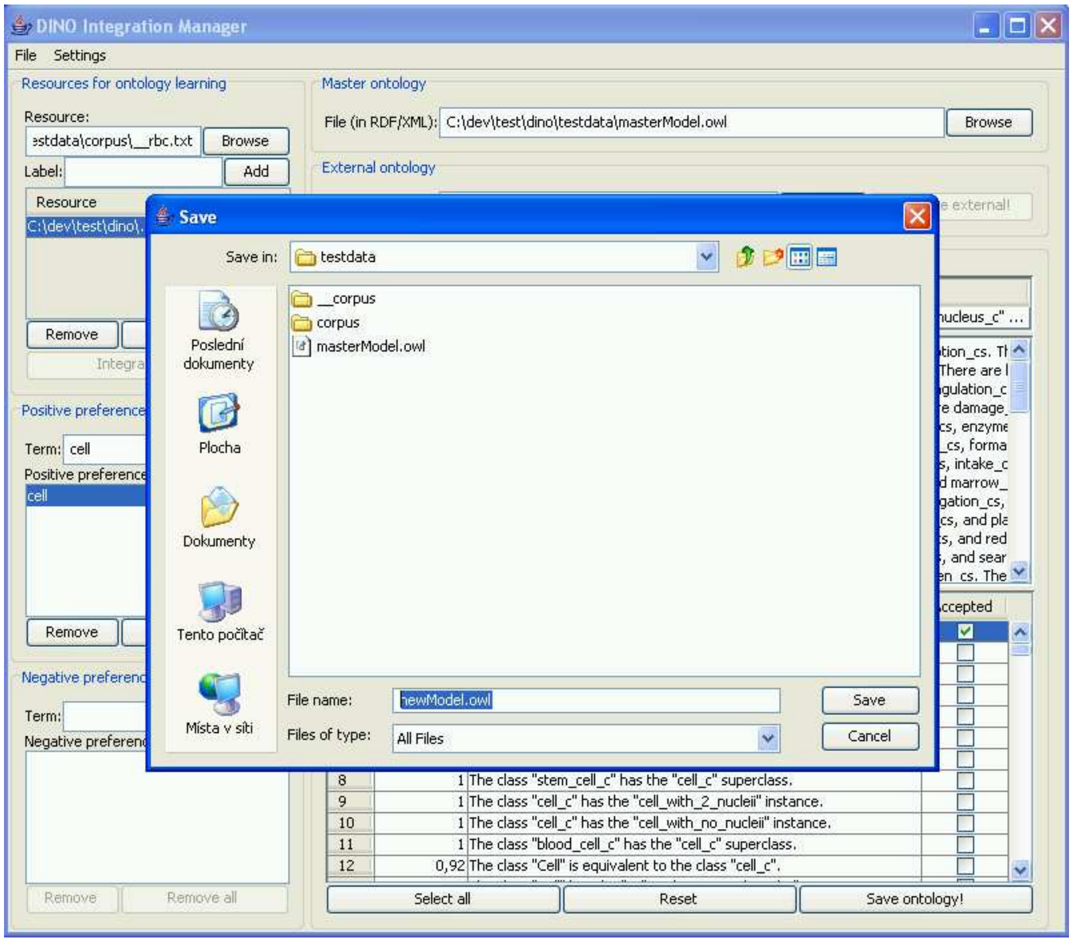
Save as:

	Inconsistency
1	The following classes are disjoint and in mutual sub-class relationship at the same time:"nucleus_c" ...

```
//This is the textual representation of an ontology.There are Cells, abnormality_cs, and absorption_cs. Th
e are act_cs, advantage_cs, and aplasium_cs. There are area_cs, atom_cs, and blockage_cs. There are l
od_cs, body_cs, and cannot_cs. There are capacity_cs, cell_cs, and change_cs. There are coagulation_c
comparison_cs, and compound_cs. There are constituent_cs, count_cs, and cycle_cs. There are damage_
, day_cs, and dioxide_cs. There are disease_cs, disk_cs, and donation_cs. There are embryo_cs, enzyme
_cs. There are fever_cs, forma
. There are horse_cs, intake_c
mal_cs, man_cs, and marrow_
myoglobin_cs, navigation_cs,
ticle_cs, pathology_cs, and pla
ma_cs. There are produce_cs, production_cs, and profile_cs. There are protein_cs, receptor_cs, and red
ct_cs. There are result_cs, right_cs, and rouleaux_cs. There are salamander_cs, saturation_cs, and sear
_cs. There are section_cs, sequester_cs, and shape_cs. There are sickle_cs, site_cs, and spleen_cs. The
```

	Relevance	Suggestion	Accepted
1	1	The class "cell_c" has the "blood_cell_c" subclass.	✓
2	1	The class "cell_c" has the "stem_cell_c" subclass.	☐
3	1	The class "muscle_cell_c" has the "cell_c" superclass.	☐
4	1	The class "cell_c" has the "cell_with_1_nucleus" instance.	☐
5	1	The class "cell_c" has the "muscle_cell_c" subclass.	☐
6	1	The class "cell_c" has the "cell_with_no_nucleii_that_not_an_abnor...	☐
7	1	The class "cell_c" has the "cell_with_unstated_nucleii" instance.	☐
8	1	The class "stem_cell_c" has the "cell_c" superclass.	☐
9	1	The class "cell_c" has the "cell_with_2_nucleii" instance.	☐
10	1	The class "cell_c" has the "cell_with_no_nucleii" instance.	☐
11	1	The class "blood_cell_c" has the "cell_c" superclass.	☐
12	0,92	The class "Cell" is equivalent to the class "cell_c".	☐

Figure 6.12: Saving the updated master ontology – step 1



Resource: ›stdata\corpus\__rbc.txt

Label:

Resource: C:\dev\test\dino\.

