Organized by:

Dipartimento di Informatica
Università degli Studi di Bari

Università degli Studi di Bari

SisInf Lab
Politecnico di Bari

Politecnico di Bari

SEMEDIA
Università Politecnica delle Marche

Università degli Studi di Trento

W3C World Wide Web Consortium
Ufficio Italiano

In collaboration with:

a.r.t.i
Agenzia regionale per la tecnologia e l'innovazione

Regione Puglia

With the support of:

AIA

Sponsored by:

Fondazione Cassa di Risparmio di Puglia

Confindustria Bari

CELI
Language & Information Technology

Cézanne

DERI
Digital Enterprise

FIMESAN
E-health projects & consulting

D.O.O.M.srl
Data Over Ontological Models

Exhicon
Information & Communication Technology

GST
guidare la conoscenza

IBM

imola
project partners

i2K
infomation to knowledge

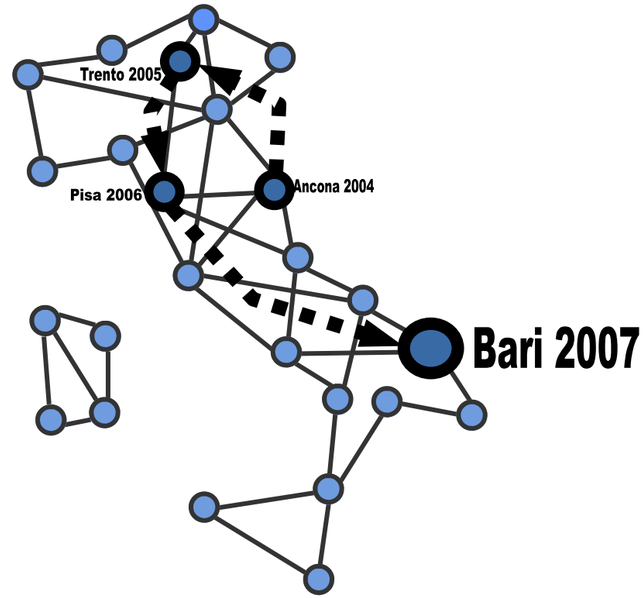
intellisemantic

KIT
Knowledge and Information Technologies s.r.l.

Workshop proceedings

SWAP 2007

Giovanni Semeraro, Eugenio Di Sciascio, Christian Morbidoni, Heiko Stoermer (Eds.)



SWAP 2007

Fourth Italian Workshop on
Semantic Web Applications and Perspectives

18-20 December 2007

Dipartimento di Informatica
Università degli Studi di Bari
Bari, Italy

Workshop proceedings

G. Semeraro, E. Di Sciascio, C. Morbidoni, H. Stoermer (Eds.)

4th Italian Semantic Web Workshop SEMANTIC WEB APPLICATIONS AND PERSPECTIVES (SWAP 2007)

Bari, December 18-20, 2007 Proceedings

ISBN: 978-88-902981-1-0. T — Dip. di Informatica, Università di Bari, 2007.

Preface

The Semantic Web initiative, since its launch by Tim Berners-Lee in 2001, has raised increasing attention and is nowadays one of the most interesting challenges the Computer Science community faces. Several technologies and standardized languages have been the initial outcome of the efforts of a number of researches and projects.

While such languages and technologies move towards maturity, various lessons have been learned; among them is that we will probably never have a single and unified "Semantic Web" but, depending on the issues, different Semantic Web technologies should be used.

Usefulness of Semantic Web technologies is now commonly acknowledged and the offsprings of the basic initiative are increasingly widespread in a variety of application fields, encompassing information retrieval, ubiquitous computing, e-commerce, Web service discovery and composition, data integration, multimedia, social networking, healthcare, among many others.

The contribution of Semantic Web researches to these various fields basically lies in making information structured and interoperable and therefore really machine understandable, thus providing means for machines to somehow automatically "reason" on such information.

As Semantic Web technologies permeate increasingly large application fields, new issues emerge. They include easily creating and managing Semantic Web content, making applications more robust and scalable, devising innovative and useful reasoning services.

To discuss these and other challenges several Italian and foreign researchers gathered in Bari for the fourth edition of the "Semantic Web Applications and Perspectives" workshop (SWAP 2007).

This book collects the papers presented at the workshop, and we are confident both the newcomer and the expert will find it an interesting reading. We received 37 submissions from Italy and several foreign countries, and the Program Committee members selected 24 papers for presentation.

The conference program offered also a tutorial day with five extremely timely topics covered, two invited talks, an FP-7 info-day held in collaboration with ARTI Puglia (Agenzia Regionale per la Tecnologia e l'Innovazione), and an industrial/academic panel on emerging Semantic technologies.

SWAP 2007 was jointly organized by the Department of Informatics of Università di Bari, and SisInfLab of Politecnico di Bari, in collaboration

with ARTI-Puglia, W3C Office in Italy, Semedia group (DEIT, Università Politecnica delle Marche), University of Trento.

We wish to thank all authors that submitted their work to SWAP 2007, the program committe members for their fruitful activity. We are indebted with the members of the Local Committee that worked hard for the success of this event, and grateful to the sponsoring and collaborating companies and entities that provided economical and organizational support.

Giovanni Semeraro,
Eugenio Di Sciascio,
Christian Morbidoni,
Heiko Stoermer

Organization

Workshop Chair
Giovanni Semeraro, University of Bari

Programme Co-Chair
Eugenio Di Sciascio, Politecnico di Bari
Christian Morbidoni, Universita' Politecnica delle Marche
Heiko Stoermer, University of Trento

Local Organization
Pierpaolo Basile, University of Bari
Marco de Gemmis, University of Bari
Anna Lisa Gentile, University of Bari
Leo Iaquinta, University of Bari
Pasquale Lops, University of Bari
Agnese Pinto, Politecnico di Bari
Domenico Redavid, University of Bari
Michele Ruta, Politecnico di Bari
Eufemia Tinelli, University of Bari

SWAP series steering committee
Paolo Bouquet, University of Trento
Oreste Signore, W3C Office in Italy/CNR
Giovanni Tummarello, DERI, Galway, Ireland

Program Commitee
Roberto Basili - University of Roma Tor Vergata
Sonia Bergamaschi - University of Modena and Reggio Emilia
Silvana Castano - University of Milano
Marco de Gemmis - University of Bari
Tommaso Di Noia - Politecnico di Bari
Francesco Donini - Università della Tuscia Viterbo
Floriana Esposito - University of Bari
Aldo Gangemi - LOA-CNR
Roberto Garca Gonzalez - Universitat de Lleida
Fausto Giunchiglia - University of Trento
Francesco Guerra - University of Modena and Reggio Emilia
Pascal Hitzler - AIFB, University of Karlsruhe
Pasquale Lops - University of Bari
Massimo Marchiori - University of Padova
Massimo Martinelli - ISTI-CNR
Daniel Olmedilla - L3S Reseach Center and Hannover University
Maria Teresa Pazienza - University of Roma Tor Vergata

Paolo Romano - National cancer Research Institute of Genoa, IST
Leo Sauermann - DFKI
Luciano Serafini - IRST-Fondazione Bruno Kessler
Sergej Sizov - University of Koblenz
Umberto Straccia - ISTI-CNR
Sergio Tessaris - Free University of Bozen - Bolzano
Giovanni Tummarello - DERI, Galway, Ireland
Guido Vetere - IBM Advanced Internet Technologies

Additional Reviewers
Pierpaolo Basile, Fabio Calefato, Simona Colucci, Stefano David, Nicola Fanizzi, Anna Lisa Gentile, Juan Manuel Gimeno, Leo Iaquinta, Marta Oliva, Mirko Orsini, Agnese Pinto, Azzurra Ragone, Domenico Redavid, Michele Ruta, Antonio Sala, Eufemia Tinelli.

Webmasters
Massimo Bux, Cataldo Musto, Fedelucio Narducci (University of Bari)

In collaboration with:
ARTI Puglia (Agenzia Regionale per la Tecnologia e l'Innovazione)

with the support of:
AI*IA - Associazione Italiana per l'Intelligenza Artificiale

Sponsors:
FCRP - Fondazione Cassa di Risparmio di Puglia
CELI s.r.l.
Cézanne Software
Confindustria Bari
DERI Galway
D.O.O.M. s.r.l.
Exhicon ICT S.r.l.
FimeSan S.p.A.
GST S.p.A.
I.B.M.
Imola s.r.l.
IntelliSemantic s.r.l.
I2K Information to Knowledge s.r.l.
KIT-Knowledge and Information Technologies s.r.l.

Table of Contents

UFOme: A User Friendly Ontology Mapping Environment

Giuseppe Pirrò[1], Domenico Talia[1]

[1]D.E.I.S, University of Calabria
87036 Rende, Italy
{gpirro,talia}@deis.unical.it

Abstract. Recently the Ontology Mapping Problem (OMP) has been identified as a key factor towards the success of the Semantic Web and related applications. This problem arises since it is possible for different people to give, through ontologies, different *conceptualizations* of the same (or overlapping) knowledge domain. In order to tackle the OMP several algorithms have been designed. They aim at discovering correspondences (*aka* mappings) between ontology entities. However, these algorithms mostly suffer from the following shortcomings: (i) do not allow to quickly combine and/or compare different mapping strategies; (ii) do not offer support for evaluating mapping strategies in terms of quality of results and performance. In this paper we present a plugin-based system called *UFOme* along with its current implementation. We illustrate how it can be exploited to graphically design mapping tasks by connecting different types of *modules*. *UFOme* provides three categories of modules. The first one (i.e., *visualization*) allows to explore the ontologies to be mapped. The second one (i.e., *matching*) provides different types of individual matchers, exploited to discover mappings between ontologies, and a module for combining them. The third one (i.e., *evaluation*) enables to evaluate each module of the mapping task, a sub mapping task, or the mapping task in the whole w.r.t performance and quality of results.

Keywords: ontology mapping environment, ontology mapping evaluation

1 Introduction

A central factor towards the success of the Semantic Web (SW) and related applications are ontologies. Ontologies can be exploited to give *conceptualizations* of knowledge domains and to make *explicit* and *machine understandable* the meaning of the adopted terminology. The SW aims at exploiting ontologies for providing resources with semantically meaningful information. However, in distributed environments (e.g., the Web), it is not feasible having a single (and universally accepted) ontology describing a knowledge domain. There will be different ontologies each of which created w.r.t "the point of view" of its designer. That's because people see the world differently and these viewpoints inevitably get encoded into data structures. Therefore, in order to enable reciprocal understanding, such different representations (i.e., ontologies) have to be brought into "mutual agreement". This problem in literature is referred to as the ontology mapping problem (OMP). In order to overcome the OMP, several ontology mapping algorithms, aimed at discovering

correspondences (*aka* mappings) between ontology entities (e.g., classes, properties), have been proposed [1,4,11,14,15,17,18]. However, as also pointed out in [7], these algorithms are often not endowed with adequate *cognitive supports* for helping users in the various steps of a mapping task. Often they do not allow to quickly design, combine and compare different mapping strategies and do not offer support for evaluating mapping strategies in terms of quality of results and performance. Since, as pointed out by several evaluation initiatives [16], ontology mapping is not yet a fully-automated task, it is necessary to enable users to interact with the mapping system in the different phases of a mapping task, as for example: to suggest initial mapping candidates as in [15], to accept/reject mapping candidates and to evaluate results. In particular, we identified three main phases in a mapping task execution:

1. *Designing*: a user design the mapping task by choosing the different *modules*, some of which can require configuration parameters (e.g., threshold), to be included in the task. In this phase s/he can also suggest initial mapping candidates.
2. *Running*: the mapping task is executed according to the strategy defined in the *Designing* phase and values of parameters.
3. *Evaluation*: results of the mapping task are presented to the user which can validate them, perform several types of evaluation and possibly restart the running for discovering additional mapping candidates.

We argue that towards a comprehensive tool for ontology mapping, adequate supports (e.g., GUIs) in all the abovementioned phases have to be provided.

In order to cope with these requirements, we delevoped the *UFOme* (*User Friendly Ontology mapping environment*) system based on the concept of pluggable *module*. *UFOme* provides three different categories of modules each of which supports the user in one or more phases of the mapping task. The first category (i.e., *visualization*) includes a module that enables exploring the ontologies to be mapped. The second one (i.e., *matching*) provides different types of individual matchers, exploited to discover mappings between ontologies, and a module for combining individual matchers. The third one (i.e., *evaluation*) allows to evaluate each module of the mapping task, a sub mapping task, or the mapping task in the whole w.r.t performance and quality of results. *UFOme* also allows to implement both new modules and categories to be included into the system as plugins. Therefore, it paves the way towards a user-friendly, effective and extensible ontology mapping environment.

The remainder of this paper is organized as follows. Section 2 describes the *UFOme* architecture. Section 3 presents a working example. Section 4 reviews related work and compares *UFOme* with similar systems and in particular with the OLA [5] system. Section 5 concludes the paper.

2 The *UFOme* architecture

This section describes the *UFOme* architecture designed taking into account two important requirements: *extensibility* and *usability*. The first requirement is fulfilled by the concept of *module*. A *module* is a generic pluggable component designed to support one or more phases of a mapping task. The second requirement is fulfilled by exploiting several GUIs covering specific aspects of the mapping process.

2.1 The *UFOme* two-layer architecture

The *UFOme* architecture, depicted in Fig. 1, is built upon two layers: *logical layer* and *graphical layer*. The latter represents the layer of interaction with the user through the *mapping task composer*. A user can choose the set of modules to be included in the mapping task and connect them according to the mapping strategy s/he wants to implement. In this phase (i.e., *designing*) both incoming and outcoming module connections are checked in order to verify that modules receive the correct data to process (e.g., a matching module should receive two ontologies whereas an evaluation module a set of mappings). If the mapping task has been correctly composed then it can be executed.

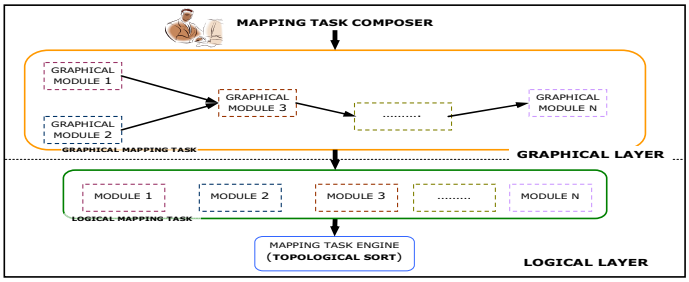


Fig. 1. The *UFOme* architecture.

A mapping task composed at the *graphical layer*, in order to be executed (*execution* phase), is converted into a mapping task at *logical layer*. Here the different modules composing the task process data they received as input. Results are both passed on to the connected modules and stored within the module for possible subsequent analysis (*UFOme* allows to individually evaluate each module). In order to guarantee the correct execution order, *UFOme* relies on the *topological sort* algorithm [2]. The topological sort of a mapping task, which can be viewed as a Directed Acyclic Graph, is a linear ordering of its modules. In particular, each node is executed before all nodes to which it has connections.

2.2 *UFOme* modules

UFOme modules are the building blocks of the system. A module can be represented by the architecture depicted in Fig. 2. It has a set of incoming connections that represent the input, and a set of outcoming connections exploited to collect results. A module also includes a set of configuration parameters.

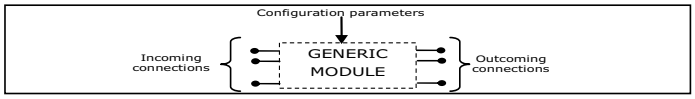


Fig. 2. A generic *UFOme* module.

Currently *UFOme* includes three categories of modules (i.e., *visualization*, *matching* and *evaluation*) that will be briefly described in the following. We want to point out that the aim of this paper is not to describe the modules but to underline the usefulness and effectiveness of the *UFOme* system.

2.2.1 Visualization

This category of modules includes the *OntoLoader* module. It, by exploiting the Jena API [9], allows to visualize an ontology and to obtain useful information such as the list of classes, properties and instances. The ontology is represented as a graph with edges representing the relationships between classes. The user can navigate the ontology and choose different types of visualizations and layouts (see Fig. 4). It is also possible to visualize in the same GUI both the ontologies to be mapped.

2.2.2 Matching

This is the most important category of modules, since through its modules the effective ontology mapping is performed. Currently, *UFOme* includes three individual matchers: *Lucene*, *String* and *Wordnet* and a module for combing them (i.e., the *Combiner* module). Here we provide an overall description of these modules.

Lucene

The Lucene [13] matcher implements the Lucene Ontology Matcher (LOM) algorithm [18]. The aim of the LOM ontology matcher is to exploit all the sources of linguistic information (e.g., local name, comments, and labels) present in the ontologies to be mapped. The LOM matcher aims at discovering mapping between entities (i.e., concepts, relationships and instances) of a *source* and *target* ontologies. In particular, each *source* ontology entity is transformed into a *virtual document* by exploiting the concept of *Lucene Document*. *Virtual documents* are stored into a Lucene index maintained in the main memory. Mappings are derived by using entities of the *target* ontology as search arguments against the index created from the *source* ontology. Similarity between *virtual documents* is computed by the scoring schema implemented in *Lucene*.

WordNet Matcher

The *WordNet* [21] matcher allows comparing ontology entities by considering their semantic meaning. In particular, for assessing similarities between entities, we adopt an adaptation of the Jiang and Conrath Metric (J&C) [10]. This metric along with several others are included in the Java WordNet Similarity Library (JWSL) [8], an ongoing project which aims at providing a Java API for accessing WordNet.

String Matcher

The string matcher implements three algorithms for comparing strings, that is, *I-Sub* [19], *Jaro Winkler* [20] and *Edit Distance* [12]. The user when choosing to include this module in a mapping task can configure the module to use one of the three implemented strategies.

The Combiner module

This module allows to combine/filter results from different matchers according to several strategies (e.g., weighting results of the matchers, introducing a threshold).

It is worthwhile pointing out that *UFOme* allows designing and implementing new matchers that can be included into the system as plugins. This way *UFOme* becomes a comprehensive mapping environment in which developers can implement and plug in new modules according to their needs.

2.2.3 Evaluation

This category of modules includes the *Evaluator*, *Comparer* and *Performance Evaluator* modules. The *Evaluator* module allows evaluating the suitability of a matching strategy in terms of quality of results. In particular, it computes measures of Precision, Recall and F-measure [3] that are classical Information Retrieval metrics. These metrics are based on the comparison of an expected result and the result returned by the system. In the context of ontology mapping, we compare a set of mappings obtained by a mapping task w.r.t a reference alignment.

The *Comparer* module allows the comparison of two matching strategies in terms of Precision, Recall and F-measure. This way the user avoids coding new programs, but just picking up graphical modules (see Fig. 3) can have an immediate background on which of these two strategies is the most appropriate.

The *Performance Evaluator* module allows to evaluate performance (in terms of time elapsed) of the different modules, of a sub mapping task (by considering a subset of modules) or of the mapping task in the whole.

3 *UFOme* : make easy ontology mapping

This section aims at showing the suitability of *UFOme* in a real ontology mapping problem. We chose two ontologies (the 101 and the 205) belonging to the OAEI 2006 [16] benchmark test suite. We examine in detail the different phases of the mapping task execution, and show how *UFOme* can be profitably exploited.

3.1 Phase 1: *Designing*

In this phase the user can choose the various modules to be included in the mapping task (see Fig. 3).

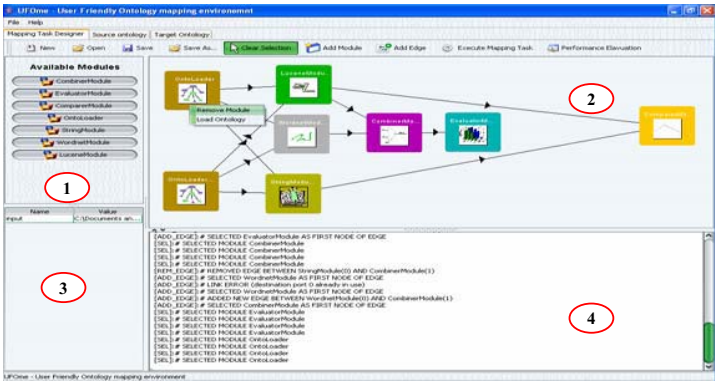


Fig. 3 The *UFOme* GUI.

An *UFOme* user can pick up the modules shown in the left hand side of the *UFOme* interface (1) and put them into the *mapping task composer* (2). Parameters of each module are assigned by exploiting the table (3). Therefore, the modules must be

connected according to the strategy that the user wants to implement. For instance in Fig. 3, the results produced by the two *OntoLoader* modules are passed on to the three matchers. Notice that the direction of the connections will be exploited by the logical layer of the *UFOme* architecture for running the topological sort algorithm (see Section 2.1). Moreover, the log area (4) provides information about mapping activities and possible errors.

After composing the mapping task, that can also be saved, the user can choose to visualize the ontologies to be mapped. That can be done by right-clicking on the *OntoLoader* modules and choosing the *Load Ontology* option (see Fig. 3). The loaded ontology appears as depicted in Fig. 4. The central part (1) shows a graph representation of the ontology while the right column (2) the ontology taxonomy. The dialog (3) allows changing the visualization layout. The toolbar (4) shows other information such as: instances, other types of relationships (i.e., not isa), domain and range of properties, and so forth. It is also possible to show the two ontologies to be mapped in the same *JTab* thus the user can discover and suggest initial mapping candidates.