Term: cell

Positive preference: cell

Term:

File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

Save in: testdata

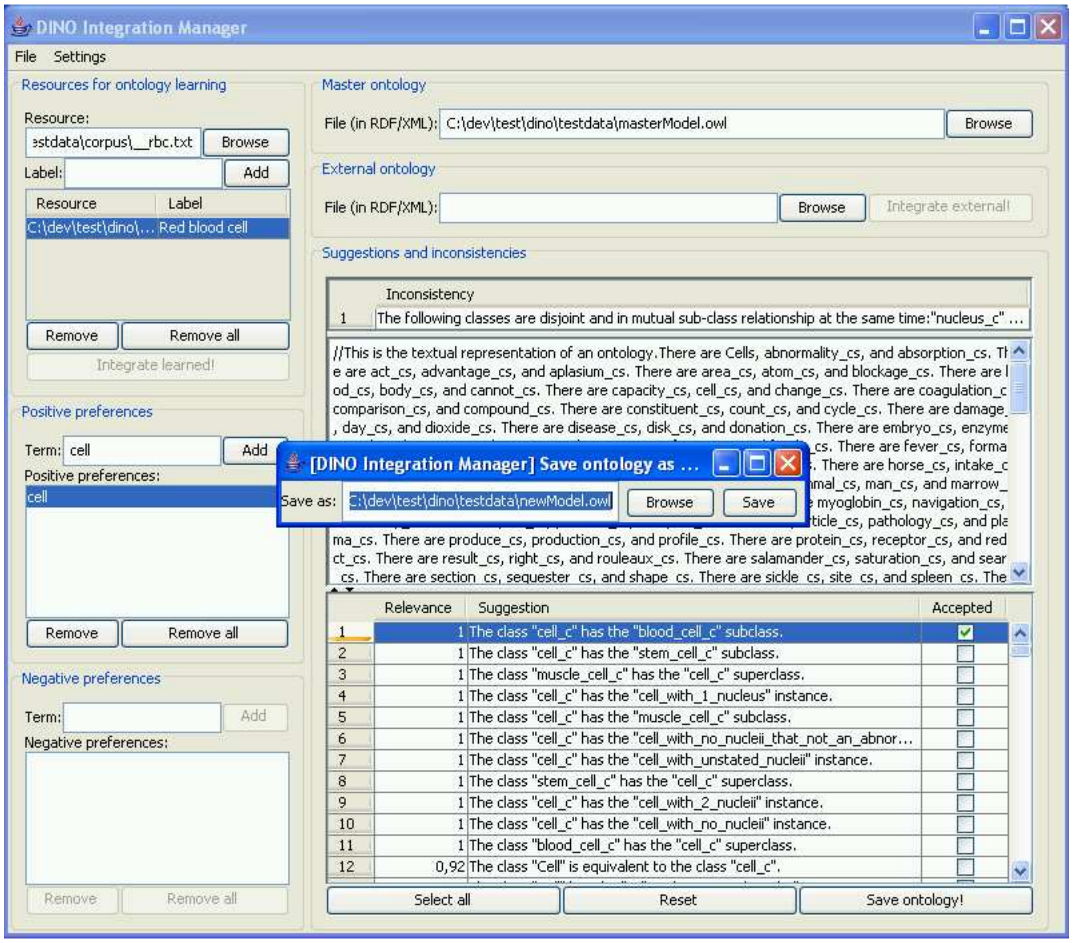
__corpus
corpus
masterModel.owl

File name: newModel.owl

Files of type: All Files

8	1	The class "stem_cell_c" has the "cell_c" superclass.	☐
9	1	The class "cell_c" has the "cell_with_2_nuclei" instance.	☐
10	1	The class "cell_c" has the "cell_with_no_nuclei" instance.	☐
11	1	The class "blood_cell_c" has the "cell_c" superclass.	☐
12	0,92	The class "Cell" is equivalent to the class "cell_c".	☐

Figure 6.13: Saving the updated master ontology – step 2



Resource: ɘstdata\corpus\__rbc.txt

Label:

Resource	Label
C:\dev\test\dino\...	Red blood cell

Term: cell

Positive preferences: cell

Term:

Negative preferences:

File (in RDF/XML): C:\dev\test\dino\testdata\masterModel.owl

File (in RDF/XML):

	Inconsistency
1	The following classes are disjoint and in mutual sub-class relationship at the same time:"nucleus_c" ...

```
//This is the textual representation of an ontology.There are Cells, abnormality_cs, and absorption_cs. Th
e are act_cs, advantage_cs, and aplasium_cs. There are area_cs, atom_cs, and blockage_cs. There are l
od_cs, body_cs, and cannot_cs. There are capacity_cs, cell_cs, and change_cs. There are coagulation_c
comparison_cs, and compound_cs. There are constituent_cs, count_cs, and cycle_cs. There are damage_
, day_cs, and dioxide_cs. There are disease_cs, disk_cs, and donation_cs. There are embryo_cs, enzyme
cs. There are fever_cs, forma
. There are horse_cs, intake_c
mal_cs, man_cs, and marrow_
myoglobin_cs, navigation_cs,
ticle_cs, pathology_cs, and pla
ma_cs. There are produce_cs, production_cs, and profile_cs. There are protein_cs, receptor_cs, and red
ct_cs. There are result_cs, right_cs, and rouleaux_cs. There are salamander_cs, saturation_cs, and sear
_cs. There are section_cs, sequester_cs, and shape_cs. There are sickle_cs, site_cs, and spleen_cs. The
```

Save as: C:\dev\test\dino\testdata\newModel.owl

	Relevance	Suggestion	Accepted
1	1	The class "cell_c" has the "blood_cell_c" subclass.	✓
2	1	The class "cell_c" has the "stem_cell_c" subclass.	☐
3	1	The class "muscle_cell_c" has the "cell_c" superclass.	☐
4	1	The class "cell_c" has the "cell_with_1_nucleus" instance.	☐
5	1	The class "cell_c" has the "muscle_cell_c" subclass.	☐
6	1	The class "cell_c" has the "cell_with_no_nucleii_that_not_an_abnor...	☐
7	1	The class "cell_c" has the "cell_with_unstated_nucleii" instance.	☐
8	1	The class "stem_cell_c" has the "cell_c" superclass.	☐
9	1	The class "cell_c" has the "cell_with_2_nucleii" instance.	☐
10	1	The class "cell_c" has the "cell_with_no_nucleii" instance.	☐
11	1	The class "blood_cell_c" has the "cell_c" superclass.	☐
12	0,92	The class "Cell" is equivalent to the class "cell_c".	☐

Figure 6.14: Saving the updated master ontology – step 3

6.3.1 Notes on Installation

After downloading the DINO API package, extract its content into a directory on your machine (this directory is referred to as `DINO_API_HOME` in the following text). Include the source files in the `DINO_API_HOME/src` directory into the build path of the project that is going to use the DINO integration library. The necessary libraries should be included in the `DINO_API_HOME/lib` directory – these are needed to be imported as well. In case a library is missing (possible in case of the 0.1 version package; usually indicated by NoClassDefFound exception thrown when executing a part of the DINO integration library code), please report to `vit.novacek@deri.org`, preferably with the exception transcript attached - we will provide you with the needed library missing in this tentative alpha distribution.

6.3.2 Executing the Integration

In order to use the ontology integration technique implemented by the DINO integration library, one needs to create a `DINOIntegration` object. See the JavaDoc documentation in the `DINO_API_HOME/doc` directory on how to configure the parameters and set the input resources within the constructor and possibly consequent set-methods calls.

In general, the DINOIntegration object creation and preferred/unwanted words setting is only needed before the integration can be executed – see the following example of typical usage:

```
...

DINOIntegration comm = new DINOIntegration(corpURI,mOnto,ba-
se,GATE_HOME);
comm.setTMP(tmpPath);
comm.setAdditionOntPath(additionPath);
comm.setPrefLabels(p);
comm.setNonPrefLabels(n);
SuggestionSeq ts = comm.integrate();
TreeMap suggestions = ts.getSuggestions();
HashSet incon = ts.getInconsistencies();

process(suggestions, incon); // custom processing

...
```

The meaning of the variables in the above code sample is as follows:

- `corpURI` – a URI path to the files forming a corpus which the ontology to be integrated shall be learned from

- `mOnto` – a path to the 'master' ontology to which the learned ontology will be integrated (OWL format supported)
- `base` – base URI to be set for the learned ontology
- `GATE_HOME` – path to the local GATE installation home directory
- `tmpPath` – path to the temporary directory (to store the temporary files created during the integration)
- `additionPath` – path to the persistent addition ontology model (will be created)
- `p` – collection of preferred terms, i.e. the terms you would prefer to be included - overall relevance of the integration results will be computed according to the lexical similarity of learned entities to the terms defined here
- `n` – collection of preferred terms, i.e. the terms you would not like to be included - overall relevance of the integration results will be computed according to the lexical dissimilarity of learned entities to the terms defined here
- `suggestions` – object containing human-readable suggestions on master ontology extension by entities from the learned ontology - the result of the integration - sorted by their relevance
- `incon` – object containing a set of inconsistencies possibly introduced by the learned ontology integration (resolved automatically by default)

6.3.3 Processing the Results of the Integration

The type `SuggestionSeq` has a `getOntologyText()` method, returning `String`, that can be used to get textual representation of the whole addition model resulting from the integration process.

The method `getInconsistencies()` returns a `HashSet` with elements of type `GenericInconsistency` (see the `rwrap` package in the JavaDoc of DINO API). This type has a `getNLRepr()` method, returning a `String` with textual representation of the respective inconsistency you can further process.

The method `getSuggestions()` returns a `TreeMap` - sorted structure with keys representing the (float type) relevance of the suggestion stored as a respective value. The value has a type `GenericSuggestion` (see the `iface` package in the JavaDoc of DINO API). You can use the `getText()` method of the `GenericSuggesgtion` type in order to get a (`String`) textual representation of the respective suggestion.

Any other details on the relevant types and methods can be found in the DINO API JavaDoc – available either in the `DINO_API_HOME/doc` directory, or at `http://smile.deri.ie/resources/2007/dino/documentation/`(if you are using the most recent API version).

Chapter 7

Conclusions and Future Work

Here we conclude the report in Section 7.1. As a part of the conclusion, we emphasise essential relation between the DINO integration framework and implementation of the dynamic lifecycle scenario we coined in [NHL$^+$06]. Section 7.2 presents an overview of the future work on the ontology dynamics topics.

7.1 Conclusions

We presented the basic principles of DINO – a novel framework for ontology development in dynamic and data-intensive domains (e.g., e-health or biomedicine). As a core contribution of the report, we described the mechanism of integration of learned and manually maintained knowledge. It covers all the requirements specified in Section 1.1. The proposed combination of automatic and manual knowledge acquisition principles, integration and inconsistency resolution ensures production and maintenance of reliable, broad and precise ontologies when using DINO in dynamic settings. The analysis of factual needs in the medicine application domains presented in Chapter 4 has shown that the proposed method we have prototyped is relevant for the contemporary industry needs (namely in the biomedical research and clinical practice). We presented and analysed initial results of practical application of DINO integration technique in Chapter 5, reporting on promising features of the approach. The following section elaborates the relations between the DINO integration and the dynamic ontology lifecycle we introduced in the previous version of this report [NHL$^+$06].