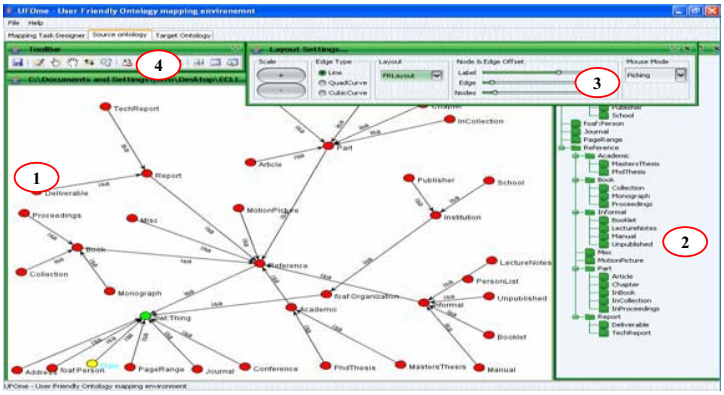


Fig. 4 The *UFOme* ontology perspective.

3.2 Phase 2: *Running*

In this phase the mapping task is executed according to the order determined by the topological sort algorithm. Results produced by each module are both stored in the module, for allowing individual analysis of the results, and passed on to the modules to which it is connected. Once executed a mapping task can be evaluated.

3.3 Phase 3: *Evaluation*

In this phase of the mapping task the user can check results of the task and improve them by choosing a different mapping strategy (i.e., a different combination of modules). In particular, while in current ontology mapping systems, designing different techniques means coding ad-hoc programs, in *UFOme* it corresponds to graphically (re)connect a set of modules.

A user, by right-clicking on a module, can find interesting information related to the execution. For instance, in Fig. 5, by right-clicking on the *Evaluator* module, the user can choose to see the correct, lost or wrong mappings discovered by the (sub) mapping task identified by the dotted area.

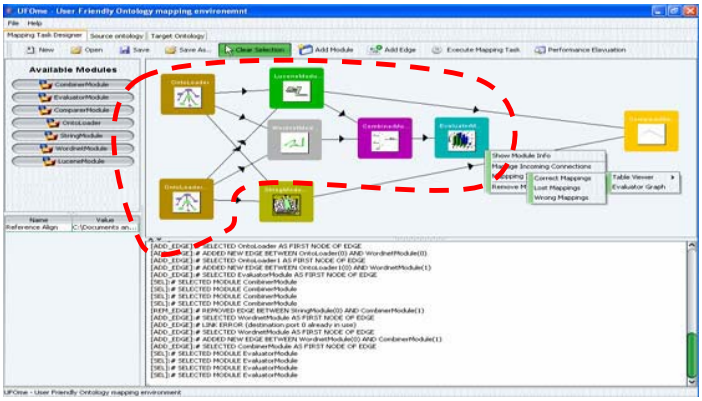


Fig. 5. *UFOme* evaluation options.

In Fig. 6 the correct mappings are compared to wrong mappings on the basis of a reference alignment.

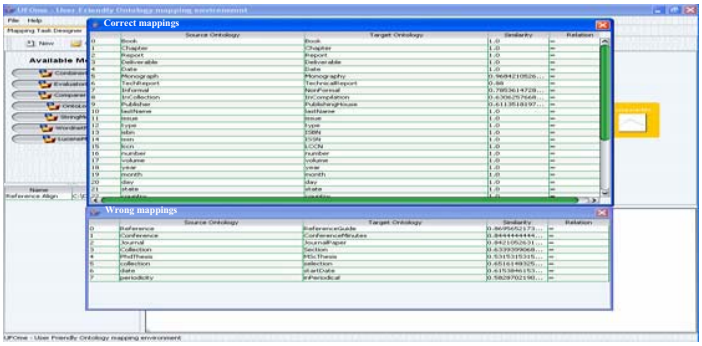


Fig. 6. Comparison between correct and wrong mappings.

Notice that the *Evaluator* module (Fig. 5) takes as input the result of the combination (obtained by the *Combiner* module) of the mappings discovered by both the *Lucene* and *WordNet* matcher. It is important pointing out that the strategies implemented by the *Combiner* module can be several (e.g., weighted sum of the mappings provided by each individual matcher, simple merging of results). The user can also choose to evaluate a mapping task in terms of Precision, Recall and F-Measure. By choosing the *Evaluator Graph* option (see Fig. 5) a new GUI will appear (see Fig. 7).

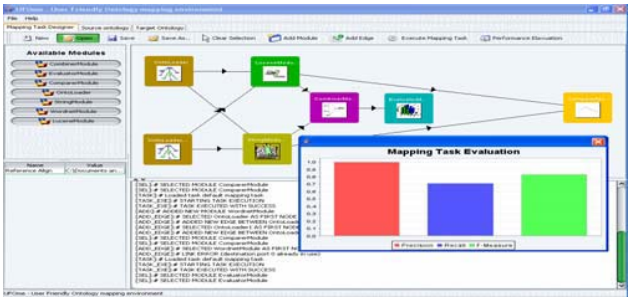


Fig. 7. Evaluation of the results of a mapping task in terms of Precision, Recall and F-Measure.

The *Comparer* module allows the comparison of two matching strategies w.r.t quality of results produced by each of them. Fig. 8 shows the comparison between the *Lucene* matcher and the *String* matcher on the considered ontologies.

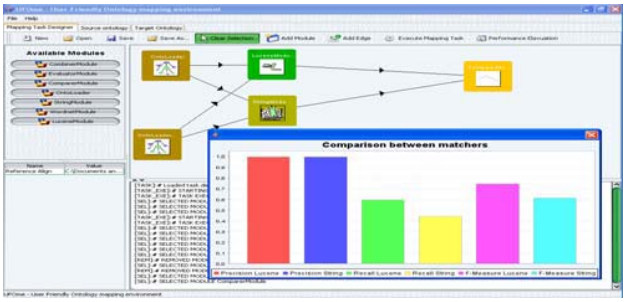


Fig. 8. Comparison between two mapping strategies.

UFOme, differently from other ontology mapping tools that underestimate the importance of performance evaluation, through the *Performance Evaluator* allows to analyze performance (in terms of time elapsed) of a mapping strategy (see Fig. 9).

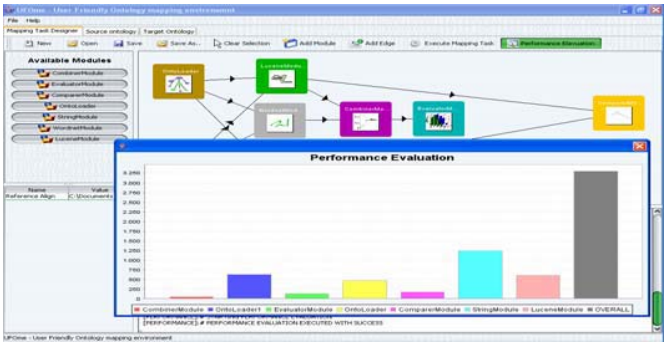


Fig. 9. Performance evaluation of the modules included in the mapping task and of the overall mapping process. Times (*y axis*) are expressed in *msec*.

Fig. 9 shows the times elapsed (on a Pentium IV 3.0 GHz with 2GB memory) by the different matchers as well as the overall mapping task execution time.

4 Related Work

To the best of our knowledge there are no system that entirely covers all the phases of a mapping task identified in the Section 1. In Table 1 we compare the main characteristics of *UFOme* with those of similar tools.

Table 1. Comparision of *UFOme* with similar tools.

	Designing				**Evaluation**	
	Ontology navigation	**Graphical Mapping Composition**	**Candidates Suggestion**	**Modular Architecture**	**Graphic Evaluation Support**	**Graphic Performance Evaluation**
UFOme	Yes	Yes	Yes	Yes	Yes	Yes
OLA [5]	Yes	No	No	No	Yes	No
Prompt [15]	Yes	No	Yes	Yes	No	No
Alignment API [6]	No	No	No	Yes	No	No

As can be noticed, some of the features of *UFOme* are supported by other tools. For instance, ontology navigation is supported by both OLA and Prompt which is implemented as a plugin of Protégé (http://protege.stanford.edu). However, *UFOme* is the only tool that provides a support for graphically composing mapping tasks. The tool closer to *UFOme* is OLA (Owl Lite Alignment). OLA [5] is a system for ontology mapping endowed with a GUI. It is built upon the API described in [6]. Both *UFOme* and OLA are endowed with a GUI. However, *UFOme* provides the mapping task *composer* that allows to: (i) quickly composing mapping tasks; (ii) combine and evaluate different alignments strategies. This latter aspect is often underestimated by mapping algorithms/tools in which designing new strategies correspond to implement new code. OLA does not support the evaluation of the *combination* of different mapping strategies, and in order to evaluate different strategies, batch programs in Java based on the API [6] need to be implemented. OLA features a tool for alignment comparison which computes different metrics (e.g. Precision, Recall). *UFOme* offers the same functionality but also features a performance evaluation module. In particular, the time elapsed for each module and the overall time of the entire mapping process are shown. Finally, *UFOme* also implements the saving of mapping tasks along with related results for future reuse.

5 Conclusions and Future Work

This paper described the *UFOme* system that features a graphical environment for supporting users in all the phases of a mapping task. To the best of our knowledge *UFOme* is the only system provided with a mapping composing interface based on graphical modules that allows a user to quickly design, combine and compare different mapping strategies. *UFOme* gives an effective support in choosing the correct mapping strategy and avoids users the burden to explicitly code new programs when changing mapping strategy. We described the architecture of the system and, through a working example, showed how it can be easily exploited by users.

Moreover, we compared it with similar systems. As future work we aim at including in the system new matching components and performing a more detailed evaluation.

References

1. Choi, N., Song, I., Han, H.: A survey on Ontology Mapping. SIGMOD Record 35(3) (2006) pp. 34--41
2. Cormen, T., Leiserson, C. E., Rivest, R. L., Stein C.: Introduction to Algorithms. MIT Press and McGraw-Hill.
3. Do, H., Melnik, S., Rahm E.: Comparison of schema matching evaluations. In Proc. of GI-Workshop Web and Databases, Erfurt, Germany, (2002)
4. Ehrig, M., Staab, S.: QOM-quick ontology mapping. In Proc. of ISWC 2004, Hiroshima, Japan, (2004) pp. 683--697
5. Euzenat, J., Loup D., Touzani D., Valtchev D.: Ontology Alignment with OLA. In Proc. of EON 2004, Hiroshima, Japan, (2004)
6. Euzenat, J.: An API for ontology alignment. In Proc. of ISWC 2004, Hiroshima, Japan , (2004) pp. 698--712
7. Falconer, S., Noy, NF, Storey, M.: Towards the need of cognitive support for ontology mapping. In Proc. of OM-2006, Athens, Georgia, USA (2006) pp. 25--37
8. Java WordNet Similarity Library (JWSL) and the Similarity Experiment. http://grid.deis.unical.it/similarity
9. Jena - The Jena Project. http://jena.sourceforce.net
10. Jiang, J., Conrath, D.: Semantic similarity based on corpus statistics and lexical taxonomy. In Proc. of ROCLING X, Taiwan (1997)
11. Kotis K, Vouros GA The HCONE Approach to Ontology Merging. In proc. of ESWS 2004, Heraklion, Greece, (2004) pp. 137-- 151
12. Levenshtein, I.V. Binary Codes Capable of Correcting Deletions, Insertion and Reversals. Soviet Physics-Doklady 10(8) (1966) pp. 707--710
13. Lucene- The Apache Lucene project. http://lucene.apache.org
14. Mitra, P., Noy, N. F., Jaiswal, A. R.: OMEN: A Probabilistic Ontology Mapping Tool. In proc. of ISWC 2004, Hiroshima, Japan (2004) pp. 71--83
15. Noy, N.F., Musen, M.A.: The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. Int. J. of Human-Computer Studies 59 (2003) 983-1024
16. Ontology Alignment Evaluation Initiative. http://oaei. ontologymatching.org
17. Pan, R., Ding Z., Yu, Y., Peng., Y.: A Bayesian Network Approach to Ontology Mapping. In Proc. of ISWC 2005, Galway, Ireland (2005) pp. 563--577
18. Pirrò, G., Talia, D.: An approach to Ontology Mapping based on the Lucene search engine library. In proc. of SWAE '07, Regensburg, Germany (2007) pp. 407--412
19. Stoilos, G., Stamou, G., and Kollias, S. A String Metric for Ontology Alignment. In proc. of ISWC 2005,Galway, Ireland, (2005) pp. 623--637
20. Winkler, W. E. The state of record linkage and current research problems. Statistics of Income Division, Internal Revenue Service Publication (4) (1999)
21. WordNet - WordNet online. http://wordnet.princeton.edu/online

A Lightweight Ontology for Rating Assessments

Cristiano Longo[1⋆] and Lorenzo Sciuto[2]

[1] TVBLOB s.r.l. Milano Italy
[2] Università di Catania, Dipartimento di Ingegneria Informatica e delle Telecomunicazioni

Abstract. Various recommender systems and trust engines use application specific formats to store and exchange data. Such data are used for statistical purposes, or to produce recommendations about items or users. This paper introduces **Ratings Ontology**, a semantic web format for ratings and related objects. This ontology aims to be *lightweight*, in the sense of minimising the physical size of data stored or sent over the network for a given ratings set. Moreover, we presents a tools suite based on this ontology to find out statistical information from a ratings data set.

1 Introduction

A big part of on-line services keeps track in different ways of users' taste and satisfaction. Usually this information is collected in form of *ratings* assessed by *users* about *items* of the system itself. *Amazon.com*[1] for example is an on-line book store, that collects user preferences about books. Saved ratings could be used to deduce several kind of statistical information, i.e. to measure user satisfaction about the whole system, or to know best and least liked items. *Recommender Systems* use these ratings to suggest items they may like to the users. As pointed out in [2], recommender system performances increase proportionally to the amount of available information. For this reason, in order to produce good recommendations, it should be convenient to share ratings collected by different, but similarly purposed, on-line services via a common ratings exchange format. Such a format also implies other advantages. It would allow to separate the ratings collection task from the collected data processing, i.e.recommendations production. As a consequence tools and recommender systems could be developed independently from ratings collection engines.

This paper introduces **Ratings Ontology**, a Semantic Web format to store and share ratings. This ontology aims to be *lightweight*, in the sense of minimising the physical size of data stored or sent over the network for a given ratings set. The rest of this paper is structured as follows : Section 2 contains a brief description of semantic web intent and languages; Section 3 describes two ontologies with similar intents to ours, but with more limitations; Section 4 describes the features provided by ratings ontology; Section 5 describes a use-case of this ontology by introducing some tools we developed which are able to process data sets in this format.

⋆ Thanks to G. Di Blasi, P.Oliveto and B.Vintrici for their lexical contribution.

2 Semantic Web Ontologies

Semantic Web provides a common framework to share and reuse data across applications. It provides languages to express information in a machine processable way. The Semantic Web core language is **RDF**[3]. An rdf document can be seen as a graph in which nodes are linked to each other by *properties*. Nodes and properties can be labelled by a Uniform Resource Identifier(URI)[4]. An rdf graph can be seen as a set of triples $(source, property, target)$, that corresponds to graph edges. Such a triple represents a relation identified by *property* that goes from *source* to *target*. The big part of rdf storage engines uses such triples as internal representation of rdf graphs. So the number of triples can be used as a metric to measure the *size* of an rdf document. The **RDF Vocabulary Description Language(RDF Schema)**[5] is a semantic extension of RDF. It provides mechanisms for describing groups of related resources(RDFS classes) and the relationship between them. It allows to define *vocabularies* in terms of classes and properties. **Web Ontology Language(OWL)**[6] enriches the RDF Schema with various constructs and constraints for properties and classes. It defines also some meta-level properties to describe relations between properties and classes. As an example, given two properties p and q, we can state that p is the inverse of q via the $owl : inverseOf$ property. OWL also allows to define cardinality and value constraints, useful to check if instances of a class are consistent; i.e. we can say that a boy has at most one father using $owl : minCardinality$.

3 Related Works

Trust Ontology[7] is an extension of the Friend Of A Friend ontology[8], that defines properties about user profiles. Trust Ontology adds features for *user-to-user* trust assessments. It provides eleven properties, one for each trust value in a zero to ten scale. As a result, trust information is stored in a very compact way into an rdf graph because for each trust assertion just one triple is stored. On the other hand, this ontology allows to express only ratings in a zero to nine scale, and offers no capabilities for other rating spaces. For example, *Movie Lens*[9] uses a five point scale, so Trust Ontology is not suitable for ratings collected by this engine.

Review Ontology[10] has more power. For each rating assessment a corresponding *Review* is defined, with a *rating* and two properties to describe the ratings range: $maxRating$ and $minRating$. This ontology suits all systems with discrete finite equispaced rating ranges. However, it is not yet enough, because some engines, i.e. *Moleskiing*[2], allow users to enter ratings in a continuous interval. Moreover, the presence of $maxRating$ and $minRating$ for each rating entered by a unique engine is redundant and it causes a growth of data storage size.

Ratings Ontology aims to cover the entire ratings spectrum and, at the same time, to reduce the amount of data stored for a given set of ratings.

4 Ratings Ontology

Ratings ontology is an OWL based format that provides classes and properties to represent in an exhaustive and machine processable way ratings collected by web sites, automated agents and other engines. The Ratings Ontology specification is available at [11]. Ratings collected by different engines could be saved in the same storage, in order to increase the accuracy of recommendations, or to get statistics from a larger data set. On the other hand, the engine that collected a rating and the context in which this rating was produced is an important piece of information. For this reason, our ontology provides features to bind a rating with the engine that collected it, and to describe *how* this rating has been collected. The next sections describe the classes and properties introduced by the Ratings Ontology.

4.1 Rating Class

A rating represents a sort of preference, or judgement, *assessed by* a user or a software agent *about* a generic item. For such ratings, Ratings Ontology provides the *Rating* class. Rated items are expressed in terms of RDF resources. This allows to provide a full description of rated items using suitable elements from other ontologies. The rating asserter must be an *Agent*, where the *Agent* class is defined in the $foaf$[8] ontology. We chose the *Agent* class instead of the more restrictive $Person$ to cover situations in which a rating assessment was not caused, directly or not, by a physical person, but by an intelligent agent. As an example, [12] describes how trust assessments among grid nodes could be used to improve the performance of the whole computational grid. Attention should be paid for understanding that this is not the case of ratings collected by an automated agent that *measures* in some way how much a user likes an item, i.e. a browser that keeps track of the amount of time you spent on a web page. In such a situation we say that the rating was collected in an $IMPLICIT$ way and the person whose behaviour has been observed to produce the rating should be considered as the asserter of the rating itself.

A rating can have a *value*. It is an additional information, whose interpretation depends on the context in which the rating has been produced. The browser of the previous example could value ratings by counting the number of times a user visited a certain web page. The great part of engines that collect *explicit ratings* ask the user to enter a preference about an item. In this case, the $value$ property is appropriate to store such a preference. To assure processability and uniformity for third parties software, a rating value must be *numerical*, in the sense that it must be a typed literal with a numeric data type. In a context where no values are assigned to ratings, a rating should be considered as a positive assertion, but the the absence of a rating should be considered like an unknown value, and not as a negative assertion. The Following section contains some code fragments as usage examples of the Ratings Ontology classes and properties. In order to improve readability, we decided to use entity references instead of

full name-spaces in URIs. We use *rat* as a shortcut for the ratings ontology name-space, and *xsd* for the XML Schema one.