7.1.1 DINO Integration and DINO Lifecycle

The DINO integration does not provide a full implementation of the dynamic ontology lifecycle scenario features proposed in [NHL$^+$06]. However, in the following we show, that it definitely implements its substantial part and allows a user to follow the scenario, indeed, if he or she combines the DINO integration platform with an external tool for (collaborative) ontology maintenance.

Recalling Figure 2.1 in Chapter 2, we can go phase by phase and decide whether it is implemented by the DINO integration platform or not:

- *creation component/ontology learning* – covered by the respective wrapper described in Section 3.1.1
- *creation component/collaborative ontology development* – not covered by DINO integration, however, users can benefit from using external state of the art applications for this task and uploading the master ontology maintained within this component into DINO; Protégé [GMF$^+$03] can serve very well as such an application, since it supports both standalone and collaborative ontology development [TN07]
- *evaluation* – (1), evaluation of the ontology learning results is performed by users when accepting or discarding suggestions for integration (see Section 3.1.5 and Section 6.2.2); (2), evaluation in the collaborative ontology development lifecycle sub-component can be done by users involved in the ontology development process, possibly using for instance methods described in [HSG$^+$05]
- *versioning* – versioning can be tackled using the SemVersion system [VG06] (see also Knowledge Web deliverables [VEK$^+$05, VKZ$^+$05]); when using Protégé for the manual ontology development, users can employ respective SemVersion plug-in [GVH06] that has recently been extended in order to support Protégé-OWL interface
- *negotiation* – this component is implemented by the DINO integration and can be used on both places in the lifecycle scheme (however, it may be incomplete for complex ontologies in the current prototype implementation)

As we can see, the applications we presented here already allow for application of all the lifecycle scenario features proposed in [NHL$^+$06], even though we are still much rather in a research prototype stage.

7.2 Future Work

The main portion of the future work consists of several points. First, the integration process should be made more scalable. The inconsistency resolution mechanism should be more transparent and user-centric (e.g., an interface for editing user-defined consistency restrictions and their consequent application in the integration process would be desirable). The set of ontology constructs supported in the integration process should be extended in order to fully cover more complex non-learned ontologies. The last but not least concerning the DINO implementation, the natural language component output should be improved in order to increase its smooth and non-distracting readability.

Further studies on the theoretical features of the integration process should be performed. This is relevant mainly in the scope of the custom-defined inconsistency restrictions and their relation to logical ontology inconsistency. Deeper studies on conformance to the ontology change operators of formal diff structures defined in [NHA+07]) would be interesting, too.

The DINO framework could also be directly incorporated into the Protégé ontology engineering platform, since it is the most widely used tool among some of the key players in the Semantic Web community (see Appendix A, Section A.2.1). Such a closer integration with a complex ontology engineering tool would certainly facilitate the dynamic ontology development process even more, thus presenting an implementation of the whole lifecycle scenario introduced in [NHL+06] within one coherent application.

Besides improvements of the implementation, we plan to continuously evaluate the framework and elicit feedback among broader expert community involved. Consequently, DINO should further be improved it in line with demands of interested industrial partners (primarily, but not only within the presented e-health and biomedicine application domains).

Bibliography

[AHS05] Ahmed Alasoud, Volker Haarslev, and Nematollaah Shiri. A hybrid approach for ontology integration. In *Proceedings of the 31st VLDB Conference*. Very Large Data Base Endowment, 2005.

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Decription Logic Handbook: Theory, implementation, and applications*. Cambridge University Press, Cambridge, USA, 2003.

[BG04] Dan Brickley and R. V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*, 2004. Available at (February 2006): `http://www.w3.org/TR/rdf-schema/`.

[BvHH$^+$04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *OWL Web Ontology Language Reference*, 2004. Available at (February 2006): `http://www.w3.org/TR/owl-ref/`.

[CBW02] F. Ciravegna C. Brewster and Y. Wilks. User-centred onlology learning for knowledge management. In *In Proceedings 7th International Workshop on Applications of Natural Language to Information Systems, Stockholm.*, 2002.

[CGL01] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. A framework for ontology integration. In *In Proc. of the First Semantic Web Working Symposium*. Springer-Verlag, 2001.

[CLCGP06] O. Corcho, A. Lopez-Cima, and A. Gomez-Perez. The ODESeW 2.0 semantic web application framework. In *Proceedings of WWW 2006*, pages 1049–1050, New York, 2006. ACM Press.

[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*, 2002.

[CV05] Philipp Cimiano and Johanna Völker. Text2Onto - a framework for ontology learning and data-driven change discovery. In *Proceedings of the NLDB 2005 Conference*, pages 227–238. Springer-Verlag, 2005.

[DKMR$^+$06] Rose Dieng-Kuntz, David Minier, Marek Ruzicka, Frederic Corby, Olivier Corby, and Laurent Alamarguy. Building and using a medical ontology for knowledge management and cooperative work in a health care network. *Computers in Biology and Medicine*, 36:871–892, 2006.

[DP06] S. M. Deen and K. Ponnamperuma. Dynamic ontology integration in a multi-agent environment. In *Proceedings of AINA '06*. IEEE Computer Society, 2006.

[DS06] K. Dellschaft and S. Staab. On how to perform a gold standard based evaluation of ontology learning. In *Proceedings of the International Semantic Web Conference. Athens, GA, USA.*, 2006.

[Eic06] Marco Eichelberg. Requirements analysis for the ride roadmap. Deliverable D2.1.1, RIDE, 2006.

[ELTV04] Jérome Euzenat, David Loup, Mohamed Touzani, and Petko Valtchev. Ontology alignment with ola. In *Proceedings of the 3rd International Workshop on Evaluation of Ontology based Tools (EON)*, Hiroshima, Japan, 2004. CEUR-WS.