4.2 Ratings Collection Engines

Information about the rating context and the engine responsible for the collection is encoded by the *RatingsCollector* class instances. It is appropriate to create an instance for each engine, in order to keep track of who is responsible for the collection of a rating. For example, if two web sites use the same software to produce and store ratings, they should be represented by two distinct *RatingsCollector* instances. For each rating collector the *mode* in which this engine works has to be specified. In the $EXPLICIT$ mode the asserter is explicitly asked to enter a rating about a resource, i.e. by using a form. The $IMPLICIT$ mode was introduced in Sect.4.1, and it covers all scenarios in which the user is not asked explicitly to assess a rating, but ratings are produced observing her behaviour. The following code fragment is the definition of a ratings collector that works in explicit mode.

```
<rat:RatingsCollector rdf:about="http://example.collector1.org">
  <rat:mode rdf:resource="&rat;EXPLICIT_MODE" />
  ....
</rat:RatingsCollector>
```

The most common way to collect explicit ratings is to ask users to choose a preference value for an item from a set of available ratings. For example, at the end of a movie, the user could be asked to choose a rating in the set $\{GOOD, BAD\}$. As pointed out in Sec.4.1, these two values must be saved into our rating data base as numerical values, i.e. using 1 for $GOOD$ and 0 for BAD. The set of available ratings can vary for each collection engine. For example *MovieLens*[9] allows users to enter preferences in the range from 1 to 5 stars. Moreover, there is some system in which available rating values are not a discrete finite set, as in previous examples, but a continuous interval(*Moleskiing*[2]). Ratings ontology provides two different ways to define the range of available ratings. The first one models a discrete finite ratings set via **exhaustive enumeration** of available rating values. For this purpose Ratings Ontology offers the property $hasAllowedRatingValue$, that allows to specify one by one allowed rating values. The following code fragment shows how to encode a Ratings Collector with mode set to explicit and $1, 2, 3$ as available rating values.

```
<rat:RatingsCollector rdf:about="http://example.collector2.org">
  <rat:mode rdf:resource="&rat;EXPLICIT_MODE" />
  <rat:hasAllowedRatingValue rdf:datatype="&xds;integer">
    1
  </rat:hasAllowedRatingValue>
  <rat:hasAllowedRatingValue rdf:datatype="&xds;integer">
    2
```

```
  </rat:hasAllowedRatingValue>
  <rat:hasAllowedRatingValue rdf:datatype="&xds;integer">
    3
  </rat:hasAllowedRatingValue>
</rat:RatingsCollector>
```

The second mechanism is more general but less expressive. At first an interval can be *bounded* or *unbounded*, in one or both directions. The $hasRatingValuesRangeLowerBound$ and $hasRatingValuesRangeUpperBound$ properties respectively allow to define an upper and a lower bound for a ratings range. The following code fragment shows how to encode intervals $[5, +\infty[\subset \mathbb{R}$ and $[1, 1.5] \subset \mathbb{R}$.

```
<rat:RatingsCollector rdf:about="http://example.collector3.org">
  <rat:mode rdf:resource="&rat;EXPLICIT_MODE />
  <rat:hasRatingValuesRangeLowerBound rdf:datatype="&xds;integer">
    5
  </rat:hasRatingValuesRangeLowerBound>
</rat:RatingsCollector>

<rat:RatingsCollector rdf:about="http://example.collector4.org">
  <rat:mode rdf:resource="&rat;IMPLICIT_MODE />
  <rat:hasRatingValuesRangeLowerBound rdf:datatype="&xds;integer">
    1
  </rat:hasRatingsLowerRangeBound>
  <rat:hasRatingValuesRangeUpperBound rdf:datatype="&xds;float">
    1.5
  </rat:hasRatingValuesRangeUpperBound>
</rat:RatingsCollector>
```

A range defined in this manner is assumed to be continuous. If we have a finite or not finite ratings range, in which available values are equally spaced we can encode it with the $ratingsEquispacedWithDistance$ property. Obviously, such a range consists of a set of discrete values. The following code fragment shows how to define the set of even numbers as the ratings range.

```
<rat:RatingsCollector rdf:about="http://example.collector5.org">
  <rat:mode rdf:resource="&rat;EXPLICIT_MODE />
  <rat:hasRatingValuesRangeLowerBound rdf:datatype="&xds;integer">
    0
  </rat:hasRatingValuesRangeLowerBound>
  <rat:ratingsEquispacedWithDistance rdf:datatype="&xds;integer">
    2
  </rat:ratingsEquispacedWithDistance>
</rat:RatingsCollector>
```

Please note that a discrete finite set of equally spaced available ratings could be defined using both of these two mechanisms. In order to increase readability

and minimise the storage size, the definition via enumeration should be used only when the amount of available values is not too large.

4.3 Ratings Collector Class Diagram

These two ways to define the set of available ratings are mutually exclusive, so you can't mix them to create an hybrid ratings range. This constraint was made explicit in the ontology definition creating two disjointed classes for rating collectors, one for each range definition mechanism. The *RatingsCollector* class is defined as the union of these two classes, with the additional *mode* property. Figure 1 outlines ratings ontology classes and properties.

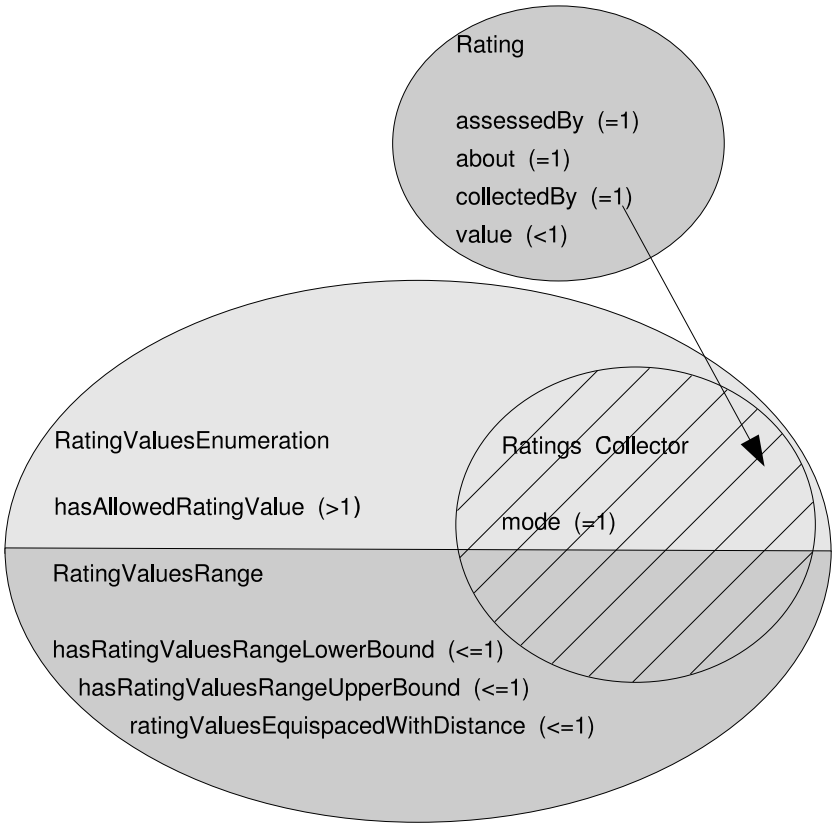


Fig. 1. Ratings Ontology class diagram

4.4 Invalid Ratings

OWL provides features to include into the ontology definition the most of the constraints needed for rating data sets. In example cardinality constraints are used to specify that a rating must have just one asserter. The only additional constraint needed is about rating values related to the set of available ratings provided by the engine. If we found a rating whose value is not allowed by the engine that collected it, this rating should be considered invalid and discarded by tools that process the data set this rating belongs to. The presence of such a rating in the data set probably was caused by an error during the collection task, or by some other processing of the data set itself.

On the other hand, you can define a ratings collector with a ratings range lower bound greater than the upper bound, producing a meaningless definition. We decide to leave unspecified how to handle such a situation, delegating this task to implementations.

5 Applications : Data Set Statistics

As pointed out in Sect.1, a universal format to deal with ratings, as Ratings Ontology aims to become, allowed us to develop software tools that process ratings independently from the engine which collected them. In this section we introduce a set of tools able to find out various statistical information from a set of ratings, stored via the ratings ontology. These tools have been developed using the Jena[13] api for RDF and OWL processing, and the SPARQL[14] support provided by the ARQ engine for queries. The tools are available as set of api together with the command line tools based on the api itself. We have planned to release a *graphical* version in the near future and to make the api also available as a web service.

5.1 Validity Checker

The first tool performs the rating validation described in Sect.4.4. Given a resource into an rdf *model*, the first feature is to detect whether it is a valid rating or not. Moreover we provide a Jena reasoner to discover all errors in a model, signalling also invalid ratings.

5.2 Collection Engines

As pointed out in Sect.4, a common data set could be used to store ratings collected by different engines. The second tool extracts all ratings engines defined into an rdf model, providing also basic information about them like the mode and the set of available ratings.

5.3 Asserters and Items

Our suite also provides features to retrieve the following information from a rating data set :

1. number of rating asserters;
2. number of rated items;
3. the list of all asserters;
4. the list of all rated items;
5. total number of ratings;
6. total number of *distinct* ratings;
7. ratings *density*.

It can happen that a user assesses two different ratings for the same item at two different times. The System that keeps track of this fact should store additional information to distinguish these two different events (for example a timestamp). We say that two ratings are *distinct* if they differ for asserter, rated item or both.

Density measures the amount of available information provided by a ratings set. It coincides with the density of the bi-dimensional matrix labelled with asserters and items, and with rating values into cells. We use the formulation of density that can be found in [15]:

$$Density = \frac{IU}{I * U} \tag{1}$$

where U is the total number of asserters, I is the total number of rated items, and IU is the total number of distinct ratings. All of these calculations can be restricted to a single collection engine. So, given a collection engine E we can retrieve the number of asserters which have at least one rating collected by E, the number of items with at least one rating collected by E, and so on.

5.4 Statistics on rating values

We can find out various statistics from a set of ratings collected by the same, well known, engine. For example we can get:

1. frequency distribution of available rating values;
2. average and variance of ratings;
3. average rating for an item;
4. the list of more rated items.

Dealing with ratings collected by different engines is a more subtle task, because the *raw* numerical value of a rating is meaningless without any information about its collector ratings range. Given two different collector engines $E1$ and $E2$ with a limited ratings range(a rating range with an upper and a lower bound), ratings collected by these two engines could be *normalised* into the interval $[0, 1]$ in order to be processed in a uniform way. For this reason, our suite allows to find out statistics about ratings collected by two or more engines with finite rating ranges.

6 Conclusions And Future Works

This paper introduces Ratings Ontology, an OWL based format to deal with ratings and related matters. This ontology also provides elements to describe collection contexts, that contain all information needed to give a correct interpretation of a rating and of its value. In Sect.5 we presented some tools that work on documents in this format, showing how a uniform exchange format could be useful to develop collection engine independent tools. We have planned to deliver two different applications that deal with data sets with elements described through Ratings Ontology. At first, to help researchers and companies we want to develop a full test suite for collaborative filtering and trust algorithms. This task involves the definition of a *common format* to describe test results, and the development of some tools for statistical results visualisation. Another application of this ontology could be a generic framework for recommender systems. We are considering to use SWAMI[16] as starting point.

References

1. Linden, G., Smith, B., York, J.: Amazon.com recommendations: item-to-item collaborative filtering. In: Internet Computing. Volume 7., IEEE (2003) 76– 80
2. Avesani, P., Massa, P., Tiella, R.: A trust-enhanced recommender system application: Moleskiing. In: Proceedings of the 2005 ACM symposium on Applied computing SAC '05. (2005)
3. Herman, I., Swick, R., Brickley, D.: Resource description framework (rdf) (2004) http://www.w3.org/RDF/.
4. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform resource identifier (uri): Generic syntax. In: Request For Comments. Number 3986. IETF
5. Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. (2004) http://www.w3.org/TR/rdf-schema/.
6. McGuinness, D.L., van Harmelen, F.: Owl web ontology language overview (2004) http://www.w3.org/TR/owl-features/.
7. Golbeck, J.: The Trust Ontology. http://trust.mindswap.org/trustOnt.shtml.
8. Brickley, D., Miller, L.: The friend of a friend (foaf) project. (2007) http://www.foaf-project.org/.
9. Miller, B., Albert, I., Lam, S., Konstan, J., Riedl, J.: Movielens unplugged: Experiences with a recommender system on four mobile devices. In: 17th Annual Human-Computer Interaction Conference, GroupLens Research (2003) http://movielens.umn.edu/.
10. Ayers, D.: Review vocabulary http://dannyayers.com/xmlns/rev/.
11. Longo, C.: Ratings ontology http://www.tvblob.com/ratings/.
12. Farag, A., Muthucumaru, M.: Evolving and managing trust in grid computing systems. In: Canadian Conference on Electrical Computer Engineering. (2002)
13. McBride, B.: Jena: Implementing the RDF Model and Syntax Specification. In: Semantic Web Workshop, WWW2001. http://jena.sourceforge.net/.
14. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (2007) http://www.w3.org/TR/rdf-sparql-query/.

15. Caldern-Benavides, M.L., Gonzlez-Caro, C.N., de J. Prez-Alczar, J., Garca-Daz, J.C., Delgado, J.: A comparison of several predictive algorithms for collaborative filtering on multi-valued ratings. In: ACM symposium on Applied computing. (2004)
16. Fisher, D., Hildrum, K., Hong, J., Newman, M., Thomas, M., Vuduc, R.: SWAMI: a framework for collaborative filtering algorithm developmen and evaluation. In: Research and Development in Information Retrieval. (2000) 366–368

Links and Cycles of Web Databases

Masao Mori[1], Tetsuya Nakatoh[2], and Sachio Hirokawa[2]

[1] O ce for Information of University Evaluation, Kyushu Univ., Fukuoka, Japan.
mori.uoc@mbox.nc.kyushu-u.ac.jp
[2] Research Institute for Information Technology, Kyushu Univ., Fukuoka Japan.
{nakatoh, hirokawa}@cc.kyushu-u.ac.jp

Abstract. This paper proposes a novel framework for composing web databases. Web databases are assumed to have explicit descriptions of I/O attributes and are considered as components of functional compositions. A user writes a script to connect output channels and input channels of components. A script determines a directed graph that may contain cycles which formalizes interactive and iterative behavior of a user through a browser. The interaction and iteration are realised by the notion of CGI-link. Auxiliary lters are introduced as components for universal manipulating tools. (**Keywords**: web service composition, mashups)

1 Introduction

This paper proposes a novel framework for composing web databases. Under the framework we implemented a system which is open to public[3].

Web databases, sometimes called *deep webs*[1], *hidden webs* or *invisible webs*, have been paid attention since around 1996 because of their huge amount of information. Recently many web databases have been newly reconstructed into *web services*, like Amazon.com, Google, and so on. Web services provide access methods (API) for their hidden databases. On the other hand, for the purpose of accessing web databases there are many researches of *web wrappers*. A web wrapper collects information by analyzing HTML codes output from the human interface of a web database, e.g. [7],[8],and [9]. By virtue of web wrappers and APIs web developers are motivated to create a *web service composition* and the new style of web contents *mashup*. While BPEL[10] is one of outcome from research of web service composition, mashup is a new style of combination of web services. Many mashup sites are implemented using visualization of AJAX techniques and communications of the REST style. Sabbouh et al.[11] proposed the Web Mashup Scripting Language which provides a set of procedures of JavaScript in order to integrate web services. Yokoyama et al.[15] studied a framework of AJAX for lightweight implementations. Importance of componetization of web services and web databases has been pointed out in [14] and [13], before mashups obtained much attention as we see now.

[3] Available at http://hyoka-inf.ofc.kyushu-u.ac.jp/%7Emori/research/PSM/

Mashups have two types of processing; server side processing and client side processing. As for client side processing AJAX become popular to realize mashups because mashups with AJAX are supposed to process light-weight data. In this paper we focus on server side processing because of heavy-weight data processing. Currently our system adopt REST style communications as for web services, and web crawling as for web databases.

It seems that most of mashups provide integration of data rather than integration of process ows. In fact most of mashup web sites use only two or three web services. They do not need complex descriptions of process ows. Focusing on integrating web service feeds, Tatemura et al.[12] proposed "Mashup Feeds" which retrieves multiple feeds from many sites and provides users with a set of tools to manipulate the collection.

Mori et al.[5] proposed a novel approach and its system that generates mashup CGIs by giving a simple description of web databases compositions and stores the mashup CGIs in order to reuse them. The problem left in the researches [5] and is the actual interface using web browsers. In this paper we propose graphical primitives for mashup and give solutions for the following questions:

1. What is an easier script style to combine web services and web databases?
2. How does the system manage to layout and display data from multiple web services?
3. What is a better way to carry out next mashup execution and search?

We will introduce the notion of "user interface component"which is a key primitive to layout and display data, and carrying out the next execution step of mashups.

The structure of the paper is organized as Fig.1. New proposals are marked with asterisks(*). Section 2 explains a standard architecture for implementing mashup which requires basic components and their composition. In section 3,

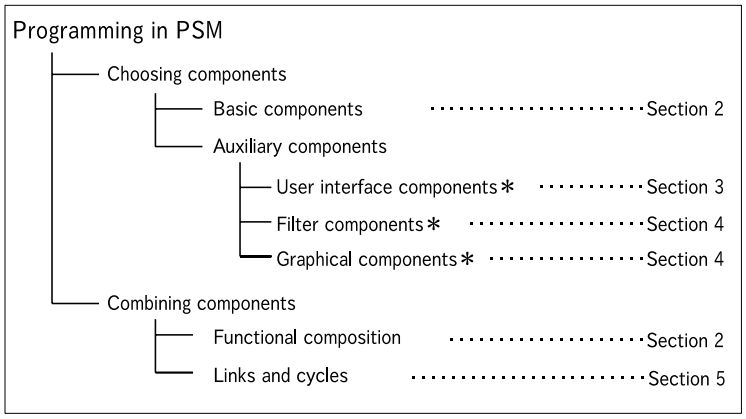


Fig. 1. The programming paradigm of PSM

we analyze how users use web databases with browsers. As a result, we introduce "user interface components" as new auxiliary components. In section 4, "fllter components" and graphical components are introduced. In section 5, we introduce the notion of links and cycles as new methods of composition. These methods capture the repeated interaction of between a user and web databases.

2 PSM Architecture

Our system consists of three parts: interface server, CGI generator and mashup server. When a user accesses the interface server, the server provides a web interface for the user to describe mashups. A description of mashup is called a *mashup script*. Once the interface server passes a mashup script to the CGI generator, the generator forms a *mashup CGI* which is stored in the mashup server. The mashup CGI is executed in the mashup server and performs administration of communication and data processing so that the user can reuse the mashup CGI. The architecture of our system is named as the *Personally Scripting Meta-CGI* architecture, *PSM* for short. The overview of the architecture is shown in Fig 2.

2.1 I/O Attributes and I/O Composition

We call the subjects that input and output in PSM, as *component*. A mashup script is essentially a graph over components: paths of the graph shows data ow amongst components and each edge shows correspondence of attributes in components. The syntax of mashup scripts will be introduced in the rest of this section.

Most of web services provide *complex queries* in their search functions. A complex query is composed of a tuple of keywords for which web services return

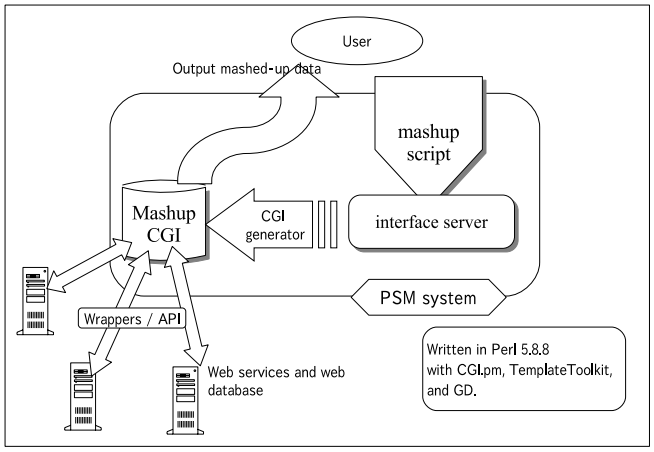


Fig. 2. An overview of PSM

	Rhapsody(`www.rhapsody.com`)		*Amazon*(`www.amazon.com`)	
	attribute	description	attribute	description
input	`artist`	name of artists	`ItemSearch`	keyword search
	`album`	names of CD titles	`ProductSearch`	product id search
output	`artist`	names of artists	`artist`	names of artists
	`album`	names of CD titles	`album`	names of CD titles
	`track`	url of the web page	`URL`	url of the web page

Fig. 3. API description of Rhapsody and Amazon

collections of tuples as search result. Search functions of web services are provided with a URL of API and variables of API. In this paper we call names of variables *attributes*. We introduce two web services for example in Fig.2.1. The flrst one is Rhapsody which is an online music web service. The second example is Amazon Web Service whose API is for database of music products in Amazon.com. Note that these examples are excerpts from original web service API.

We deflne attributes of complex queries as *input channels* and attributes of tuples in search results from web services as *output channels*. We call both of them *I/O channels* of web services. In PSM data on I/O channels are collections of tuples.

2.2 Functional Composition

Functional composition of web services is data passing from output channels on one web service to input channels on another. A *mashup script* consists of descriptions of functional compositions. For example, in order to pass data from the output channel `artist` of Rhapsody to the channel `ItemSearch` of Amazon, the mashup script should have:

```
Rhapsody.artist -> Amazon.ItemSearch,
```

We call a pair of components as a *functional composition expression*, fc-expression for short.

The mashup CGI starts to work when the initial query is given, so that the mashup script must include at least one description about the initial query. Let us consider a special component *Start* to output the initial query to web components.

```
Start.x -> Amazon.ItemSearch,
```

The initial query might be complex, like

```
Start.k1:k2 -> Rhapsody.artist:album,
```

Keywords from the output channels `k1` and `k2` of `Start` are passed to the input channels `artist` and `album` of `Rhapsody`, respectively. A fc-expression with complex data passing is written with tuples of channels separated by colon.

2.3 The Syntax of Scripts

Now we deflne the mashup script with BNF. Note that $\langle fce \rangle$ denotes fc-expressions.

$$
\begin{aligned}
\langle MashupScript \rangle &::= \langle wslist \rangle \text{ "|" } \langle exps \rangle \\
\langle wslist \rangle &::= \langle wsname \rangle \{\text{","} \langle wsname \rangle\} * \\
\langle exps \rangle &::= \langle fce \rangle \{\text{","} \langle fce \rangle\} * \\
\langle fce \rangle &::= \langle ws \rangle \text{"->"} \langle ws \rangle \\
\langle ws \rangle &::= \langle wsname \rangle \text{"."} \langle chan \rangle \\
\langle chan \rangle &::= \langle attr \rangle \{\text{" : "} \langle attr \rangle\} * \\
\langle attr \rangle &::= \langle attrname \rangle \mid \langle attrname \rangle \text{" } * \text{"} \\
\langle wsname \rangle &::= \text{"names of web services"} \\
\langle attrname \rangle &::= \text{"names of attributes"}
\end{aligned}
$$

Asterisks "$*$" added to $\langle attr \rangle$ is a *word separator* which will be introduced in the next section. Like Rhapsody and Amazon, web services and web databases with structured I/O channels are called by *web components*.

3 User Interface Component and CGI link

Now we consider roles of web browsers in PSM. Web browsers display data from web components on client PCs. Since we suppose that data in PSM are

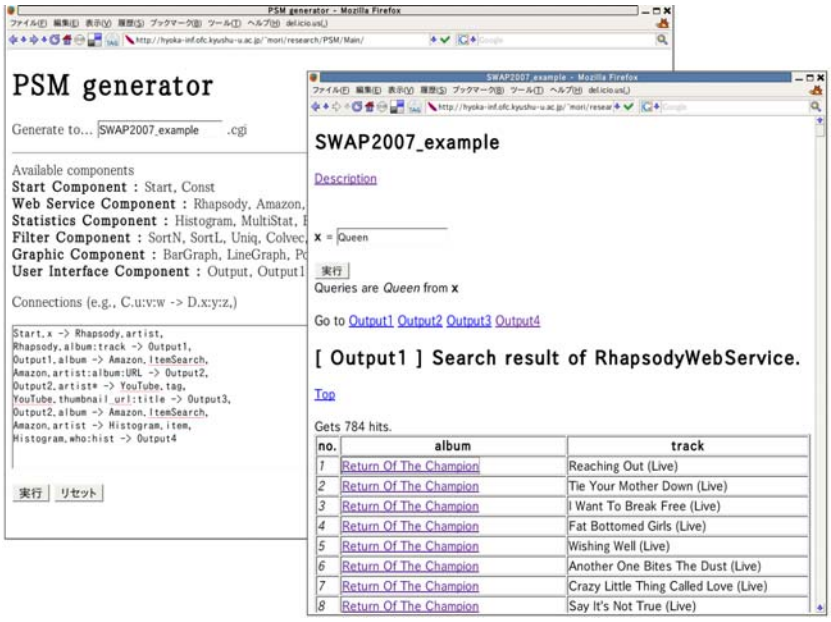


Fig. 4. Interface server(left) and a generated CGI "`SWAP2007_example.cgi`"(right)

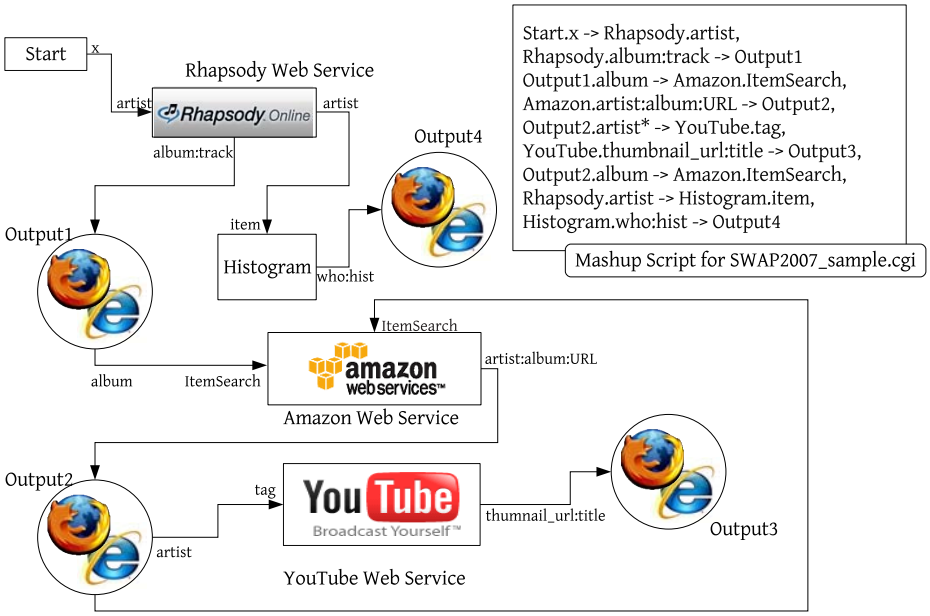


Fig. 5. The mashup script and its graph for `SWAP2007_example.cgi`

sent through structured I/O channels from some web component, web browsers obtain not as text but collections of tuples. We defne *user interface components*, UIC for short, that receive collections of tuples into input channels and display them in appropriate forms (e.g., HTML tables `<table>...</table>`) on client PCs. We denote it by `Output`. If distinct UICs are required, we can distinguish them by indexing, like `Output1`, `Output2` and so on.

As input channels of a UIC can be known by output channels of the web component, channels of UIC can be omitted. For example, the functional compositions to a UIC `Output` like;

```
Rhapsody.album:track -> Output.album:track,
```

but we can write

```
Rhapsody.album:track -> Output,
```

We note three things about channels of UICs. Firstly we set that all of UICs must have the same names of output channels as names of input channels while names of input channels are determined by web components. Secondly we note variability of input channels of UIC. In the case that a UIC is on the right hand side of fc-expression, input channels of the UIC depend on the web component on the left hand side of fc-expression. Thus input channels of UIC are variable. Thirdly, output channels of a UIC can be regarded as output from users. This idea is very important. We will study this idea in the rest of this section.

How and what do we find keywords to continue web search? In many cases keywords might be chosen from the previous results. Let us consider the mashup script that generates `SWAP2007_example.cgi`[4]. The UIC `Output1` appears both in the right hand side of the second line and in the left hand side of the third line. While `Output1` of the second line displays a collection of tuples (`album`,`track`) from the web component `Rhaspody`, functional composition of the third line means to set hyperlinks on all words which appears at the "`album`" column in the table `Output1`. Those hyperlinks call `SWAP2007_example.cgi` that send those words as queries to `ItemSearch` of Amazon web service API. Seeing Fig.4 search results of `SWAP2007_example.cgi` with hyperlinks on "`Return Of The Champion`" are shown in the front window. We call hyperlinks generated by output channels of UICs, *CGI links*. Note that a loop appears in the 4th and the 7th lines of the script, and a component named as `Histogram` appears in the 8th and 9th lines. These notions are introduced in section 4 and 5.

Sometimes data in one column of a UIC forms series of keywords. For example, let us observe the search result from Amazon web service arisen by a CGI link of `Output2` in Fig.6. Series of names of artists can be seen at the `artist` column. They are marked o one phrase (or word) with comma. In order to make a CGI link for each keyword, the word separator, asterisk $*$, is put after the concerned output channel of UICs, like the forth line of the script in Fig.5.

The right window of Fig.6 (c) is the result by clicking the CGI link of "Queen" (in the 6th row, "`artist`" column) which arise the search API of YouTube with keyword "`Queen`".

Now we discuss the three questions posed in the first section. By giving graphs of components we resolve the first question. We introduce user interface components for data layout management(question 2). Finally we prepare the CGI link machinery in order to set triggers for next search(question 3).

4 Filters and Graphical Components

As we have seen, mashup scripts are essentially graphs over components. A path on the graphs can be regarded as a pipeline for collections of tuples. So that we have implemented *filter components*, like UNIX pipeline processing.

SortL, SortN To sort data with respect to the first input channel of a web component. `SortL` for lexicographic order and `SortN` for numbers. Input channels of the sort component can be omitted.

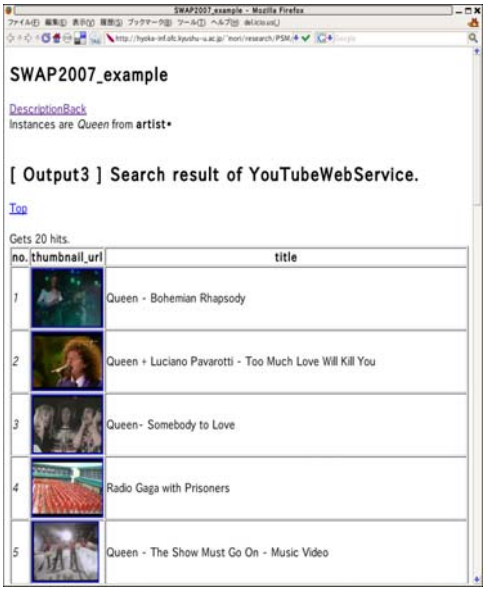
```
Rhapsody.track:artist:album -> SortL,
SortL.album:track:artist -> Output,
```

The result of this example would be ordered data with respect to music title (`track`) from Rhapsody.

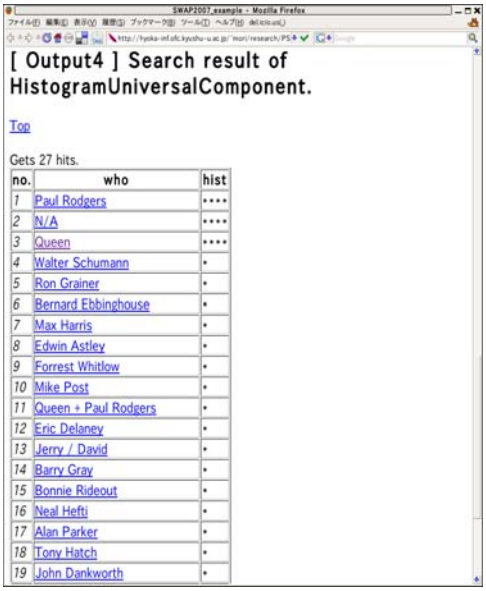
[4] The sample script is available at
`http://hyoka-inf.ofc.kyushu-u.ac.jp/%7Emori/research/PSM/Generated SWAP2007_example.cgi`
and the description of YouTube API can be found in `www.youtube.com`.

(a) Search result for the query "`Queen`"

(b) Invocation of the query "`Return Of The Champions`" from Output1



(c) Invocation of the query "`Queen`" in the 6th row, a rtist column from Output2



(d) Invocation of Histogram component from Output1

Fig. 6. Executions of `SWAP2007_example.cgi`

Uniq To remove duplication of records.

```
Rhapsody.track:artist:album -> Uniq,
Uniq.artist:album -> Output,
```

Colvec Extract a column from the collection of tuples.

```
Rhapsody.artist -> Colvec,
Colvec.id:value -> Output,
```

`Colvec` extract the column `artist` of data from Rhapsody component.

Transpose Regarding the collection of tuples as a matrix, this fllter transpose data.

I/O channels of sort and uniq fllter components are variable as well as UICs, and input channels of fllters in the right hand side of fc-expression can be omitted. Now *graphical components* are introduced.

Histogram To count the appearance of a specifled attribute at the input channel `item` of this component and make histograms. Output channels are `who` for appeared keywords and `num` for numbers of appearance. See the 8th and 9th line of `SWAP2007_example.cgi` and the result in Fig.6 (d).

BarGraph, LineGraph, PointGraph Those components receive vectors and plot graphs.

We can generalize about the component in terms of the standard output from user interface components and graphical components. Those components involve CGI links not only on text, but also multimedia objects.

5 Links and Cycles

Mashup feeds[12] is designed to make programs to collect feeds periodically. It iterates procedures by time-based scheduling. This method is suitable for feeds processing. On the other hand WMSL[11] utilized the control structure of JavaScript for iteration.

Since mashup scripts are written in simple descriptions of graphs over components, they might include cycles in the graphs of components. Note that cycles play a role of iteration in PSM. If the cycle consists of only web components, its execution would result in an inflnite loop. If the cycle includes at least one UIC, it is possible to stop the iteration at the UIC. Thus links and cycles in PSM can control loops.

See the mashup script `SWAP2007_example.cgi` again and note the 4th and 7th lines where a loop can be found. It is easy to presume that the mashup script would stop at each loop step by the CGI links in `Output2`.

6 Conclusion and Future Works

We proposed the mashup scripting system PSM which resolve the three proper questions for mashups introduced in the flrst section. The idea of functional composition of components leads us to a simple format (graphs) of mashup

scripts. Moreover we proposed the new mashup programming style like UNIX pipeline processing. In this style loops can be realized by cycles of components, and can be controlled by CGI links.

PSM is implemented in Perl, independent of WSDL[2]. Data from web services and web databases are transformed into lists of hash in perl codes. All of data processing are done in single server, so that we need to improve the system to reduce overhead.

References

[1] BrightPlanet. Deep web. White Paper, 2000.

[2] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. Technical report, World Wide Web Consortium, March 2001. `http://www.w3.org/TR/wsdl`.

[3] K. Hemenway and T. Calishain. *Spidering Hacks*. O'Reilly & Associates Inc., Mar. 2003. ISBN-13 978-0596005771.

[4] R. Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

[5] M. Mori, T. Nakatoh, and S. Hirokawa. Functional composition of web databases. In *Proceedings of International Conference Asian Digital Libraries 2006*, Lecture note in Computer Science 4312. Springer Verlag, 2006.

[6] M. Mori, T. Nakatoh, and S. Hirokawa. A light-weight implementation of mashups (in japanese). In *Proceedings of Data Engineering Workshop 2007*, C7-152. IEICE, 2007.

[7] T. Nakatoh, K. Ohmori, and S. Hirokawa. A report on metadata for web databases. In *IPSJ SIG Technical Reports*, 2004-ICS-138(17), pages 95–98, 2004.

[8] T. Nakatoh, K. Ohmori, Y. Yamada, and S. Hirokawa. Complex query and metadata. In *Proceedings of ISEE2003*, pages 291–294, 2003.

[9] T. Nakatoh, Y. Yamada, and S. Hirokawa. Automatic generation of deep web wrappers based on discovery of repetition. In *Proceedings of the First Asia Information Retrieval Symposium (AIRS 2004)*, pages 269–272, 2004.

[10] OASIS. *Web Services Business Process Execution Language Version 2.0*, April 2007. OASIS Standard.

[11] M. Sabbouh, J. Higginson, S. Semy, and D. Gagne. Web mashup scripting language. In *Proceedings of the 16th international conference on World Wide Web 2007*, pages 1305 – 1306. ACM Press, May 2007.

[12] J. Tatemura, A. Sawires, O. Po, S. Chen, K. S. Candan, D. Agrawal, and M. Goveas. Mashup feeds: Continuous queries over web services. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1128 – 1130. ACM Press, June 2007.

[13] J. Yang. Web service componentization. *Communications of the ACM*, 46(10):35–40, 2003.

[14] J. Yang and M. P. Papazoglou. Web component: A substrate for web service reuse and composition. In *Advanced Information Systems Engineering: 14th International Conference, CAiSE 2002 Toronto, Canada, May 27-31, 2002. Proceedings*, pages 21–36. Springer Verlag, May 2002.

[15] S. Yokoyama, A. Matono, S. M. Pahlevi, and I. Kojima. A framework for modularization and mashup of javascript codes on web2.0 (in japanese). *DBSJ Letters*, 5(3), December 2006.

Ontology-Driven Generation of a Federated Schema for GIS

Agustina Buccella[1], Domenico Gendarmi[2], Filippo Lanubile[2], Alejandra Cechich[1], and Attilio Colagrossi[3]

[1] GIISCO Research Group,
Departamento de Ciencias de la Computación,
Universidad Nacional del Comahue, Neuquen,Argentina
{abuccel,acechich}@uncoma.edu.ar
[2] Dipartimento di Informatica,
University of Bari,
Via E. Orabona, 4 - 70125, Bari, Italy
{gendarmi,lanubile}@di.uniba.it
[3] Dipartimento Tutela delle Acque Interne e Marine,
APAT, Via Curtatone, 3 - 00185, Rome, Italy
attilio.colagrossi@apat.it

Abstract. In this work we propose an extension of a Federated System, named Information Broker, developed with the Italian Agency for Environmental Protection and Technical Services (APAT). The main objective of this proposal is to build an integrated system taking into account autonomous, distributed and heterogeneous geographic sources. Our extension is aimed at improving aspects as redundancy, consistency, and scalability by adding semantic interoperability through the use of ontologies and the ISO 19100 standards.

Key words: Geographic Information Systems, Federated Systems, Ontology, ISO 19100 Standards

1 Introduction

The APAT was established in 1999 to carry out scientific and technical activities in the national interest to protect the environment, water resources and soil. Data collected include climatic, hydrometric, cartographic and water pollution measures. Although all the information is owned by the same organization, the huge amount of information is managed by different departments and units. Besides, given the large diversity in syntax and semantic of data, measures are stored into several independent systems, which are based on the most appropriate technology for their data type. All these characteristics have made very hard to share information among the different systems. Thus, the main goal of the APAT Information Broker project is to develop a system to provide a fully and user-transparent integration of the heterogeneous data sources, ensuring at the same time, the existing legacy applications that operates on them will continue operating autonomously, without undergoing any sort of modification.

In previous work [1, 2] we have developed an Information Broker System together with a schema integration process focusing specially on syntactic interoperability. This system is mainly represented by using XML data models for the integration process without storing semantic information. Therefore, the process is made manually, increasing the chance of introducing errors and inconsistences.

In this work, we propose an extension of the schema integration process by adding semantic information through the use of ontologies [3]. Then, the Information Broker System will be implemented as an ontology-driven system in order to share the real common vocabulary contained in the sources. We have focused on ontologies due to the advantages they provide to an integration process – as ontologies are formally described, i.e. by using some logic language such as Description Logic [4], we will be able to perform inferences and check inconsistencies easily.

Our extension is based on previous work on integration of geographic information [5, 6], which focuses on two main aspects: modelling and integrating ontologies. With respect to the former, the ontologies are created towards integration by using a family of the ISO 19100 standards (prepared by ISO Technical Committee 211 (TC211)[4]). Specially ISO 19109 [7], ISO 19110 [8], and ISO 19107 [9] are used in these works. On the other hand, we propose an integration methodology focused on three main phases: *unit*, *integration* and *system*. Each phase takes advantage of the semantic of ontologies and their specific representation. This integration process is mainly based on our work in [5].

This paper is organized as follows: next Section describes the current Information Broker System. Section 3 presents the extension describing its architecture. In Section 4 we discuss some related work. Finally, future work and conclusions are discussed afterwards.

2 The Information Broker System

Distributed and overlapped information in APAT have motivated the construction of a federated system based on hydrological features. As the main goal of the project is to develop a system to provide a fully and user-transparent integration of the sources, in previous work [1, 2] we have introduced and implemented an Information Broker System based on a layered-based architecture. Figure 1 shows this architecture consisting of three main layers, *wrapper*, *federation* and *presentation*.

In the *Wrapper Layer*, Data Access Services (DASs) have been developed to wrap each available data source and to extract the information required on demand. Following, the *Federation Layer* offers a uniform and transparent access to the data stored in data sources through the *Query Processor* and the *Federated Schema Browser* components. The *Query Processor* performs the task of decomposing a global query in a set of local queries and integrating all the obtained results in a single response. The *Federated Schema Browser* provides

[4] http://www.isotc211.org/

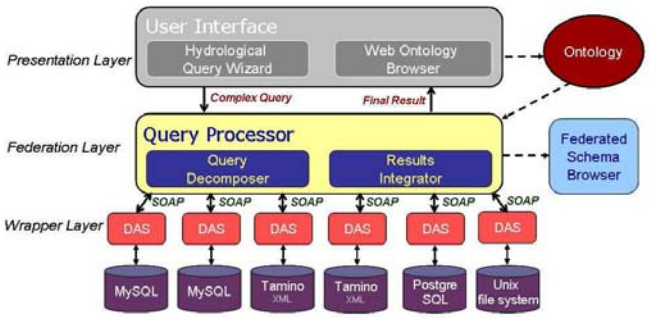


Fig. 1. The Information Broker Architecture

a high-level access to the federated schema, and is used by the query processor to discover the appropriate DAS which, in turn, provides access to a specific concept.

Finally, the *Presentation Layer* represents the communication medium between the broker and the end-users. It consists of two main components: the *User Interface* and the *Ontology*. Two different user-interfaces have been developed: an hydrological query wizard, used to perform global queries and view consequent results in a common web browser; and a web ontology browser, enabling users to navigate through the hydrological concepts (the ontology) within the APAT domain.

We have developed a first prototype of the Information Broker System [2]. This first release is composed of six databases managed by three distinct DBMS, namely MySQL Server, used for collecting real time measures; PostgreSQL server, used for collecting information on water quality; and Tamino XML Server, used for collecting data on extreme hydrological events and hydrography of the territory. Empirical evaluations about the use of this system are still outstanding.

In this paper, we are interested in one of the main processes to build the *Federated Schema* of the federation layer. Next sub-section describes some details of this process.

2.1 Building the Federated Schema

The federated schema is designed to provide a shared vocabulary of the information sources. Based on this vocabulary, we implement the user interface and the query processor components in order to give a global view of the whole system. In this way, the federated schema constitutes the core of the Information Broker system.

A bottom-up process consisting of four steps has been designed taking into account syntactic interoperabilities. Figure 2 shows graphically the components created within each step.

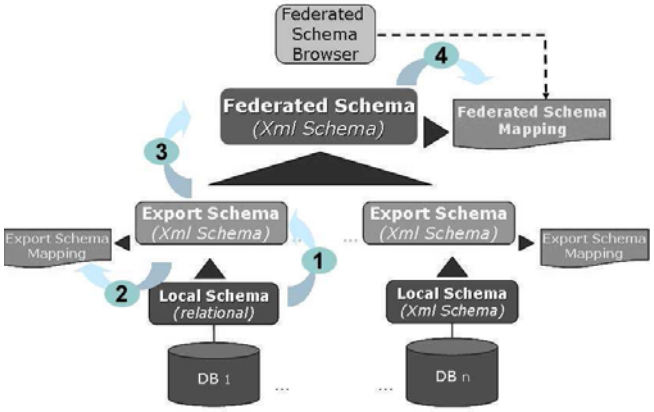


Fig. 2. Schema Integration Process

The first step transforms the local schemas into so-called export schemas, which are expressed in a common data model (CDM) and represented by XML data models. Thus, local schemas of the different databases of the federation converge on a common structure of data.

Then, the second step creates the export-schema mappings, which are XML files manually generated at design time from each export schema. Such files contain the mappings between the local and export schemas; that is, correspondences between low-level data and high-level domain concepts.

Finally, the third step builds the federated schema, which represents the logical model of the virtual database containing all data available within the federation. The federated schema is the result of merging all the export schemas.