[Euz04] J. Euzenat. An API for ontology alignment. In *ISWC 2004: Third International Semantic Web Conference. Proceedings*, pages 698–712. Springer-Verlag, 2004.

[FLGPJ97] M. Fernandez-Lopez, A. Gomez-Perez, and N. Juristo. Methontology: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, pages 33–40, Stanford, USA, March 1997.

[FLGPR00] M. Fernandez-Lopez, A. Gomez-Perez, and M. D. Rojas. Ontologies' crossed life cycles. In *Proceedings of International Conference in Knowledge Engineering and Management*, pages 65–79. Springer–Verlag, 2000.

[GMF$^+$03] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubezy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human–Computer Studies*, 58(1):89–123, 2003.

[GPFLC04] A. Gomez-Perez, M. Fernandez-Lopez, and O. Corcho. *Ontological Engineering*. Advanced Information and Knowledge Processing. Springer-Verlag, 2004.

[GVH06] T. Groze, M. Völkel, and S. Handschuh. Semantic versioning manager: Integrating semversion in protégé. In *Proceedings of Proégé'06 conference*, 2006.

[HH00] Jeff Heflin and James Hendler. Dynamic ontologies on the web. In *Proceedings of AAAI 2000*. AAAI Press, 2000.

[HSG+05] J. Hartmann, P. Spyns, A. Giboin, D. Maynard, R. Cuel, M. C. Suarez-Figueroa, and Y. Sure. Methods for ontology evaluation (D1.2.3). Deliverable 123, Knowledge Web, 2005.

[HV05] Peter Haase and Johanna Völker. Ontology learning and reasoning - dealing with uncertainty and inconsistency. In *Proceedings of the URSW2005 Workshop*, pages 45–55, NOV 2005.

[LBT+07] L. Laera, I. Blacoe, V. Tamma, T. Payne, J. Euzenat, and T. Bench-Capon. Argumentation over ontology correspondences in mas. In *In Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2007), Honolulu, Hawaii, USA. To Appear*, 2007.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Cybernetics Control Theory*, 10:707–710, 1966.

[LTE+06] L. Laera, V. Tamma, J. Euzenat, T. Bench-Capon, and T. R. Payne. Reaching agreement over ontology alignments. In *Proceedings of 5th International Semantic Web Conference (ISWC 2006)*. Springer-Verlag, 2006.

[NHA+07] Vít Nováček, Zhisheng Huang, Alessandro Artale, Norman Foo, Enrico Franconi, Tommie Meyer, Mathieu d'Aquin, Jean Lieber, Amedeo Napoli, Giorgos Flouris, Jeff Z. Pan, Dimitris Plexousakis, Holger Wache, Heiner Stuckenschmidt, and Siegfried Handschuh. Theoretical aspects for ontology lifecycle (d2.3.9). Deliverable 239, Knowledge Web, 2007.

[NHL+06] Vít Nováček, Siegfried Handschuh, Loredana Laera, Diana Maynard, Max Völkel, Tudor Groza, Valentina Tamma, and Sebastian Ryszard Kruk. Report and prototype of dynamics in the ontology lifecycle (D2.3.8v1). Deliverable 238v1, Knowledge Web, 2006.

[NM02] N. Noy and M. Musen. The prompt suite: Interactive tools for ontology merging and mapping, 2002.

[NM04] Lyndon Nixon and Malgorzata Mochol. Prototypical business use cases (D1.1.2). Deliverable 112, Knowledge Web, 2004.

[SEA+02] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. OntoEdit: Collaborative Ontology Development for the Semantic Web. In *1st International Semantic Web Conference (ISWC2002)*, Sardinia, 2002. Springer.

[SS04] S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer-Verlag, 2004.

[TN07] Tania Tudorache and Natasha Noy. Collaborative proégé. In *Proceedings of WWW'07*. ACM Press, 2007.

[TPCB06] V. Tablan, T. Polajnar, H. Cunningham, and K. Bontcheva. User–friendly ontology authoring using a controlled language. In *Proceedings of LREC 2006 - 5th International Conference on Language Resources and Evaluation*. ELRA/ELDA Paris, 2006.

[VEK$^+$05] M. Völkel, C. F. Enguix, S. R. Kruk, A. V. Zhdanova, R. Stevens, and Y. Sure. SemVersion – versioning RDF and ontologies (D2.3.3v1). Deliverable 233v1, Knowledge Web, 2005.

[VG06] Max Völkel and Tudor Groza. SemVersion: RDF-based ontology versioning system. In *Proceedings of the IADIS International Conference WWW/Internet 2006 (ICWI 2006)*, 2006.

[VKZ$^+$05] M. Völkel, S. R. Kruk, A. V. Zhdanova, R. Stevens, A. Artale, E. Franconi, and S. Tessaris. SemVersion – versioning RDF and ontologies (D2.3.3v2). Deliverable 233v2, Knowledge Web, 2005.

[VS05] J. Voelker and Y. Sure. D3.3.1 data-driven change discovery. Technical Report 331, SEKT, 2005.

[VTW05] B. Lithgow Smith V. Tamma, I. Blacoe and M. Wooldridge. Introducing autonomic behaviour in semantic web agents. In *In Proceedings of the Fourth International Semantic Web Conference (ISWC 2005), Galway, Ireland, November.*, 2005.