During this merging, all possible conflicts must be identified and solved. This is accomplished through two different activities. The *Correspondence Investigation* activity searches for correspondences among the export schemas. The output of this activity is a set of conflicts, grouped in *naming conflicts* and *structural conflicts*. After that, the *Conflict Resolution* activity is carried out reviewing and fixing each conflict.

Once the federated schema has been generated, the last step in the process manually generates the federated-schema mapping file. It consists of an XML file that stores the correspondences between complex concepts and simple concepts distributed in the different export schemas; simple concepts and constraints that characterize them; and simple concepts and services able to retrieve them.

With respect to semantic aspects of the Information Broker System, we add a new component, called *Ontology Schema Mapping*, in order to represent the correspondences between concepts in the domain ontology and queries.

3 An Ontology-Based Extension for Generating a Federated Schema

Although the current Information Broker architecture is well suited for manipulating standard information through XML formatting rules, integration completely depends on users' interpretations and background. As we aforementioned, the task of building the federated schema is completely manual and in the case of large information sources (as we have to consider in this project) it becomes tedious and error-prone. Aspects as modificability and scalability were not taken into account because re-executing the integration process only for some changes on data can take several days.

In this way, the process of building the federated schema becomes difficult to standardize and evolve. Taking into account these two points we propose some changes on the general process of building the federated schema in order to facilitate the use of more suitable processes. The proposed extension is based on previous work [5, 6] in which an architecture and a merging process have been defined.

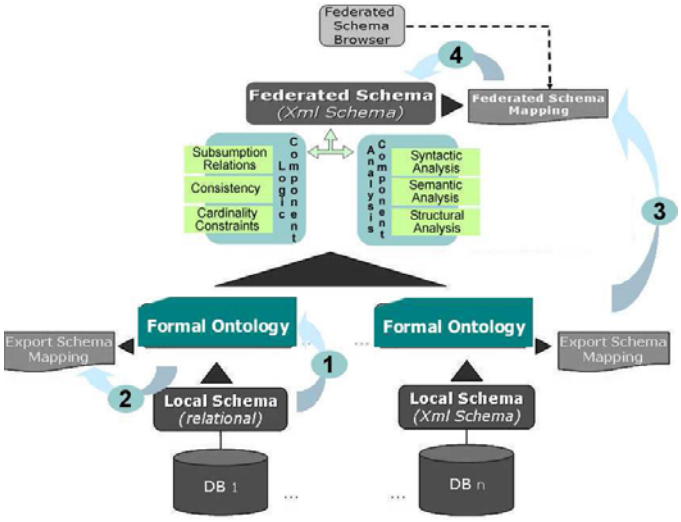


Fig. 3. Changes on the Schema Integration Process

Figure 3 shows the main changes made on the original Information Broker architecture. Like in the original schema (Figure 2), four bottom-up steps are necessary to build the federated schema. However, these steps are very different. The first and second steps, which were in charge of transforming local schemas into export schemas and generating export schema mappings, are now responsible for standardizing the geographic information of sources.

The third and fourth steps, which were in charge of creating the federated schema and its mappings, are now responsible for applying the method for merging ontologies.

In this way, the four steps are combined into two main processes, *enriching local ontologies* and *the merging process* itself. The first process defines the steps to create formal ontologies by applying the ISO 19100 standard for geographic information. Then, the *merging process* implements our merging method. Next two sub-sections provide a brief description of these processes.

3.1 Enriching local ontologies

The use of the ISO 19100 standard gives a new perspective to face integration problems for the interoperability of geographic systems. New ontology modelling techniques of this type of systems should be based on this standard in order to allow integration methods take advantage the benefits they provide.

In our extension, a *top-level ontology* and a *domain ontology* are built based on the information provided by the models of the standard (ISO 19109 and 19107 std.). Gray arrows in the Figure show how the information flows among the models. Thus, the domain ontology is built considering the General Feature Model (GFM) and the Application Schema [7]. The GFM is a meta-model of feature types. It defines the structure for classifying features used then to build the application schema. In the case of the top-level ontology, it is based on the structure of the GFM and the general features of the model being built.

Currently, there are new methodologies proposing the creation of ontologies such as [10, 11], including *Semantic Enrichment* as one of the most important steps. The main goal of this is to reconcile semantic heterogeneity, so it involves adding more semantic information about data. In our work, as both ontologies – top-level and domain – have to be based on the standard before being created, we add a new step in the process named *the enrichment step*. In this step, the components of the ontologies are enriched in their descriptions, through the metaclasses (from GFM) which they are instance of and the schemas on which they are based. In this way, all metaclasses extracted from the GFM and representing information by the application schema are created as abstract classes in the local ontology. Creating an ontology with these characteristics is not a complex task because the information needed with respect to the GFM can be extracted from the Feature Catalogue. Besides, by using an ontology editor as Protégé [5] to model OWL ontologies [12], ISO ontologies from `http://loki.cae.drexel.edu/~wbs/ontology/list.htm` can be imported.

[5] `http://protege.stanford.edu/`

Thus, all the ontologies will have the same structure due to all components are subclassifying the same model. The GFM acts as a top-level ontology classifying the elements of the ontology and making the integration easier. We will discuss this in the next sub-section.

3.2 The Merging Process

The merging process involves the task of merging the geographic sources in order to create a global vocabulary (federated schema) by defining two main components (Figure 3), *logic* and *analysis*. Both processes are used in different parts of the merging process.

This process is composed of three main phases: *unit*, *integration* and *system*. In the *Unit Phase* each system is analyzed separately. The top-level and domain ontologies can be seen as a unique ontology in which generalization / specialization relations are the connectors between them. This ontology will be formally represented by using OWL [12].

Then, once the ontologies are correctly created, a Reasoning System (such as RACER [13]) is applied in order to discover inferences not detected by users. We take advantage of the capability of inferring subsumption relations between classes and properties in the schema (TBox). That is, the reasoning system will determine where a concept can be located in a taxonomy hierarchy (a hierarchy built by means of a subconcept relation). Besides, the reasoner is used to check the consistency of the formal ontologies. Here, the validity of intentional definitions (in TBox) is checked. If an inconsistency is found, an expert user is responsible for solving it.

As result for each system, a normalized ontology (that can be divided into a top-level and a domain ontology) is returned. This ontology will be based on the geographic standards containing metaclasses descriptions (GFM) and the geographic schemas on which they are based. Thus, after passing through the logic process, the ontologies will have the correct structure we need to start with the following phase.

In the *Integration Phase* three processes are responsible for matching two normalized ontologies in order to create the global ontology. It contains the general concepts users will use to query the integrated system. In addition, a set of mappings are returned in order to represent the matching among the ontologies.

Merge, *General Analysis*, and *Specialized Analysis* are the processes of this phase. To do the first process, both ontologies of each system are joined by using generalization / specialization relations. In this way, the ontologies are taken as they are returned from the unit phase. Then, the two ontologies belonging to two different systems are merged. The merge process is performed by matching the classes that are part of the standard (metaclasses). As both ontologies have the same superclasses, merging is an easy task.

Once the merge process is finished, the *General Analysis* starts. It applies two types of analysis: syntactic and semantic. Within the syntactic analysis three syntactic functions are used in order to compare the names of the concepts in a

different way. Thus, functions return a different similarity result depending on the syntaxes of the compared names.

Then, in the semantic analysis, a thesaurus is used to extract synonym relationships between the concepts of the ontologies. These relationships are necessary because synonyms (in general) are not similar syntactically. In this case, WordNet[6] is used as the thesaurus. The *Specialized Analysis* performs a structural comparison by applying the similarity function described in [6, 14]. This function compares the number of properties that the classes have in common and analyzes them in a hierarchy (by calculating the depth of the most common superclass between the classes).

In the two last processes, user interaction is needed in order to determine the correct mapping. In this way, processes are user-driven and users are responsible for the final decisions.

Finally, it is possible the processes executed before generate inconsistencies within this final ontology. Therefore, the *System Phase* re-normalizes the global ontology created in the last phase. Like in the unit phase, a logic process is applied, where the reasoning system is used once more to analyze possible subsumption relations and inconsistencies in the global ontology.

User participation is also needed in this phase. Users here have two types of responsibilities – committing the options the reasoner system detects and testing the global ontology.

4 Related Work

Mapping discovery by using ontologies has being extensively investigated during the last years. Various approaches have emerged proposing processes and techniques to find similarities between elements of different but related ontologies.

In particular we are interested in methods for integration of geographic sources. In general, we can find three main overlapped mechanisms to perform integration, *the use of top-level ontologies*, *logical inferences* and/or *matching functions*. Table 1 shows the more representative and referenced proposals classified by these three types.

One particularity of all these proposals is the use of ontologies to represent either top-level information or domain information or both of them. In the case of ODGIS several ontologies are built (top-level, domain, and application ontologies) in order to provide more information about the domain and thus facilitate the integration process. But the activity of creating these ontologies is not an easy task and it demands a lot of effort. Other proposals as GeoNis, Aerts et al. and Hakimpour et al. use a top-level ontology together with the advantages of a formal language (to make inferences) as tools to find more suitable mappings. The use of similarity functions, in proposals as MDSM and SIM-DL, involves a set of functions that analyze the concepts and properties syntactically and semantically. In particular the use of these types of functions is useful when

[6] `http://wordnet.princeton.edu/`

Table 1. The three mechanisms for integration mapped to the proposals

	Top-level ontology	**Logical Inferences**	**Matching Functions**
BUSTER [15]		√	
Hakimpour et al. [16]	√	√	
MDSM [14]			√
ODGIS [17]	√		
GeoNis [18]	√	√	
Aerts et al. [19]	√	√	
Buccella et al. [5]	√	√	√
SIM-DL [20]			√

the ontologies are not complete (that is, there is absent information about the domain) and/or as starting point of an integration process when a top-level ontology is not involved. Proposals performing some manual step within the integration process require the assistance of an expert user to do so. For example, BUSTER needs of an expert user although it uses inferences during the query process.

Our merging method applies the three mechanisms to integrate ontologies. On one hand, top-level ontologies are created by using the information provided by the geographic standard. Then, logic capabilities and matching functions are combined in order to find more suitable mappings. The use of these three options makes our approach take advantage of the inherent benefits of using the standard in geographic information, the logic of data, and the semantic information from ontologies.

5 Conclusion and Future Work

In this work, we have presented an extension of the current Information Broker System in order to add capabilities which improve the generation of the federated schema. Particularly, our proposal aims at improving interoperability and consistency through the use of ontologies. However, there are still many issues that need further research. For example information sources in APAT Information Broker are not currently standadized, which may hinder consistency. The use of the ISO 19100 Stds. is a starting point for improving that. In addition, further validation of the ontology merging process would be absolutely neccessary for large ontologies – although our experiences [5] have shown good results when using small ones.

References

1. Calefato, F., Colagrossi, A., Gendarmi, D., Lanubile, F., Semeraro, G.: An information broker for integrating heterogeneous hydrologic data sources: A web services

approach. In Xu, A., Chaudhry, L., Guarino, S., eds.: Research and Practical Issues of Enterprise Information System, IFIP Series (Springer). Volume 205. (2006)
2. Gendarmi, D., Lanubile, F., Lichelli, O., Semeraro, G., Colagrossi, A.: Water protection information management by syntactic and semantic interoperability of heterogeneous repositories. In: Proceedings of the ISESS'07. (2007)
3. Gruber, T.: A translation approach to portable ontology specifications. Knowledge Acquisition **5**(2) (1993) 199–220
4. Baader, F., Calvanese, D., McGuiness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook - Theory, Implementation and Applications. Cambridge University Press (2003)
5. Buccella, A., Cechich, A.: Towards integration of geographic information systems. Electronic Notes in Theoretical Computer Science **168** (2007) 45–59
6. Buccella, A., Cechich, A., Brisaboa, N.R.: A three-level approach to ontology merging. In: MICAI'05. LNCS 3789, Monterrey, México, Springer-Verlag (November 2005) 80–89
7. : Geographic information. Rules for Application Schema. Draft International Standard 19109, ISO/IEC (2005)
8. : Geographic information. Geographic Information and Methodology for Feature Cataloguing. Draft International standard 19110, ISO/IEC (2005)
9. : Geographic information. Spatial Schema. International standard 19107, ISO/IEC (2003)
10. Belussi, A., Negri, M., Pelagatti, G.: An iso tc 211 conformant approach to model spatial integrity constraints in the conceptual design of geographical databases. In: ER (Workshops). (2006) 100–109
11. Jang, S., Kim, T.J.: Modeling an interoperable multimodal travel guide system using the iso 19100 series of international standards. In: Proceedings of the GIS '06, New York, NY, USA, ACM Press (2006) 115–122
12. Smith, M.K., Welty, C., McGuinness, D.: Owl web ontology language guide. W3C (February 2004)
13. Haarslev, V., Moller, R.: Racer system description. In Lambrix, P., Borgida, A., Lenzerini, M., Moller, R., Patel-Schneider, P., eds.: Proceedings of the CEUR-WS. Number 22, Linkoeping, Sweden (August 1999)
14. Rodríguez, M., Egenhofer, M.: Comparing geospatial entity classes: An asymmetric and context-dependent similarity measure. International Journal of Geographical Information Science **18**(3) (2004) 229–256
15. Visser, U.: Intelligent Information Integration for the Semantic Web. Volume 3159 of Lecture Notes in Computer Science. Springer Berlin - Heidelberg (2004)
16. Hakimpour, F.: Using Ontologies to Resolve Semantic Heterogeneity for Integrating Spatial Database Schemata. PhD thesis, Zurich University (2003)
17. Fonseca, F.: Ontology-driven Geographic Information Systems. PhD thesis, University of Maine (2001)
18. Stoimenov, L., Stanimirovic, A., Djordjevic-Kajan, S.: Discovering mappings between ontologies in semantic integration process. In: Proceedings of the AGILE'06, Visegr, Hungary (2006) 213–219
19. Aerts, K., Maesen, K., van Rompaey, A.: A practical example of semantic interoperability of large-scale topographic databases using semantic web technologies. In: Proceedings of the AGILE'06, Visegr, Hungary (2006) 35–42
20. Janowicz, K.: Sim-dl: Towards a semantic similarity measurement theory for the description logic *cnr* in geographic information retrieval. In: OTM Workshops (2). (2006) 1681–1692

Software Semantic Provisioning: *actually* reusing software

S. Sguera[a], A. Stellato[a], P. Ombredanne[b], M. T. Pazienza[a]

s.sguera@ieee.org
{pazienza, stellato}@info.uniroma2.it
philippe.ombredanne@eclipse.org

a: Università di Roma Tor Vergata, Dipartimento di Informatica, Sistemi e Produzione
b: The Eclipse Software Foundation

Abstract. Software development nowadays largely consists of adapting existing functionalities or components to perform in a new environment, and is biased towards delivering component-oriented architectures. Finding, choosing, provisioning and integrating the right libraries or components is still an ad-hoc and error prone task. This paper describes the SSP (Software Semantic Provisioning) project, funded in its early stages by Google™ Inc., developed during the Google Summer of Code™ 2007 program, and incubated by the Eclipse Software Foundation; the project aims to actually achieve software reuse in an effective, reliable and developer-friendly fashion, integrating cutting edge technologies in the component provisioning and integration areas, and providing support to decision-making in choosing the right dependencies set. A prototypical RESTful repository, and an Eclipse plug-in consuming the repository services have been implemented and will be discussed.

1. Introduction

Software development nowadays largely consists of adapting existing functionalities or components to perform in a new environment, and is biased towards delivering component-oriented architectures. Finding, choosing, provisioning and integrating the right libraries or components is still an ad-hoc and – thus – error prone task. Furthermore, it is sadly well know that object-oriented programming promised a lot about code reuse, but so far it never delivered it that much.

The problem of component provisioning, choosing the right software libraries set, and integrating it affects software developers and libraries providers. The impact of this is library choosing, component provisioning and integration tasks are carried out by developers, with little or no help at all.

The very general concept which lies behind software collection and reuse can be observed (in terms of needs) and applied (through successful methodologies and technical solutions) at very different level of specializations. While very general frameworks for software delivery and provisioning may offer services for accessing

and contributing to large library repositories, relying on dedicated metadata for organizing and retrieving the archived objects, there could be specific fields of interest where a more complex and organized description of the repository, tailored upon explicit needs and requirements which characterize the given domain, would improve the shareability of data, information and tools inside really active and participating communities.

Following previous research in the software components and libraries provisioning and integration by the ART group[1] at University of Rome Tor Vergata, this paper describes the SSP (Software Semantic Provisioning) project, funded in its early stage by Google™ Inc., developed during the Google Summer of Code™ 2007 program (details in [6]), and incubated by the Eclipse Software Foundation.

In Section 2 we will briefly introduce the main provisioning, build and integration support technologies currently available. Representative use case scenarios have been studied exploiting the prototypical implementation provided, and will be presented in Section 3, giving the reader a more thorough understanding of the surrounding environment and the actual benefits delivered to developers and component providers by the project. Section 4 will describe our approach, key goal and significant design issues. The software component domain has been formalized in the Software Provisioning Ontology (SWPO) whose main classes, properties and possible evolutions will be discussed in Section 5. Section 6 and 7 will be dedicated respectively to the discussion of architectural choices and issues we took both in server and client side development, while Section 8 will hold our conclusions and future directions of work and research.

2. State-of-the-art

A number of existing projects and efforts aim to describe software. Each one focuses upon a peculiar aspect, but no known product provides a thorough description enabling complex search and integration features. Hereafter we discuss the main characters populating the current component provisioning and integration panorama.

DOAP
The DOAP[2] (Description Of A Project) effort aims to describe a software project in terms of URI, maintainers, code repository and other product release-related features. No hints about what a given piece of software does or does not are given.

Maven
Maven[3] is one of the cutting edge integration and build management technology, and gained a significant market share in latest years. Its main goal is helping developers in

[1] http://ai-nlp.info.uniroma2.it
[2] http://usefulinc.com/doap
[3] http://maven.apache.org

handling dependencies and relieve the burden of integration and build process. The m2eclipse plug-in[4] allow developers to use POM files directly from the Eclipse[5] IDE.

Even if the folksonomy feature provided by the repository is quite functional and easy to use, and perfectly in line with the Web 2.0 hype, it does not provide a reliable mechanism to spot functional resemblance or more formal mappings and correspondence between components, as we propose in this paper.

OSGi Bundle Repository
OSGi[6] is the technology which enabled – among other things – the major shift in Eclipse's aims, from being a tooling platform (versions before 3.0) to a Rich Client Platform [3], and the subsequent changes in the requirements set, in terms of dynamic plug-in management, services, security, and performance. It provides an excellent platform for bundle provisioning and building dynamically extensible applications. A still evolving specification for building OSGi bundle repositories is given in [5].

Orbit
Orbit[7] mainly aims to reduce component duplication: it provides a repository of bundled versions of third party libraries that are approved for use in one or more Eclipse projects. It also clearly indicates the status of the library (i.e., the approved scope of use). Yet our aim is a bit more general, not simply attempting to reduce duplication, but collapsing – where possible – two or more libraries' functionalities in just a single one.

Buckminster
Buckminster[8]'s goal is to leverage and extend the Eclipse platform to make mixed-component development as efficient as plug-in development. It is very much focused on dependencies handling as well, while our approach is mainly aimed to improve components search and facilitate software reuse.

Kepler
The purpose of Kepler[9] is to address the complexities involved with provisioning, managing, and to use a shared infrastructure in order to support a community-oriented development model. The focus remains much tied to community-oriented development, more than component-oriented as in our effort.

Ivy
Ivy[10] is a project incubated by the Apache Software Foundation: it provides a tool for managing (recording, tracking, resolving and reporting) project dependencies. An

[4] http://m2eclipse.codehaus.org
[5] http://www.eclipse.org
[6] http://www.osgi.org
[7] http://www.eclipse.org/orbit/
[8] http://www.eclipse.org/buckminster/
[9] http://www.eclipse.org/proposals/kepler/
[10] http://incubator.apache.org/ivy/

interesting feature is transitive dependencies management: it shows simple inference capabilities, but no support for functionalities-driven smart search and reasoning, which characterize our approach, and are essential to us to enhance software reuse possibilities.

3. Main use cases and benefits

Despite the proliferation of provisioning systems and frameworks, the component search and choice activities are still carried out by developers with little or no help at all. Programmers are left to themselves scouting the web to find libraries and components, and no systematic approach nor thorough frameworks exist.

In the next paragraphs we will discuss some of the most representative use cases and the benefit they deliver to developers and components providers, stressing how our system tackles various aspects which currently undermine software reuse and often lead to write ex-novo already existing code.

Assert and spot functional equivalence between components

The number of components and libraries, along with their versions, makes practically impossible for a developer to know them all. On the other hand, there may exist more than a piece of software accomplishing the same task, fulfilling the same requirements set, or even implementing the same specification. To some extent, such components could be considered *functionally equivalent.*

This is the case, for instance, of Hibernate[11], Apache Cayenne[12] and all of the other frameworks implementing the Java Persistence API, or any implementation of the Java Servlet API, any JDBC driver, or any HTTP server (or client as well). The list would go a long way.

Furthermore, the equivalence is symmetrical, reflexive and transitive; the inference mechanism helps building relations upon social-generated contents: relations and functional equivalence among software components are both explicitly declared and inferred by the system, thus building a dense semantic network with a little effort. Machine-readable metadata allow much more granularity and raise the formal level and the *intelligence* of search-related features.

Let's suppose we just finished developing, for some obscure reason, a novel implementation of the Java persistence API. Let's suppose also that metadata about two common frameworks implementing the same API – i.e. Hibernate and Apache Cayenne – are already present in the repository, and (just as an example) that the two are declared as *functionally equivalent*. As we declare our library as equivalent to Hibernate, since they implement the same API, the inference engine can conclude my library is equivalent to Cayenne as well; Cayenne's mapping to our product is nowhere in the repository, but was just inferred. A developer looking for "Hibernate or equivalent" or "Cayenne or equivalent" libraries, or again "Java Persistence API

[11] http://www.hibernate.org
[12] http://cayenne.apache.org/

implementation" will then see our implementation among the query results, obtain information and in case decide to use it.

Find components providing a set of tasks
Describing a software component or library in terms of the tasks it fulfills is the very first way to tell whether a piece of software fits our needs or it does not. During the analysis and design phase developers must choose the right set of enabling technologies and components which will drive further development phases, and will construct the base for building our application's architecture.

Let's suppose – just as an example – we are planning to develop two components, one carrying out the *"dom-parsing"* task and the other fulfilling the *"sax-parsing"* task, and we would like to know if there is already a unique component providing both the tasks. It would be useful to browse the repository and discover at design time that *xerces-j* actually carries out both *sax* and *dom* xml parsing. We might then decide to use it if it fits our project's requirements.

Assessing reputation of components
Whenever a developing team picks up third-party code to underlie its application, it is implicitly taking responsibility someone else's code, which could affect their product's security and credibility. To this purpose, we could want to know which – and how many – components actually use one: this may give us valuable information about its reputation. On the other hand, if we developed a new component – and added it to the repository, it could be interesting to know which and how many components rely on our work.

4. Approach and design goals

Our key goal is to provide developers with a complete environment to exploit semantic metadata in order to effectively find and provision software components.

We tried to overcome the main limitations in current mainstream provisioning systems and frameworks, which are in turn tied to a particular technology or show a formalization level which grants no access to technology-independent, high level and enough granular information for a component.

Moreover, even if current provisioning technologies follow different approaches and stress different aspects proper of the software domain, there is a substantial overlap among the components' description they provide and rely upon.

Thus an ontology, meant to be a shared, higher level domain vocabulary among developers, allowing to semantically describe software and eventually mapping a subset of available metadata to one of the technologies available, would enable a thorough description of a component, aimed to stress *what* does the component do in an unambiguous fashion; this supports interoperability among developers and among technologies, provide some ground concepts to establish, declare or infer relationships among software components, and eases the reuse of existing software, giving developers a significant help in the early discovery phases.

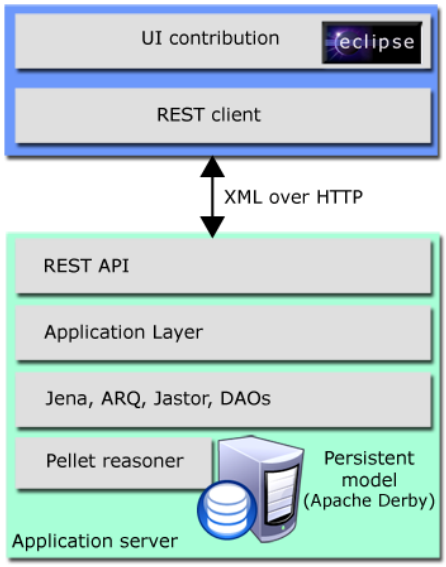


Figure 1: Server and Client side full stack architecture

A RESTful semantic repository (Figure 1), as it will be clearer in the next sections, easily allows the developing of a multitude of clients (i.e. browsers extensions, IDE plug-ins, et cetera), and broadens the field of possible applications.

5. Knowledge Model

The Knowledge Model of the SSP environment offers, at the current state of development, those concepts and relations which are necessary for providing a sufficiently detailed description of software entities and for modeling the functionalities which have been presented in the use-cases section.

Reference to past research work on modeling ontologies, like [4], for describing software systems has been made by reusing concepts from these ontologies for describing common software entities like: *component*, *library* and *software license*.

As it can be seen in Figure 2, our framework is centered about the description of software objects, providing several semantic anchors through which they can be identified, classified according to different perspectives and needs, and thus easily retrieved on these same aspects.

SoftwareObject(s) can be mainly distinguished according to two different categories: Components, which are "Program modules that are designed to interoperate with each other at runtime", that is software objects for which there is a well-defined runtime behavior, and Library(ies) which define "collections of subprograms used to develop software".

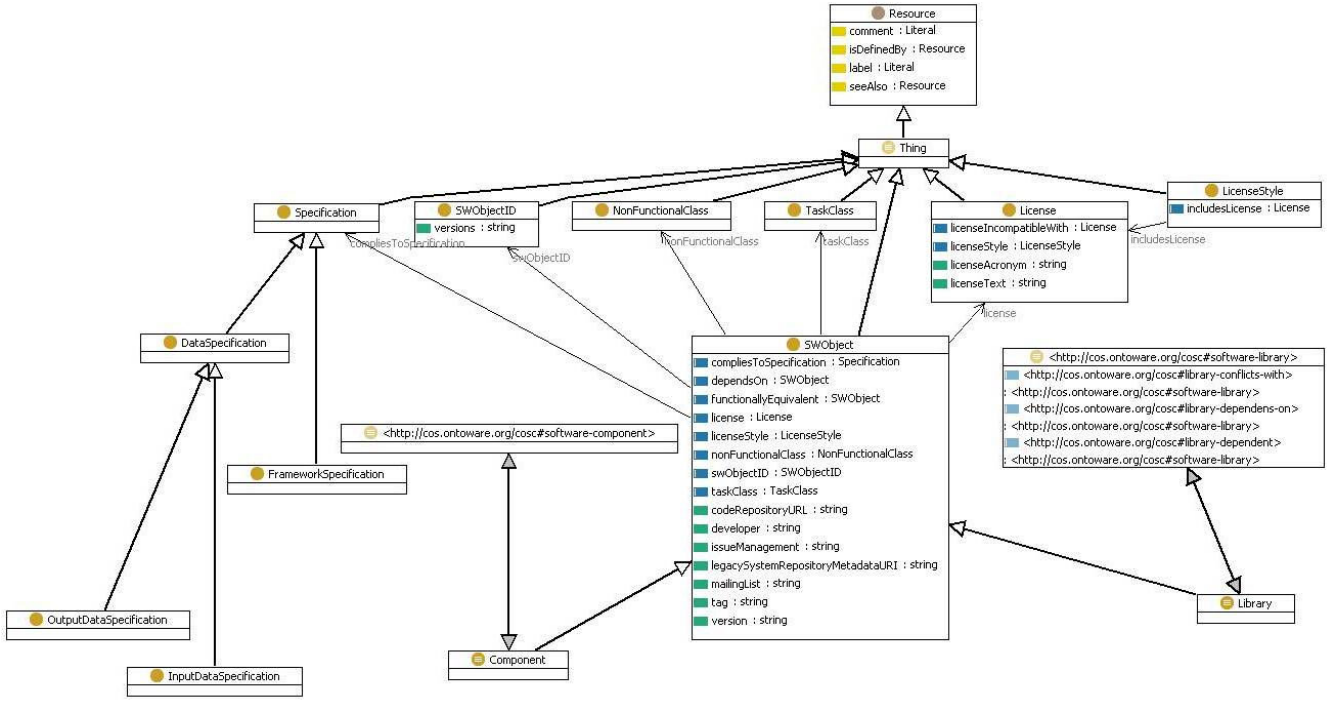


Figure 2: Knowledge Model of the SSP Framework

Other classes offer further perspectives over which software objects registered in the SSP repository may be clustered and accessed: License has been introduced to describe the diverse software licenses adopted by software developers and vendors. This way users may filter their choice if, as an example, they need only software licensed under a specific contract. This filtering can even less explicit, by automatic reasoning over class of licenses and the relationships between them. A property licenseIncompatibleWith allows to establish incompatibilities between use of components licensed under different contracts, while the class LicenseStyle describes categories of licenses which share common aspects. A reification technique (see [2] for a wider discussion on this topic) has been adopted to describe license styles both as objects of the domain as well as classes of licenses (so, as rdfs:subClassOf License), still remaining inside a first order description of the domain. This way we can "talk about" software license styles as ground objects (which may exhibit specific contractual expressions, have a reference web site for their general specifications etc…) and, at the same time, consider them as set of licenses, offering class level restrictions on the values that their belonging instances should expose on their properties. The explicit links between the objects (instances of LicenseStyle) and the set of Licenses (subclasses of License) is given by a restriction on a property which describes the specific style (if present) of any given license; the semantic repository thus automatically generates subclasses of License for each new introduced license style, together with their associated restriction.

With the same approach, it is possible to describe software with licenses according to a specific style, as for the following example:

$$\text{ApacheStyleLicensedSoftware} \equiv \exists \text{license.ApacheStyledLicense}$$

which describes (in description logic syntax) software distributed according to a license instantiating class ApacheStyledLicense, where this last is defined as:

$$\text{ApacheStyledLicense} \equiv \text{style} \ni \text{apacheLicense}$$

The same reification technique described above is used to automatically generate subcategories of SWObject which cluster sets of components and libraries according to their purposes, which are considered first class citizens inside the repository and not mere simple attributes for describing software. Specific Tasks can thus be defined in the repository and fully qualified according to their specifications and to descriptive information thought for human inspection; software objects can then be accessed, among the other ways, according to the task(s) they fulfill (e.g XML parsing, object persistence, text indexing etc…)

6. Server-side: the SSP Semantic Repository

The semantic repository publishes a set of REST API, in compliance to the well known architectural style described in [1] allowing clients to easily consume its services, and enabling any kind of Web 2.0 buzzword-compliant mashup. The RESTlet framework was embedded into a servlet container to deploy the repository as a web application.

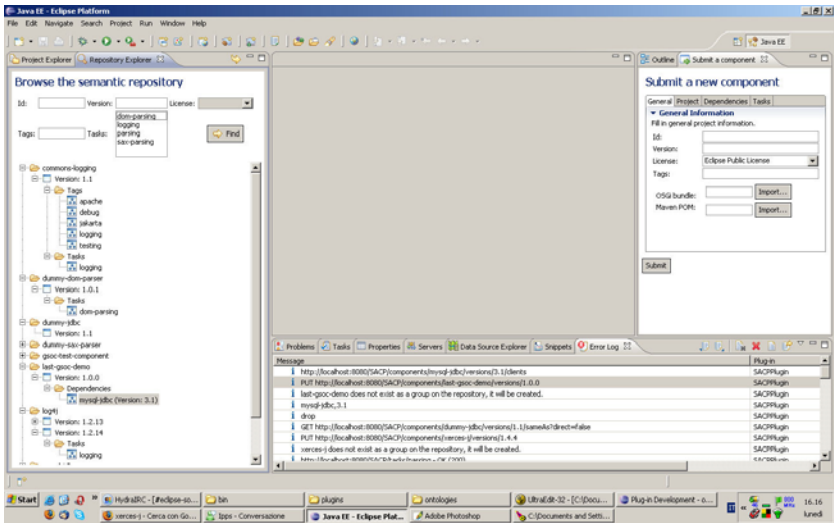


Figure 3: SSP Eclipse plug-in - UI contribution

Data serialization (beans to XML and vice versa) and complex services are handled by the application layer, while to access RDF triples stored in a persistent Jena model we took advantage of the IBM Jastor framework, providing OWL to Java mapping. Anyway, a further level of indirection was introduced not to tie the topmost layers to the specific technologies (i.e. Jastor) used in the data access layer. To enable inference-based web services we plugged Jena with the Pellet DIG reasoner.

7. Client-side: Eclipse SSP Plugin

We developed a RESTlet client consuming the repository's web services, decoupling the client-server interaction from the UI contributions.

The repository location can be both local (i.e. this can be achieved simply deploying the repository web application inside Eclipse itself, exploiting the embedded Jetty server used by the *help* plugin), or remote, and it can be chosen using the provided preference page, accessed in the usual Eclipse way.

Two views were implemented (Figure 3): the *Repository Explorer*, on the left, allows the developer to browse components by name, version, license, tags, tasks or navigate the semantic relations among the components; the *Submit a new component* view makes use of the Eclipse SWT Forms widgets to provide developers with an elegant and fast way to submit a new component to the repository. It is possible to define a component's dependencies, simply by dragging a component from the *Repository Explorer* on the left, and dropping it on the *Dependencies* tab in the component submission form, on the right. It is also possible to choose among the tasks already described in the repository, or add a new one throughout the submission process.

8. Conclusions and future works

In this paper we introduced a novel approach to software components and libraries discovery and provisioning. Indeed we believe current mainstream provisioning systems lack a shared vocabulary and technology-independent formalization of the software domain, supporting richer semantic description to support reasoning and the generation of a consensus based upon the specific domain the considered software belongs to.

Future iterations will involve a deeper axiomatization of *License* and *License-style* concepts, since they represent the contract between the product provider and the consumers, and often is a strict non-functional requirement to be satisfied when a third-party software is chosen. A strong investigation on "software specifications" could contribute to further discriminative arguments for facilitating classification (and thus more precise retrieval) of software objects in the repository. Integration with – and metadata reuse from – OSGi and Maven, and user interface improvements are top priorities for the project.

Acknowledgments

This work was funded by Google™ Inc. as part of the Google Summer of Code™ 2007 program, and developed by Savino Sguera mentored by Philippe Ombredanne – details in [6] – as a result of previous research work done in the area of software component provisioning by M. T. Pazienza, S. Sguera and A. Stellato at the ART research group at the University of Rome Tor Vergata.

We would like to thank Leslie Hawthorn and the whole Google Summer of Code™ team for the great job they did, and the Eclipse open source community for supporting the project and giving invaluable feedback throughout the development.

References

1. Fielding, R. (2000). Architectural Styles and the Design of Network-based Software Architectures, University of California Irvine, PhD Dissertation
2. Gangemi, A. & Mika, P. (2003). "Understanding the Semantic Web through Descriptions and Situations." Proceedings of the DOA/CoopIS/ODBASE 2003 Confederated International Conferences. LNCS 2888. Springer Verlag, 2003
3. Gruber, O., et al., 2005. The Eclipse 3.0 platform: Adopting OSGi technology, IBM Systems Journal, Vol 44, No 2, 2005
4. Oberle, D., Lamparter, S., Grimm, S., Vrandecic, D., Staab, S. Gangemi, A. Towards Ontologies for Formalizing Modularization and Communication in Large Software Systems Journal of Applied Ontology 1 (2): 163-202. 2006
5. OSGi RFC0112, 2005. http://www2.osgi.org/Download/File?url=/download/rfc-0112_BundleRepository.pdf
6. Sguera, S., 2007 http://code.google.com/soc/2007/eclipse/appinfo.html?csaid=1221666D7EBA3415

OWL-S Atomic services composition with SWRL rules

Domenico Redavid[1], Luigi Iannone[2], and Terry Payne[3]

[1] Dipartimento di Informatica, Università degli Studi di Bari
Campus Universitario, Via Orabona 4, 70125 Bari, Italy
{redavid}@di.uniba.it
[2] Computer Science Dept., University of Liverpool
Ashton Building, Ashton Street L69 3BX, Liverpool UK
{L.Iannone}@csc.liv.ac.uk
[3] School of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ, United Kingdom
{trp}@ecs.soton.ac.uk

Abstract. This paper presents a method for encoding OWL-S atomic processes by means of SWRL rules and composing them using a backward search planning algorithm. A description of the preliminary prototype implementation is also presented.

1 Introduction

Semantic Web (SW) aims at proposing standards, tools and languages for knowledge representation on the Web. Amongst the other issues, it deals with the provision of semantics to Web Services in order to achieve a more abstract and flexible automation. The result of this effort is the notion of Semantic Web Services (SWS) [1]. This term refers to traditional Web services that have been annotated by means of SW languages and techniques so as to make possible their automatic discovery, composition and invocation. In order to achieve that, in literature there are different approaches which produced different frameworks, among which the most widespread are OWL-S [2], WSMO [3] and WSDL-S [4].

In this paper we will focus on OWL-S as underlying language for annotating Web Services. OWL-S provides an ontological framework based on which an abstract description of a service can be created. It is an upper ontology whose root class is the *Service* class that directly corresponds to the actual service that is described semantically (every service that is described maps onto an instance of this concept). The upper level *Service* class is associated with three other classes: *ServiceProfile* (specifies the functionality of a service), *ServiceModel* (specifies how to ask for the service and what happens when the service is carried out) and *ServiceGrounding* (specifies how the service has to be invoked). In particular, the service model tells a client how to use the service, by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes. For nontrivial services (those composed of several steps over time), this description may be used by a service-seeking agent in different ways.

The *ServiceModel* defines the concept *Process* that describes the composition of one or more services in terms of their constituent processes. A *Process* can be atomic (a non-decomposable service), composite (a set of processes within some control structure that defines a workflow) or simple (a service abstraction).

In this paper our aim is the composition of OWL-S atomic processes adopting SWRL [5] as language for the representation of their IOPR (Inputs, Outputs, Preconditions and Results) models. Such SWRL descriptions are used as input to generate candidate service compositions in order to achieve a given goal.

The rest of the paper is organized as follows: in section 2 we report the basic notions of the OWL-S process model with some considerations on the guidelines that should be followed in order to have useful metadata for the Web services to be described. Section 3 identifies some requirements needed for encoding an OWL-S atomic process by means of SWRL rules. An algorithm for SWRL rules composition is described in section 4. In section 5 an application example that shows the applicability of our method is presented, while sections 6 and 7 are devoted to related work and conclusions, respectively.

2 Preliminary Considerations

In this section we report the basic notions about the OWL-S process model with some considerations on the guidelines that should be followed in order to have useful metadata for the Web services to be described.

Each OWL-S process [2] is based on an IOPR model. The *Inputs* represent the information that is required for the execution of the process. The *Outputs* represent the information that the process returns to the requester. *Preconditions* are conditions that are imposed over the *Inputs* of the process and that must hold for the process to be successfully invoked. Since an OWL-S process may have several results with corresponding outputs, the *Result* entity of the IOPR model provides a means to specify this situation. Each result can be associated to a result condition, called *inCondition*, that specifies when that particular result can occur. Therefore, an inCondition binds inputs to the corresponding outputs. It is assumed that such conditions are mutually exclusive, so that only one result can be obtained for each possible situation. When an *inCondition* is satisfied, there are properties associated to this event that specify the corresponding output (*withOutput* property) and, possibly, the *Effects* (*hasEffect* properties) produced by the execution of the process. *Effects* are changes in the state of the world.

The OWL-S conditions (*Preconditions*, *inConditions* and *Effects*) are represented as logical formulas. If needed, OWL-S provides some extra variables, called *ResultVars* and *Existentials*[4], that can be used in these formulas.

Formally, *Input* and *Output* are subclasses of the more general class *Parameter* declared in its turn as a subclass of *Variable* in SWRL ontology. Every parameter has a type, specified using a URI. Such type is needed to refer it to an entity within the domain knowledge of the service. The type can be either a *Class* or a *Datatype* (i.e.: a concrete domain object such as a string, a number, a date and so on) in the domain knowledge. Nevertheless, we argue that providing descriptions of Web services parameters using

[4] These entities appeared in OWL-S 1.2 Pre-Release, available at: http://www.ai.sri.com/daml/services/owl-s/1.2/

concrete datatypes gives very little in terms of added semantics. For example, consider the following declaration of the input in a process that retrieves books:

```
<process:Input rdf:ID="BookName">
    <process:parameterType
        rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</process:Input>
```

The fact that *process:parameterType* is declared as datatype means that the reference knowledge model of this input parameter is a concrete XML Schema datatype (string) instead of being an entity within a domain ontology. This mismatch becomes critical in automatic composition of services. Indeed, suppose that, during an hypothetical composition process, we need to find another service whose output will be fed into the service described above. Our composer, then, must necessarily consider those services that have as output a resource of the same type of our input parameter. In the example above, this type is string, hence every service that returns a string as an output can be composed with our service. Therefore, this would result in meaningless compositions of totally unrelated services due to the fact that parameters have been semantically poorly described. In the rest of this paper we consider only those services that have parameters (i.e. *Inputs* and *Outputs*) declared as entities in a domain ontology (i.e. not as datatype).

3 Encoding OWL-S atomic processes with SWRL rules

The aim of this section is to illustrate our approach for transforming process descriptions into sets of rules expressed in an ontology-aware rule language, namely Semantic Web Rule Language (SWRL). The motivation for doing this lies in the fact that, starting from this rule-based representation, an algorithm for detecting possible sequential composition of services (described in section 4) can be applied. SWRL [5] extends the set of OWL [6] axioms to include Horn-like rules [7]. The proposed rules are in the form of an implication between an antecedent (body) and consequent (head); both consist of zero or more conjunctive atoms. The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. An important characteristic of the rules is *safety*, i.e. only variables that occur in the antecedent of a rule may occur in the consequent. Furthermore, a rule with conjunctive consequent can be transformed into multiple rules each with an atomic consequent (Lloyd-Topor transformations [7]). A SWRL weakness is the non decidability of the whole language. A solution to this problem has been proposed in [8] where decidability is achieved by restricting application of SWRL rules to individuals explicitly introduced in the *ABox*. This kind of SWRL rules are called DL-safe.

In order to perform the transformation we impose some requirements on OWL-S process descriptions. It is necessary that all the entities within the process model are described in terms of a domain ontology. This means, basically, that Inputs and Outputs types, in order to satisfy this requirement, cannot be datatypes.

Within OWL-S, conditions (logical formulas) are either string literals or XML literals. The latter case is used for languages whose standard encoding is in XML, such

as SWRL. Body and head are logical formulas, whereby the OWL-S conditions can be identified with the body or with the head of a SWRL rule. Such conditions are expressed over *Input* and *Output*. Therefore, if the above requirement is met, conditions will be also expressed in terms of a domain ontology and will hence have the right level of abstraction.