Related Deliverables

The work presented here is directly related to the following deliverables:

Project	Number	**Title** and relationship
KW	D1.1.2	**Prototypical business use cases** studies the needs of the industry using elaborated use cases of semantics-enabled business solutions.
KW	D2.3.3v1	**SemVersion – Versioning RDF and Ontologies (D2.3.3v1)** – ontology versioning methodology proposal and implementation
KW	D2.3.3v2	**SemVersion – Versioning RDF and Ontologies (D2.3.3v2)** – ontology versioning methodology proposal, implementation and evaluation
KW	D2.3.7	**Report on negotiation/argumentation techniques among agents complying to different ontologies** introduces a technique used for computation of agreed ontology alignment among agents with different preferences.
KW	D2.3.8v1	**Report and Prototype of Dynamics in the Ontology Lifecycle (D2.3.8v1) –** proposal of dynamic ontology lifecycle scenario
KW	D2.3.9	**Theorectical Aspects for Ontology Lifecycle (D2.3.9)** – formalisation of dynamics in ontology maintenance and exploitation

Appendix A

Ontology Versioning Questionnaire – Brief Report on the Results

by VÍT NOVÁČEK[1], SIEGFRIED HANDSCHUH[1] AND MAX VÖLKEL

A.1 Introduction

The document reports on the results of an (anonymous) ontology versioning survey performed in July, 2007 (the survey's online interface is publicly available at `http://smile.deri.ie/limesurvey/index.php?sid=2` – there you can browse the questions, provided by definitions and hints on the proper interpretation of terms used). The survey was created as a joint activity of DERI, NUIG and FZI research centre representatives (the authors of this document).

This introductory section briefly describes the main purpose of the survey, its structure and character of the collected responses. Section A.2 provides analysis of the particular answers. General trends and significant features identifiable among the answers are discussed in Section A.3. Section A.4 summarises the results of the questionnaire. If a reader is interested only in a rough overview of the most important findings, Section A.3 should be sufficient after reading the introduction.

A.1.1 Purpose of the Questionnaire

The main purpose was to analyse requirements and views on ontology versioning among some of the key industry and academia players in the Semantic Web field. Opinion on various issues ranging from abstract theoretical matters to rather specific technical details of vocabulary maintenance was solicited. As such, the query results can provide a basis for standardisation activities in the field of vocabulary management. Moreover, the requirement analysis serves as an input for the SemVersion (see `http://semweb4j.org/site/semversion/`) ontology versioning tool extension.

A.1.2 Structure and Content of the Questionnaire

The survey's structure was organised into three sections according to the topic of the respective questions:

1. *Respondent Specific Modes of Ontology Application* – Aimed at specification of the way in which respondents use ontologies. It was also possible to indicate the type, typical size, complexity, dynamics and other features of the ontologies they use, maintain and/or develop.

2. *General Approaches to Ontology Versioning* – The respondents could select and possibly further specify the approach to ontology version maintenance that is most suitable for their practical needs (e.g. syntactic versioning similar to CVS, transaction-based approach or semantic versioning).

3. *Required Features of an Ontology Versioning System* – Meant to specify some features of a system for ontology version management respondents would find useful in their application domain.

The particular questions are given in Section A.2, together with an analysis of the collected answers.

A.1.3 Characteristics of the Respondents and Responses

23 respondents, mainly from U.S. and Europe, participated in the survey. About 57% were from academia, 30% from industry, 9% from non-profit organisations and competence centers (the rest with unspecified affiliation). The spectrum of fields wherein the respondents were active at the time of making the survey was quite broad – ranging from ontology engineering and reasoning applications development through decision support, e-health and biomedical data processing or NLP to business intelligence and process management, knowledge management, manufacturing or governmental applications. Most respondents answered all the questions properly and provided also additional comments when requested.

A.2 Analysis of the Answers

This section gives a rough statistical overview on the answers to the particular questions.

A.2.1 Respondent Specific Modes of Ontology Application

Q1.1 What is your primary affiliation? See Section A.1.3.

Q1.2 What are the main application domains in which you employ ontologies? See Section A.1.3.

Q1.3 In which way you are involved in ontology/knowledge engineering? About 87% of respondents were active in ontology development, 39% in ontology maintenance. Besides that, about 65% of respondents were also involved in applications of the ontologies (either their own or developed by someone else). One respondent was active in ontology-related tools development.

Q1.4 What type of ontologies do you use? Most respondents deal with domain-specific ontologies (about 78%). Besides that, 39% and 48% of respondents deal also with mid-level and foundational ontologies, respectively.

Q1.5 - Q1.8 What is the average size of other ontologies you use? The sizes for particular types of ontologies (as used by the respondents) are as follows:

1. *foundational* – size specified by about 52% of the respondents
   - class-level – mostly ranging from tens to hundreds, only one respondent specified range $1001 - 10000$
   - property-level – uniformly tens to hundreds again, one respondent specified range $500 - 1000$
   - instance-level – relatively lower number of respondents deal with instances in foundational ontologies; if they do at all, the numbers uniformly range from tens to tens of thousands, two respondents even specifying more than 100000
2. *mid-level* – size specified by about 48% of the respondents
   - class-level – most respondents (about 20%) specify range $11 - 50$, otherwise the answers were uniformly distributed along ranges from units to tens of thousands
   - property-level – most respondents (about 13% in both cases) specify ranges $1 - 10$ and $51 - 100$, ranges of tens to hundreds were also given and one respondent employs tens of thousands of relations
   - instance-level – relatively low number of respondents employs instances in mid-level ontologies – one respondent specified range $11 - 50$, three specified more than thousand (one even more than 100000)
3. *domain-specific* – size specified by about 80% of the respondents
   - class-level – mostly in range of tens (35% of respondents), 13% in range of thousands, two respondents more than 100000, otherwise uniformly distributed along all other ranges
   - property-level – most respondents specified ranges from units to tens (47%), 13% specified range of $101 - 500$ and two users specified ranges $501 - 1000$ and more than 100000, respectively

- instance-level – almost all users deal with instances in domain-specific ontologies; the ranges were more or less uniformly distributed along ranges from tens to thousands, about 26% of respondents deal with more than 100000 instances then

Note that the ranges collected from these questions do not have to be absolutely representative, since there is no "standard" and widely agreed definition of different types of an ontology (even though we explained the sense of the terms we used).