After these considerations, we can describe the guidelines we follow for encoding an OWL-S process into SWRL.

- For every result of the process there exists an *inCondition* that expresses the binding between inputs variables and the particular result (output or effect) variables.
- Every *inCondition* related to a particular result will appear in the antecedent of each resulting rule, whilst the *Result* will appear in the consequent. An inCondition is valid if it contains all the variables appearing in the *Result*.
- If the *Result* contains an *Effect* composed of more atoms, the rule will be split into as many rules as the atoms are. Each resulting rule will have the same inCondition as antecedent and a single atom as consequent.
- The *preconditions* are conditions that must be true in order to execute the service. Since these conditions involve only the process *Inputs*, the corresponding SWRL rules should have the condition as antecedent and a boolean predicate that indicates whether the condition is true or not. In this work we consider always true all the *Preconditions*.

The first guideline is needed because there may be processes in which such binding is implicit in their OWL-S descriptions. Let us consider, for example, an atomic process having a single output. In this case there might be no *inCondition* binding inputs and output variables since, being the output the unique outcome, such binding is obvious. In this case, though, our encoding with SWRL rules would not be possible because the second guideline is not applicable. However, we can add a new *inCondition* that makes explicit such implicit binding.

For example, suppose we have a process declared only with the following *Parameters*, without *inCondition*:

```
<process:Input rdf:ID="BookName">
   <process:parameterType rdf:datatype="&xsd;#anyURI"> &kb;#BookTitle
   </process:parameterType>
</process:Input>

<process:Output rdf:ID="BookInfo">
   <process:parameterType rdf:datatype="&xsd;#anyURI"> &bibtex;#Book
   </process:parameterType>
</process:Output>
```

we should write the corresponding rule as follows:

```
kb:BookTitle(?process:BookName) → bibtex:Book(?process:BookInfo)
```

but the variable *process:BookInfo* does not appear in the antecedent of the rule (i.e. in the *inCondition*), consequently this is not a valid SWRL rule. Since every service

produces the output manipulating the inputs, we can suppose that there exists a predicate (*hasTransf* predicate) always true that binds every input to the output. In order to obtain valid rules, we add this predicate at antecedent of the rule:

```
kb:BookTitle(?process:BookName) ∧
kb:hasTransf(?process:BookName,?process:BookInfo) →
    bibtex:Book(?process:BookInfo)
```

including also the implicit *inCondition*.

4 A backward search algorithm for SWRL rules composition

In this section we present our SWRL composer prototype that implements a backward search algorithm for the composition task. It works as follows: it takes as input a knowledge base containing SWRL rules and a goal specified as a SWRL atom, and it returns every possible path built combining the available SWRL rules in order to achieve such goal. These rules comply with SWRL safety condition mentioned in the previous section.

In details, the algorithm performs backward chaining starting from the goal in the same fashion Prolog-like reasoners work for query answering. The difference is that this algorithm does not rely just on Horn clause but on SWRL DL-safe rules. This means that, besides the rule base, it takes into account also the Description Logic ontology to which the rules refer.

The SWRL rule path found, and consequently the resulting OWL-S service composition, will be valid, in the sense that it will produce results for the selected goal, only if the SWRL rules in the path are DL safe. In other words the DL-safety means that rules are true for individuals that are **known**, i.e.: they appear in the knowledge base[5]. At present, the prototype performs DL-safety check. This guarantees that the application of rules is grounded in the ABox and consequently that the services that embody those rules can be executed.

5 Example

In this section we present an example that shows the applicability of our method. The dataset of OWL-S services can be found on Mindswap Web site [6]. In such dataset, there are some OWL-S atomic services and, based on these ones, some OWL-S composite services. The latter set will be used to validate our method. We will evaluate how many composite services in such set can be actually built automatically by our composer.

In table 1 we report the set of the atomic services with the information needed for the scope of this section. Among them, only two services have not inputs and outputs described as datatype in knowledge domain and only one service contains a *Precondition*. All services have no declared *inConditions*, hence we assume that for each of them there is only one *Result* corresponding to the service output and there is no *Effect*.

[5] It might not be the case in general, given the Open World Assumption holding in Description Logics, see [8] and chapter 2 in [9]

[6] http://www.mindswap.org/2004/owl-s/services.shtml

To obtain SWRL rules that satisfy the requirements described in the section 3, we have modified the atomic services as follows:

a)

```
<!--namespace of this ontology is &kb;-->
..
<owl:Class rdf:ID="BookTitle">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="BookEntity"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:DatatypeProperty rdf:about="hasBookName">
  <rdfs:domain rdf:resource="#BookTitle"/>
    <rdfs:range rdf:resource="&xsd;#string"/>
  </owl:DatatypeProperty>
..
```

b)

```
..
<process:Input rdf:ID="BookName">
  <process:parameterType
    rdf:datatype="&xsd;#anyURI">
      &kb;#BookTitle
  </process:parameterType>
</process:Input>
..
```

Fig. 1. The transformation of a datatype in a knowledge domain entity

- For every parameter having a datatype as type, we created a class in the domain ontology having a datatype property with the corresponding datatype as range (fig. 1a). The OWL-S descriptions have been modified assigning the newly created class to the corresponding *parameterType* (fig. 1b).
- For each service, we create two logical formulas. The first composed of unary atoms having the *parameterType* URI as their predicate and the input as their variable, for each input. The second composed of a unary atom having the *parameterType* URI as its predicate and the output its variable. We set these two logical formulas as, respectively, the antecedent and consequent of a new SWRL rule.
- Since every service produces the output manipulating the inputs, we can suppose that there exists a predicate (*hasTransf* predicate) always true that binds every input to the output. We did this in order to guarantee the SWRL safety condition, then we added *hasTransf* predicates to the antecedent of the rule built in the previous step. With this modification the antecedent can be identified with a new *inCondition*.

The obtained SWRL rule set is given as input to our composer and the resulting composition can be compared with the processes proposed in the Mindswap composite services examples. However, OWL-S composite processes can use control constructs (such as *iteration* and *selection*) that are more complex than the simple sequence, hence some considerations are needed w.r.t. composed services in 2:

- The French Dictionary service returns the meaning of a French word in French. To do this, it uses the processes of two atomic services: BabelFishTranslator and English Dictionary. It defines the sequences reported in the column 2 of the table 2 that are combined by means of other control constructs to return its result.
- Find Cheaper Book Price service returns the smallest price of a book along with the name of the bookstore that sells it. To do this, it uses the processes of three

ATOMIC SERVICE	DATATYPE INPUTS	DATATYPE OUTPUTS	SWRL CONDITION
Book Finder: Returns the information of a book whose title best matches the given string.	"BookName"	No	No
Zip Code Finder: Returns the zip code for the given city/state.	"City", "State"	No	No
Latitude Longitude Finder: Returns the latitude and longitude for a given zip code.	No	No	No
Barnes & Nobles Price Finder (BNPrice): Returns the price of a book as advertised in Barnes and Nobles web site given the ISBN Number.	No	No	No
Amazon Book Price Finder (AmazonPrice): Returns the price of a book as advertised in Amazon web site given the ISBN Number.	No	No	No
English Dictionary: Returns the meaning of a word from the dictionary.	"InputString"	"OutputString"	No
BabelFish Translator: Convert text from one language to another language using the online BabelFish translator services.	"InputString"	"OutputString"	One Precondition: "SupportedLanguagePair"
Currency Converter: Converts the given price to another currency.	No	No	No

Table 1. Some characteristics of the OWL-S atomic services datataset

atomic services: BookFinder, BNPrice and AmazonPrice. It defines the sequences reported in the column 2 of the table 2 that are combined by means of *selection* control construct to return its results.

As mentioned above, our composer is not able to work with complex control sequences and composite services as inputs. In both cases, we have to check if our composer was able to retrieve the basic sequences reported above on the basis of which the composite services have been built. In order to verify this, in Tab. 2 we put as input of our composer the outputs of the sequences instead of the outputs of the services.

COMPOSITE SERVICE	SEQUENCES IN THE SERVICE	RETRIEVED? (Yes/No)
French Dictionary	1) BabelFishTranslatorProcess ⇒ EnglishDictionaryProcess 2) EnglishDictionaryProcess ⇒BabelFishTranslatorProcess	1) Yes 2) Yes
Book Price	1) BookFinderProcess ⇒ BNPriceProcess ⇒ CurrencyConverterProcess	1) Yes
Find Cheaper Book Price	1) BookFinderProcess ⇒ BNPriceProcess 2) BookFinderProcess ⇒ AmazonPriceProcess	1) Yes 2) Yes

Table 2. Some characteristics of the OWL-S composite services datataset

BookPrice service returns the price of a book in a desired currency. To do this, it uses the processes of three atomic services: Book Finder, BNPrice and Currency Converter. Since it uses only the sequence construct, the searched goal can be the output of the service and our system retrieves the correct composition.

6 Related work

To the best of our knowledge no approach in literature makes use of SWRL for the composition of Semantic Web Services. Researchers focussed either on semi-automated or fully automated methods for service composition, drawing inspiration especially from AI planning [10] and state machines [11].

One approach aims at integrating Semantic Web formalisms into classical planner methodologies. Berardi et al. [12] address the problem of automatic composition synthesis of e-Service. They developed a framework in which the exported behavior of an e-Service is described in terms of its possible executions (execution trees). Then they specialize the framework to the case in which such exported behavior (i.e., the execution tree of the e-Service) is represented by a finite state machine. In [13], the semantics underlying the DAML-S specification (the ancestor of OWL-S) has been translated into FOL, obtaining a set of axioms for describing the features of each service. By combining these axioms within a Petri Net, the authors have obtained process-based service models that enable reasoning about the interactions among the processes that form the structure of a service. Traverso and Pistore [14] propose a planning technique for the automated composition of Web services described in OWL-S process models, which can deal with nondeterminism, partial observables, and complex goals. Such technique facilitates the synthesis of plans that encode compositions of web services with the usual programming constructs, like conditionals and iterations. In [15] an approach for developing a Semantic Web service discovery and composition framework on top of the CLIPS rule-based system is presented. More specifically, it describes a methodology for using production rules over Web services semantic descriptions expressed in the OWL-S ontology.

Other approaches, in which our methodology can be framed, apply methodologies and tools developed in the field of AI planning directly on Semantic Web settings. Sirin and Parsia [16] demonstrate how an OWL reasoner can be integrated within an AI planner, called SHOP2 [17], for the composition of Semantic Web Services. The reasoner is used to store the world states, answer the planners queries regarding the evaluation of preconditions, and update the state when the planner simulates the effects of services. An approach for using SPARQL [18] as an expression language for OWL-S conditions is presented in [19]. It describes how SPARQL can be used to give a compact representation of the preconditions of a service, and of its results. To our knowledge there hasn't been any attempt to use SPARQL query engines for achieving service composition.

The first type of approach foresees a translation from the Semantic Web formalisms to a dedicated formalism so that tools developed in particular research areas can be applied maintaining the same performances. On the contrary, the second type of approach foresees a porting of the algorithms and methodologies from other research fields using the Semantic Web technologies. The advantage of this approach, in which we frame our methodology, is the direct use of the Semantic Web formalisms. In this manner, we are able to use methodologies coming from more consolidated research fields exploiting the advantages that Semantic Web guarantees, i.e. a distributed knowledge base and the semantic interoperability. Furthermore, the use of SWRL, in particular, allowed us to exploit its greater expressiveness with respect to OWL-DL itself. Indeed, OWL-DL, being a Description Logic, does not allow to formulate constructs like property

compositions without becoming undecidable. SWRL partially reliefs these constraints, especially in the fragment we adopted in our work (i.e.: DL-safe SWRL rules), providing a more powerful means that becomes very useful in most SWS description portions such as *inConditions* and *Effects*.

7 Conclusions and future work

In this paper we have presented a new method that exploits SWRL for OWL-S atomic services composition. We have proved that if the OWL-S services have a meaningful semantics and valid SWRL conditions it is possible to build composer exploiting only the Semantic Web technology to achieve the composition task. This work can be considered as a starting point for the solution of a broader issue like the orchestration of SWS. Indeed rules for service coordination can be added to the rules for SWS encoding completing the knowledge that is necessary for the orchestration.

Future work will mainly consist of augmenting the number of services that can be encoded into SWRL rules. In other words the system should be able in the future to handle composite services as input and to produce more complex control structures (such as selection and iteration). The latter seems to be the most challenging task since it will require more powerful algorithms for the composition task.

Furthermore, an interesting aspect to deal with is the management of knowledge bases when there are changes produced by the effects of a service execution. Semantic Web languages are based on Description Logics which implement monotonic reasoning. In other words, they do not provide any means for retracting or modifying the status of the knowledge base that is not adding some new facts. This is somewhat a too restrictive requirement to represent, for instance, service execution in such formalisms. Think for example of representing the status of an activity as a functional property [7]. Now, as soon as this activity changes its status (say, for instance, it passes from 'scheduled' to 'in progress'), an equivalent change should be carried out on its description in the knowledge base. At the moment there is no DL reasoner allowing for that, meaning that performing such change in a traditional knowledge base would lead to an inconsistency of the whole knowledge base.

References

[1] McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems **16** (2001) 46–53

[2] OWL-S: Semantic markup for web services, http://www.w3.org/submission/owl-s/ (2004)

[3] WSMO: Web service modeling ontology d2v1.3, http://www.wsmo.org/tr/d2/v1.3/ (2006)

[4] WSDL-S: Web service semantics, http://www.w3.org/submission/wsdl-s/ (2005)

[5] Horrocks, I., Patel-Schneider, P.F., Bechhofer, S., Tsarkov, D.: OWL rules: A proposal and prototype implementation. J. of Web Semantics **3** (2005) 23–40

[6] OWL: Owl web ontology language overview, http://www.w3.org/tr/owl-features/ (2004)

[7] Lloyd, J.W.: Foundations of logic programming (second, extended edition). Springer series in symbolic computation. Springer-Verlag, New York (1987)

[7] a property allowing only one value for each instance it is applied to.

[8] Motik, B., Sattler, U., Studer, R.: Query answering for owl-dl with rules. Journal of Web Semantics: Science, Services and Agents on the World Wide Web **3** (2005) 41–60
[9] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook. Cambridge University Press (2003)
[10] Georgeff, M.P.: Planning. In Allen, J., Hendler, J., Tate, A., eds.: Readings in Planning. Kaufmann, San Mateo, CA (1990) 5–25
[11] Gurevich, Y.: Evolving algebras 1993: Lipari Guide. In Börger, E., ed.: Specification and Validation Methods. Oxford University Press (1994) 9–37
[12] Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., Mecella, M.: Automatic composition of transition-based semantic web services with messaging. In Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P.Å., Ooi, B.C., eds.: VLDB, ACM (2005) 613–624
[13] Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, ACM Press (2002) 77–88
[14] Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: International Semantic Web Conference. Volume 3298 of Lecture Notes in Computer Science., Springer (2004) 380–394
[15] Meditskos, G., Bassiliades, N.: A semantic web service discovery and composition prototype framework using production rules. In OWL-S:Experiences and Directions Workshop at 4th European Semantic Web Conference (ESWC) (2007)
[16] Sirin, E., Parsia, B.: Planning for Semantic Web Services. In Semantic Web Services Workshop at 3rd International Semantic Web Conference (2004)
[17] Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: Shop2: An htn planning system. J. Artif. Intell. Res. (JAIR) **20** (2003) 379–404
[18] Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (Candidate Recommendation). Technical report, W3C (2007)
[19] Sbodio, M.L., Moulin, C.: SPARQL as an expression language for OWL-S. In OWL-S:Experiences and Directions Workshop at 4th European Semantic Web Conference (ESWC) (2007)

The JUMP project: domain ontologies and linguistic knowledge @ work

P. Basile, M. de Gemmis, A.L. Gentile, L. Iaquinta, and P. Lops

Dipartimento di Informatica
Università di Bari
Via E. Orabona, 4 - 70125 Bari - Italia
{basilepp,degemmis,al.gentile,l.iaquinta,lops}@di.uniba.it

Abstract. The JUMP project aims at bringing together the knowledge stored in different information systems in order to satisfy information and training needs in knowledge-intensive organisations. Electronic Performance Support Systems provide help, advices, demonstrations, or any other informative support that a user needs to the accomplishment of job tasks in her day-to-day working environment. The paper describes the JUMP framework, which is designed to offer multiple ways for the user to query the knowledge base resulting from integration of autonomous legacy systems. Semantic Web languages and technologies are used throughout the framework to represent, exchange and query the knowledge, while Natural Language Processing Techniques are implemented to understand natural language queries formulated by the user and provide consistent and satisfying results.
Keywords: Semantic Web, Interoperability, Word Sense Disambiguation, Entity Recognition

1 Introduction

The JUMP project [1] aims at developing an EPSS *(Electronic Performance Support System)* capable of intelligent delivery of contextualized and personalized information to knowledge workers acting in their day-to-day working environment on non-routinary tasks. While generic queries can be easily fulfilled by means of standard information retrieval tools, such as general purpose search engines, the scenario is more difficult if the search goal concerns grey information stored in various forms and spread in different company knowledge bases, managed by different applications, all running within the company intranet. It is the case of users knowledgeable w.r.t. the IT infrastructure and that already have the background knowledge necessary to achieve most of the task they are involved in, but not being expert of all the domains in which the task to be achieved spans. Tasks of this kind are neither generally codified in corporate

[1] JUst-in-tiMe Performance support system for dynamic organizations, co-funded by POR Puglia 2000-2006 - Mis. 3.13, Sostegno agli Investimenti in Ricerca Industriale, Sviluppo Precompetitivo e Trasferimento Tecnologico

procedures nor completely new to the worker. Above all, those tasks are by no means solvable, in terms of information retrieval, by a standard Internet search. Any brute-force approach like Google desktop search can solve the problem in this case and, even if it could, the result would never take into account the connections existing between the various sources. An EPSS aims at supporting information needs spanning through multiple knowledge bases, namely all the available information systems in the company, be them formalized or not, including binary documents such as video or audio streams. It acts as an agent gluing together the different sources by means of semantic connections, and provides the user with contextualized and personalized information tied to both the task being accomplished and to her characteristics. On the basis of an accurate and formalized description of user's features and of those of the software tool she is using, as well as the textual information describing the task being accomplished (for example, the text of an e-mail just received), the EPSS should select relevant items from the KBs, ranking them according to the user profile, and provide them in a list to the user who will eventually give a feedback about the relevance of the provided information. The JUMP system has been designed to achieve this goal by means of a centralized recommendation system that takes advantage of a shared ontology describing the various knowledge bases and advanced Natural Language Processing (NLP) techniques to handle natural language requests.

The paper is organized as follows: after giving a general description of JUMP framework focusing on abstract layers of its architecture and the underlying shared ontology, in section 3 we give a detailed description of the Content Analyzer Module encapsulated in the framework, which provides NLP tools. In section 4 we argument the project prototype and conclusions to work, anticipating possible future work.

2 JUMP: Ontology-centric Architecture

The system general architecture is depicted in Figure 1, where the central component (JUMP-EPSS) acts as a hub of many autonomous peripheral systems. The involved systems in the current implementation are a Human Resources Managements system (HRMS), an Enterprise Resource Planning (ERP), a Document Management system (DMS) and a Learning Management system (LMS), but the design of the platform is such that new systems can be added as they become available.

2.1 Modularized Design

The basic design idea of the JUMP project is to encourage the loosely coupling among framework components according to a Service Oriented Architecture perspective so that the framework has the ability to seamlessly add information sources or peripheral systems, each one based on different technologies, programming languages and knowledge representation metaphors. This has lead to adopting standard languages and protocols when designing the interfaces that

each of the systems participating in JUMP has to implement in order to expose search services. The communication level between JUMP and the ancillary sys-

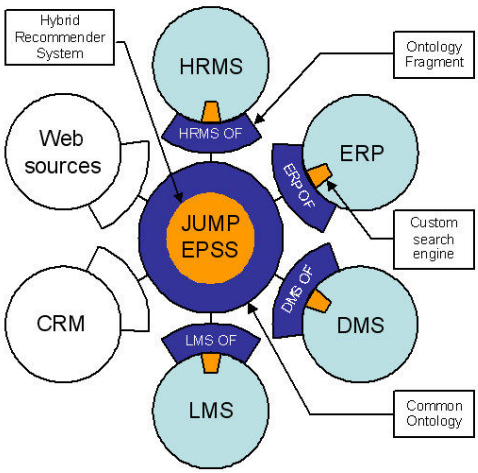


Fig. 1. Sketch of the JUMP system architecture

tems is designed to exchange both metadata about relevant items stored in the subsystems and the items themselves. While the items considered here (which are the results JUMP can present to the user) are generic binary objects ranging from email addresses to audio/video streams, the metadata about them are expressed through Semantic Web technologies; to make this possible, some OWL [2] ontologies about the items have been created, in order to structure specific domain knowledge and instantiate resources to describe the stored items.