Q1.9 What are the knowledge representation formalisms you use within your ontology representation? Most users use more than one knowledge representation formalism in their applications. The most favourite were RDFS (about 57%), OWL DL (about 48%) and pure RDF (about 43%). Other flavours of OWL – Full and Lite – were used by about 30% and 22%, respectively. A DL-based rule representation language SWRL is used by about 26% of respondents. About 43% of respondents uses also less "classical" (from the Semantic Web point of view) or proprietary knowledge representation formats (e.g. OBO, Datalog and *dlv*, Prolog, Jena Rules, CLIPS or BRM systems implementations).

Q1.10 What is the complexity of ontologies you use? According to the definitions provided in the survey interface, most respondents deal with *intermediate* complexity in ontologies (about 48%). However, the distribution among the *simple* and *complex* alternatives is quite even – about 39% and 43%, respectively. About 22% of respondents deal with more than one level of complexity in their ontologies (either intermediate and complex at the same time, or all alternatives).

Q1.11 What is the schema-level ontology dynamics in your application domain? Most respondents account for rare changes at the schema-level (about 26%), however, about 35% respondents answered that the changes occur often (i.e. weekly) or on daily basis.

Q1.12 What is the instance-level ontology dynamics in your application domain? About 26% of respondents answer that changes at the instance-level occur rarely or occasionally. Almost half of the respondents (about 48%) indicate changes occurring often or on daily basis.

Q1.13 Do you use a versioning system for your ontologies? About 52% of respondents use a versioning system. However, the only real "system" actually used is *subversion* (if specified at all), or custom management of version URIs and associated dates. No system specifically tailored for ontology versioning is referenced.

Q1.14 Do you develop and/or maintain ontologies in a de-centralised and/or collaborative way? About 52% of respondents do deal with ontologies in a collaborative way, about 35% answered no to this question. The decentralised solutions were again mainly based on architectures aimed at general software development. Only one respondent explicitly specified a (custom) methodology specifically tailored to ontology development.

Q1.15 Do you only reuse and/or extend some ontologies? About 43% of respondents reuse external ontologies, whereas about 35% deal only with ontologies developed by themselves.

Q1.16 Which ontology editor do you use? Protégé is the most popular editor (about 52% of respondents use it). Swoop is also relatively popular (about 13%). Besides that, about 60% of respondents use one or more from variery of other editors, ranging from text power-editors like *emacs* through custom XML editors to OntoEdit, OBOEdit or proprietary ontology editors.

A.2.2 General Approaches to Ontology Versioning

Q2.1 Which approach to the ontology versioning would you prefer in your application domain? About 30% of the respondents prefer syntax-based ontology versioning, however, about 22% would prefer rather semantic versioning for their applications. The demand for another offered alternatives was relatively marginal.

Q2.2 What types of inference would you like to be included in the versioning process? Most respondents who answered the question (the 22% who would prefer rather semantic versioning) indicated need for every inference type offered (transitive closure computation, subclass subsumption computation, logical or constraint-based consistency checking). One respondent indicated need for subclass subsumption computation only. No other types of inference were suggested.

Q2.3 What is the preferred alternative of ontology diff computation for your application domain? The respondents were rather undecided between two provided basic alternatives. More answering respondents (about 13%) would prefer semantically rich over computationally efficient (about 4%) diff computation.

Q2.4 What are the features you would like to be included in the computed semantic diff? Presence of ontology change identification (about 22%) is slightly preferred over inconsistencies included in the diff (about 13%). Other feature demanded by one respondent is a link to ontology management interface (e.g. diff visualisation for human users).

A.2.3 Required Features of an Ontology Versioning System

Q3.1 Do you need a facility enabling to discuss versions before they become official? About 61% of respondents need such facility, whereas about 22% do not.

Q3.2 Do you need ontology version branches (like in CVS or SVN) in your application domain? About 65% of respondents need branches, whereas about 17% do not.

Q3.3 What mechanism of addressing versions would you prefer? About 30% of respondents would prefer just URIs for addressing version, about 30% would favour labels of ontology versions. Most respondents who provided additional comments or “Other” answer would welcome both possibilities for addressing versions.

Q3.4 What are the essential ontology versioning functions needed for your application domain? About 65% of respondents consider syntactic diff essential. Semantic diff is considered as essential by about 43%.

Q3.5 Do you need version locking (like in CVS) in your application domain? About 26% of respondents need locking, whereas about 39% do not find it essentially necessary for their application.

Q3.6 What kind of ontology version metadata do you need? The answers were distributed along the provided alternatives (with possibility of specifying additional metadata types) as follows:

- *creation date* – about 78%
- *author* – about 65%
- *valid time* (i.e. automatic expiry time for ontologies) – about 26%
- *provenance URL* – about 35%
- *arbitrary RDF encoded metadata* – about 30%
- *other* – about 13% (basically arbitrary RDF-expressible data as well)

Q3.7 What types of relations between versions are necessary for your application domain? The answers were distributed along the provided alternatives (with possibility of specifying additional types) as follows:

- *successors* – about 65%
- *predecessors* – about 57%
- *suggested alternative versions under discussion* – about 26%
- *other* – one respondent (missing parts, broken parts, relationship of semantics to contexts)

Q3.8 What are the general actions to be performed by an ontology versioning system for your application domain? The answers were distributed along the provided alternatives (with possibility of specifying additional actions) as follows:

- *commit a new version as a successor* – about 83%
- *commit a diff as a new version* – about 26%
- *merge two versions into a new third versions* – about 52%
- *compare two versions* – about 65%
- *query versions* – about 48%
- *other* – one respondent (basically version comparison)

Q3.9 What type of manipulations on the graph of different ontology versions are needed? The answers were distributed along the provided alternatives (with possibility of specifying additional types) as follows:

- *rollbacks* – about 35%
- *cut out a version in the middle* – about 17%
- *insert a version in the middle* – about 13%
- *delete at the end (delete HEAD version)* – about 13%
- *other* – about 9% (cross-linking of ontologies using a special properties, adding weights to then, respective visualisation)

Q3.10 Does your application of ontologies require querying and/or reasoning across multiple ontology versions? About 39% of respondents need such reasoning, whereas about 43% do not find it essentially necessary for their application.

Q3.11 What kind of query functionality do you need? About 26% of the respondents need querying across all versions of all ontologies in the versioning system. About 13% need querying across particular branches only. About 26% need querying against single versions of an ontology. For about 9% of the respondents, no querying on versions is needed at all.

Q3.12 What is the main desired function to be performed by the versioning system? The main desired function for most of the respondents (about 52%) is committing new versions. Retrieving and and querying were much less important (both favoured by about 9% of the respondents). However, the respondents consider all functions as important in general.

A.2.4 Further Comments

There were two relevant comments in this section. First of the states that:

> *. . . there is a paucity of information regarding the versioning of semantic web ontologies, particularly those of OWL. The development and advertising of best practices for ontology versioning would be greatly appreciated. The development of tools that enforce best practices are the next logical step.*

The second one was made by a respondent who comments on two possible alternatives of ontology maintenance work-flow:

> *There are two issues (at least). 1) Repeated editing of a single version on the way to release, in which there may be multiple checkins, and for which diffs and merges are important. 2) Different versions of the same ontology*

released to the public. One approach changes the names of all the classes (e.g. via namespace) but this sometimes bothers the consumers. The other is to publish the ontology, with the same named classes, but at a different URI. I am unclear which is more desirable, but I suspect the latter.

A.3 Analysis of Significant Trends and Features

There are several general trends, features and requirements identifiable among the collected answers, as presented in the dedicated sections below[1]. The number of the participating respondents was not very high, so the results are not necessarily statistically well-founded. However, only the key players in the field of the Semantic Web research and industry were addressed, moreover, the spectrum of domains of interest of the particular respondents was rather broad and representative (see Section A.1.3). This assures certain level of plausibility of the general findings – it allows us at least to draw informative conclusions, infer at least some important public requirements and possibly also base relevant recommendations on them.

A.3.1 Versioning Tools Needed

Many respondents claimed they were using an ontology versioning tool, however, this boils down mostly to use of CVS-like version management (i.e. *subversion*). Practically no tools specifically tailored for ontology versioning were used. At the same time, respondents specify several rather sophisticated and ontology-specific requirements in the survey (e.g. semantic diffs or inter-version ontology querying) that can be hardly implemented within the solutions aimed originally at collaborative software development and maintenance. This leads to the following possible conclusions:

- specialised ontology versioning tools in production state are needed
- until such tools are widely available and in production state, it would be good to have a kind of "best-practices" of ontology maintenance using the current CVS-like systems – this would facilitate adopting mutually transparent policies for vocabulary maintenance among ontology developers

A.3.2 Forked Nature of the Ontology Versioning Topic

The second respondent remark in Section A.2.4 mentions (at least) two different instantiations of ontology versioning settings. Both of these alternatives may be relevant in

[1]More elaborated overview of the findings and answer interpretations available in the full version of the report (part of the Knowledge Web D2.3.8v2 deliverable).

certain application scenarios with some distinct properties (e.g. ontology maintained during time by a centralised authority vs. ontology once being developed by an institution or a research project and then released in order to be further extended and maintained by general public in an uncontrolled way). Possible recommendations on vocabulary management should attempt to cover such differences.

A.3.3 Agreement on Basic Version Metadata Exists

There is a relatively uniform agreement on basic metadata for version annotation. The basic set should be interpreted more or less in the same way among ontology developers. Need for arbitrary RDF-encoded metadata was indicated by about 30% of the respondents. It would be good to have principles and/or examples of creating and documenting such data publicly available.

A.3.4 Discussion is Important Part of the Process

More than half of the respondents explicitly or implicitly admits that the discussion and collaboration is important for the ontology development and maintenance in their application scenarios. However, no common methodology is usually followed (if there is any level of formalisation of the process at all), nor a tool facilitating discussion is used. This leads to the following possible conclusions:

- having methodologies (even very simple ones) and/or tools facilitating discussion on ontology changes over time in a production state would be good; specification of such common principles can be helpful for instance if more subjects are active in an ontology development – following a common "protocol" of change discussion and adoption could be much more productive than negotiating changes in an informal way

- this is partially related to documentation of particular changes – before proposing a change for discussion, it should documented in a way comprehensible (i.e., kind of standardised) by all parties involved

A.3.5 Semantic Versioning Welcome

Semantic version management would be welcome, even though there is no appropriate tool in use. Several features of the semantic versioning were agreed upon among the respondents. This could serve as a basis for recommendations regarding semantic versioning tools development.

A.3.6 Multi-version Reasoning Demanded

Need for querying (which is inherently bound to reasoning) among several versions of an ontology was indicated by many of the respondents at several places in the survey. However, there are currently no tools in production state that would support this feature. Therefore, identification and elaboration of (some) possible approaches to the multi-version reasoning would be helpful in order to facilitate development of mature tools dealing with this issue.

A.4 Conclusions

Though the number of respondents answering the query was not overwhelmingly high, the range of their affiliations and domains of interest was sufficiently representative with respect to the field of the Semantic Web. All respondents answered the questions properly, in many cases providing extensive additional feedback and comments. This allowed for several valuable conclusions (as presented here in Section A.3 and in the extended version of this report), serving well the intended purpose of the survey.