2.2 Ontologies in the JUMP project

An ontology, following Grubers widely accepted definition [6], is a shared formalization of a conceptualization. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. In order to define an ontology it is necessary to choose a formalism, to use this formalism to encode the conceptualization that the applications are going to use, and to make this conceptualization shared, i.e. ensure that the ontology is used consistently by all the systems involved. To define the JUMP ontology we adopted OWL as representation language and Description Logics [1] was consequently adopted as the underlying formalism. Separate ontology fragments have been handcrafted using the Protege editor [3]

[2] http://www.w3.org/2004/OWL/
[3] http://protege.stanford.edu

in order to represent the most important concepts inside each of the involved systems. Ontologies have been then designed bottom-up in order to reflect the semantics of the underlying databases and coded functional processes as much as possible, but not aiming at a total ontological replication of the knowledge bases. After developing the single Ontology Fragments (OF), they have been divided into system specific ontologies and upper ontologies; these upper ontologies are the part of the OFs that the JUMP system should use when formulating queries for the subsystems. The Shared Ontology (SO) is therefore the union of all the upper OFs plus all the relations and concepts that are specific to the JUMP system; since some concepts are repeated across systems, the creation of the SO is the point in which alignment techniques have to be used in order to simplify and generalize the query writing phase of the search. The concept in SO are annoted using lexical concept that are exploited by Word Sense Disambiguaiton algorithm described in Section 3.1. The Ontology Fragments are aligned manually using the concepts in the Shared Ontology. The single Ontology Fragments are populated automatically creating a mapping between each legacy system's DBMS and each fragment.

2.3 JUMP EPSS Core: External and Internal Interactions

JUMP architecture has two abstract layers, as showed in figure 2, logically organized as a stack:

- *User Interface Layer*
- *Application Domain Layer*

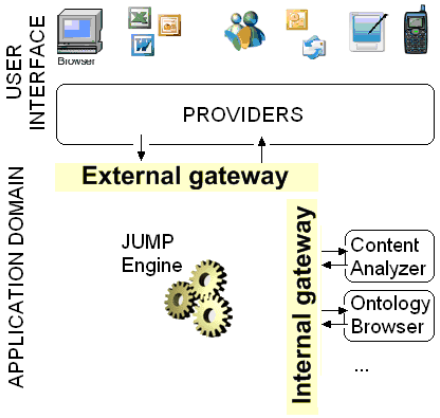


Fig. 2. JUMP Engine and Framework Layers

The *User Interface Layer* has been designed to support both online and offline client. It supports several devices: we handle both *untrusted devices*, such

as web browser or Office tools (Word/Excel), for which we predisposed a login procedure and *trusted devices*, such as mobile devices (e.g. smart phone) or email client for which the system is able to identify the user without any login procedure. The application domain layer has a software layer, namely *external gateway*, that communicates with all user devices through their specialized provider. The *JUMP Engine* elaborates all requests making use of specialized modules to resolve specific requests (e.g. plain text is sent to *Content Analyzer Module*, described in Section 3, in order to capture its semantics). Communication between the *JUMP Engine* and all specialized modules is implemented through a software layer, namely *internal gateway*.

3 NLP Processes in the Content Analyzer Module

Since user's requests are formulated by using natural language, Natural Language Processing (NLP) techniques are adopted in order to covert the original requests into an internal representation processable by the JUMP system. The *Content Analyzer Module* is devoted to this task: it extracts relevant concepts from the text describing the request and build an internal data structure called Bag-Of-Concepts (BOC). The goal is to include semantics in the process and to overcome well-known problems in text processing, such as polysemy, due to the use of keywords. The BOC structure contains two type of concepts:

1. relevant *linguistic* concepts recognized in the text by a Word Sense Disambiguation process [8] exploiting external linguistic knowledge-bases such as the WordNet lexical database [9];
2. relevant *domain* concepts extracted by a Named Entity Recognition (NER) process. The NER process is guided by JUMP ontology.

The implemented NLP process also includes operations preliminary to the BOC extraction step, such as:

- Text normalization: the original text is modified to prepare it for the following steps (for example, all formatting characters are removed);
- Tokenization: it is the process of split up input a string into tokens;
- Stop words elimination: all commonly used words are deleted;
- Stemming: it is the process of reducing inflected (or sometimes derived) words to their stem. In our project we adopt the *Snowball stemmer* [4];
- POS-tagging: it is the process of assign a part-of-speech to each token. We develop a JAVA version of *ACOPOST tagger* [5] using Trigram Tagger T3 algorithm. It is based on Hidden Markov Models, in which the states are tag pairs that emit words;
- Lemmatization: it is the process of determining the lemma for a given word. We use WordNet Default Morphological Processor (included in the WordNet

[4] http://snowball.tartarus.org/
[5] http://acopost.sourceforge.net/

distribution) for English. For the Italian language, we have built a different lemmatizer that exploits the *Morph-it!* morphological resource [6].

As final output each word in the original document is enriched with syntactic and semantic information collected during all the steps. In the two following subsections (3.1 and 3.2) we provide more details about the process of BOC extraction process.

3.1 JIGSAW: Word Sense Disambiguation

The goal of a WSD algorithm consists in assigning a word w_i occurring in a document d with its appropriate meaning or sense s, by exploiting the *context* C in where w_i is found. The context C for w_i is defined as a set of words that precede and follow w_i. The sense s is selected from a predefined set of possibilities, usually known as *sense inventory*. In the proposed algorithm, the sense inventory is obtained from WordNet. JIGSAW is a WSD algorithm based on the idea of combining three different strategies to disambiguate nouns, verbs, adjectives and adverbs. The main motivation behind our approach is that the effectiveness of a WSD algorithm is strongly influenced by the POS tag of the target word. An adaptation of Lesk dictionary-based WSD algorithm has been used to disambiguate adjectives and adverbs [2], an adaptation of the Resnik algorithm has been used to disambiguate nouns [10], while the algorithm we developed for disambiguating verbs exploits the nouns in the *context* of the verb as well as the nouns both in the glosses and in the sentence examples that WordNet utilizes to describe the usage of a verb. JIGSAW takes as input a document $d = (w_1, w_2, \ldots, w_h)$ and returns a list of WordNet synsets $X = (s_1, s_2, \ldots, s_k)$ in which each element s_i is obtained by disambiguating the *target word* w_i based on the information obtained from WordNet about a few immediately surrounding words. We define the *context* C of the target word to be a window of n words to the left and another n words to the right, for a total of $2n$ surrounding words. The algorithm is based on three different procedures for nouns, verbs, adverbs and adjectives, called $JIGSAW_{nouns}$, $JIGSAW_{verbs}$, $JIGSAW_{others}$, respectively. A short description of procedures $JIGSAW_{nouns}$ and $JIGSAW_{verbs}$ follows, more details about all the procedures and experiments are reported in [3].

***JIGSAW*$_{nouns}$** The procedure is obtained by making some variations to the algorithm designed by Resnik for disambiguating noun groups. Given a set of nouns $W = \{w_1, w_2, \ldots, w_n\}$, obtained from document d, with each w_i having an associated sense inventory $S_i = \{s_{i1}, s_{i2}, \ldots, s_{ik}\}$ of possible senses, the goal is assigning each w_i with the most appropriate sense $s_{ih} \in S_i$, according to the *similarity* of w_i with the other words in W (the context for w_i). The idea is to define a function $\varphi(w_i, s_{ij})$, $w_i \in W$, $s_{ij} \in S_i$, that computes a value in $[0, 1]$ representing the confidence with which word w_i can be assigned with sense s_{ij}. $JIGSAW_{nouns}$ differs from the original algorithm by Resnik in several ways.

[6] http://sslmitdev-online.sslmit.unibo.it/linguistics/morph-it.php

First, in order to measure the relatedness of two words we adopted a modified version of the Leacock-Chodorow measure [7], which computes the length of the path between two concepts in a hierarchy by passing through their *Most Specific Subsumer* (MSS). In our version, we introduced a constant factor *depth* which limits the search for the MSS to *depth* ancestors, in order to avoid "poorly informative MSSs". Moreover, in the similarity computation, we introduced both a Gaussian factor $G(pos(w_i), pos(w_j))$, which takes into account the distance between the position of the words in the text to be disambiguated, and a factor $R(k)$, which assigns s_{ik} with a numerical value, according to the frequency score in WordNet (more importance is given to the synsets that are more common than others). This algorithm considers the words in W pairwise. For each pair (w_i,w_j), the most specific subsumer MSS_{ij} is identified, by reducing the search to *depth* ancestors, at the most. Then, the similarity $\textsc{Sim}(w_i, w_j, depth)$ between the two words is computed. MSS_{ij} is considered *as supporting evidence* for those synsets s_{ik} in S_i and s_{jh} in S_j that are descendants of MSS_{ij}. The amount of support contributed by the pairwise comparison is the similarity value $\textsc{Sim}(w_i, w_j, depth)$, weighted by a gaussian factor that takes into account the position of w_i and w_j in W (the shorter is the distance between the words, the higher is the weight). The value $\varphi(i, k)$ assigned to each candidate synset s_{ik} for the word w_i depends on both the amount of support it received and a factor that takes into account rank of s_{ik} in WordNet, i. e. how common sense s_{ik} is for the word w_i. The synset assigned to each word in W is the one with the highest φ value. More details about both the procedure and the computation of the similarity value are reported in [3].

***JIGSAW*$_{verbs}$** We define the *description* of a synset as the string obtained by concatenating the gloss and the sentences that WordNet uses to explain the usage of a synset. First, $JIGSAW_{verbs}$ includes, in the context C for the target verb w_i, all the nouns in the window of $2n$ words surrounding w_i. For each candidate synset s_{ik} of w_i, the algorithm computes $nouns(i, k)$, that is the set of nouns in the description for s_{ik}. Then, for each w_j in C and each synset s_{ik}, the following value is computed:

$$max_{jk} = max_{w_l \in nouns(i,k)} \{\texttt{sim(}w_j\texttt{,}w_l\texttt{,}depth\texttt{)}\} \tag{1}$$

where $\texttt{sim(}w_j\texttt{,}w_l\texttt{,}depth\texttt{)}$ is the same similarity measure in $JIGSAW_{nouns}$. In other words, max_{jk} is the highest similarity value for w_j wrt the nouns related to the k-th sense for w_i. Finally, an overall similarity score among s_{ik} and the whole context C is computed:

$$\varphi(i, k) = R(k) \cdot \frac{\sum_{w_j \in C} G(pos(w_i), pos(w_j)) \cdot max_{jk}}{\sum_h G(pos(w_i), pos(w_h))} \tag{2}$$

where both $R(k)$ and $G(pos(w_i), pos(w_j))$, that gives a higher weight to words closer to the target word, are defined as in $JIGSAWnouns$. The synset assigned to w_i is the one with the highest φ value.

3.2 Named Entity Recognition Step

The Named Entity Recognition (NER) task has been defined in the context of the Message Understanding Conference (MUC) as the capability of identifing and categorizing entity names, defined as instances of the three types of expressions: entity names, temporal expressions, number expressions [5]. Further specializations of these top level classes have been proposed [11] and general purpose lists of Named Entities are publicly available and incorporated e.g. within well-known Text Processing Software, as GATE (General Architecture for Text Engineering) [4], to give a popular example. However, for the aim of JUMP project we cannot rely on general purpose gazetteers to perform the step of Named Entity Recognition, due to specificity of categories and their instances for this particular project. For this reason we developed a simple algorithm to recognize entities using as gazetteers a domain ontology: we tag each token in the original document with the ontology class value if it represents an instance of that class in the domain ontology. The idea of the algorithm follows. Given $C = \{C_1, C_2, \ldots, C_n\}$ the set of classes in the domain ontology, for each class C_k we consider the set $P = \{p_1, p_2, \ldots, p_m\}$ of properties belonging to C_k. Given $T = \{t_1, t_2, \ldots, t_s\}$ the list of tokens obtained from document d, for each token t_j we consider a window of h following tokens. The algorithm checks for each C_k if value of any combination of $t_j, \ldots, t_{j+h}$ matches with the value of any p_m, for all instances of C_k, and assigns to t_j the correspondent label. The search is done beginning from longer combinations of tokens and in the worst case it ends without any class annotation for the single token t_j. Taking advantage of semantic information provided by ontology, we can simply obtain relations between all entities found in the text, exploiting the object properties defined by the ontology, without added computational cost.

3.3 Concept-based Text Representation

Both the WSD procedure and the NER process are fundamental to obtain a concept-based text representation that we called Bag-Of-Concepts (BOC). In this model, a vector of concepts (WordNet synsets or named entities) corresponds to a document, instead of a vector of keywords. Therefore, each document (for example, an email text) representing the user request is converted in a BOC structure in order to be processed by the JUMP Engine. A more formal description of the BOC text representation follows. Assume that we have a document d_n (corresponding to a user's request n)processed by the *Content Analyzer Module*. The document is converted into the following BOC structure:

$$d_n = \langle (t_{n1}, w_{n1}), (t_{n2}, w_{n2}), \ldots, (t_{n|V|}, w_{n|V|}) \rangle$$

where:

- t_{nk} is the k-th token (synset or named entity) recognized in document d_n by NLP procedures;
- V is the set of distinct tokens recognized in d_n;

– w_{nk} is the weight representing the informative power of token t_{nk} in document d_n.

In other words, a text is represented by a vector of pairs *(token, weight)*, where tokens are recognized from keywords in the text by NLP procedures which assign each token with a numerical score representing the discriminatory power of that token in the text. Weights can be computed in different ways for synsets and named entities. For example, we consider a user who is going to prepare a technical report for a project. She has technical skills but no idea about how to compile such kind of document. She queries the Jump System, using the following expression q: *"I have to prepare a technical report for the Jump project"*. The Jump Engine passes the user's request q to the Content Analyzer module that processes the document in order to both disambiguate the task to be accomplished, by using the WSD procedure, and to recognize entities potentially useful for the task. The final output of this stage is q represented according to the BOC model:

$$q = \langle (01704982, 0.75), (06775158, 0.85), (Jump, 0.99), (00746508, 0.80) \rangle$$

where both synset identifiers of the concepts recognized in the text and named entities are used in the BOC structure. For example, the verb "prepare" has been disambiguated as the synset reported below:

`01704982 (verb.creation) prepare -- (to prepare verbally, either for written or spoken delivery; ‘‘prepare a report’’`

which identifies the task to be accomplished. This structure is used to query the Shared Ontology. As a result, the Jump Engine provides all istances of the concepts *technical report* and *project* and relations intercourring among these istances and istances of different classes of the ontology. In the example, the system returns the list of partners and documents related to the Jump project, and the list of technical reports already written for other projects.

4 Conclusions and Future Work

In this paper we presented the design and initial implementation of a framework for knowledge sharing within knowledge-intensive organizations by personalized information retrieval. The user current task and background knowledge are used to fulfill informative request. NLP and shared domain ontology are exploited to semantic interpretation of user request in order to query legacy knowledge bases. The JUMP project is an ongoing project; so far, a prototype implementing what presented in this paper has been developed as an internal proof of concept to verify that interfacing systems through the JUMP framework is feasible and useful even outside the project scope itself.Other features currently under development in the prototype are related to the different possible interfaces that the user can exploit in order to query the system. In particular, the possible interactions that have been depicted so far include support to Microsoft IBF (Information Bridge Framework) smart tags (in PUSH mode, i.e. without the user explicitly requesting services), and SMS and email support (in PULL mode, i.e. as answer to a

user explicit request). Since the user is likely to be a fairly experienced computer user and not a computer programmer, the query is expected to be a simple text query, not different from a normal query that could be issued against a standard query engine such as Google or Yahoo. As future work, we intend to improve the WSD algorithm and the NER process by including the shared ontology, enriched with external links to WordNet, into these processes.

Acknowledgements

This work has been co-funded by Regione Puglia, Italy, through the research funding program named POR Puglia 2000-2006 - Mis. 3.13, Sostegno agli Investimenti in Ricerca Industriale, Sviluppo Precompetitivo e Trasferimento Tecnologico.

References

1. F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. S. Banerjee and T. Pedersen. An adapted lesk algorithm for word sense disambiguation using wordnet. In *CICLing '02: Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing*, pages 136–145, London, UK, 2002. Springer-Verlag.
3. P. Basile, M. de Gemmis, A.L. Gentile, P. Lops, and G. Semeraro. Jigsaw algorithm for word sense disambiguation. In *SemEval-2007: 4th Int. Workshop on Semantic Evaluations*, pages 398–401. ACL press, 2007.
4. H. Cunningham, Y. Wilks, and R.J. Gaizauskas. Gate: a general architecture for text engineering. In *Proceedings of the 16th conference on Computational linguistics*, pages 1057–1060, Morristown, NJ, USA, 1996. Association for Computational Linguistics.
5. R. Grishman and B. Sundheim. Message understanding conference- 6: A brief history. In *COLING*, pages 466–471, 1996.
6. T. R. Gruber. A translation approach to portable ontology specifications. In *Knowledge Engineering, 5(2)*, pages 199–220. Academic Press, 1993.
7. C. Leacock and M. Chodorow. *Combining local context and WordNet similarity for word sense identification*, pages 305–332. MIT Press, 1998.
8. C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*, chapter 7: Word Sense Disambiguation, pages 229–264. MIT Press, Cambridge, US, 1999.
9. G. A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.
10. P. Resnik. Disambiguating noun groupings with respect to WordNet senses. In *Proc. of the 3th Workshop on Very Large Corpora*, pages 54–68. ACL, 1995.
11. S. Sekine, K. Sudo, and C. Nobata. Extended named entity hierarchy. In *Proceedings of the LREC-2002*, 2002.

The HMatch 2.0 Suite for Ontology Matchmaking *

S. Castano, A. Ferrara, D. Lorusso, and S. Montanelli

Università degli Studi di Milano
DICo - Via Comelico, 39, 20135 Milano - Italy
{castano,ferrara,lorusso,montanelli}@dico.unimi.it

Abstract. In this paper, we present the HMatch 2.0 suite for a flexible and tailored ontology matchmaking, by focusing on the architectural features and on the evaluation results. Applications of HMatch 2.0 are also discussed, with special regard for the ontology evolution issues in the frame of the BOEMIE research project.

1 Introduction

The increasing complexity of knowledge-intensive applications such as data integration, semantic search, semantic web services, peer-to-peer systems, social networks, demands for sophisticated and flexible ontology matchmaking systems, to appropriately manage and compare independent heterogeneous ontologies adopted by the various parties for knowledge representation and discovery. In particular, the capability of finding mappings between semantically related elements of two different ontologies according to different notions of similarity is a key feature for effective ontology matchmaking [1–3]. In this paper, we present the HMatch 2.0 suite for a flexible and tailored ontology matchmaking, by focusing on the architectural features and on the evaluation results and application issues also in the frame of the BOEMIE research project for the ontology evolution [1]. HMatch 2.0 has been designed to provide: i) a comprehensive suite of components for ontology matchmaking, that can be invoked alone or in combination to fit a wide range of matching requirements arising in different matching scenarios and applications; ii) a family of matching techniques for each different ontology matching component, to perform the matching process in the most suitable way, according to different understandings of the notion of semantic similarity; iii) an open architecture to ensure a high level of flexibility in combining the various matching components and to support a service-oriented interaction with knowledge intensive applications. With respect to our previous work on ontology matching [4, 5],the main contribution of this paper regards the new component-based architecture of HMatch 2.0 and the new functionalities of

* This paper has been partially funded by the BOEMIE Project, FP6-027538, 6th EU Framework Programme and by the ESTEEM PRIN project funded by the Italian Ministry of Education, University, and Research.

[1] http://www.boemie.org

instance matching and mapping composition that have been introduced in the framework of the BOEMIE project to support multimedia ontology evolution. The paper is organized as follows. In Section 2, we introduce the architecture of HMatch 2.0. Section 3 describes the components available for concept and instance matching. In Section 4, we discuss main issues on the evaluation of HMatch 2.0 while, in Section 5, we discuss expected application scenarios and current application to ontology evolution in BOEMIE. Finally, in Section 6, we provide our concluding remarks.

2 Architecture of HMatch 2.0

HMatch 2.0 is designed with the goal of providing a comprehensive framework for ontology matchmaking. In particular, HMatch 2.0 is composed by several matching components that can be used alone or in combination. Each component has the goal of evaluating a different type of matching under different understandings of similarity and by using different kind of matching techniques.

2.1 Component Interactions

HMatch 2.0 components and their interactions are shown in Figure 1.

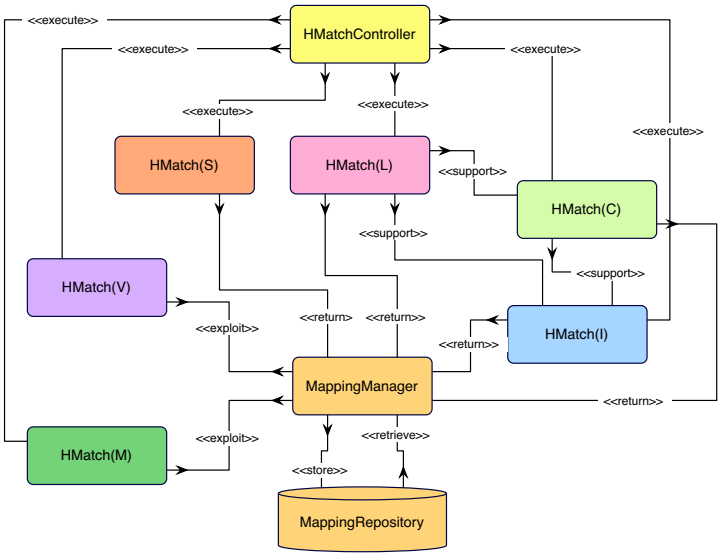


Fig. 1. High-level view of the HMatch 2.0 components and their interactions

The HMatchController is responsible of managing the configuration of HMatch 2.0 and the execution of the matching workflow. It also provides the interface

to all the matching components and to the mapping manager. HMatch($\mathcal{L}$) is the component devoted to enforce the linguistic matching. It is used both for providing to the designer the mappings derived from the linguistic analysis and as a support for the other modules which rely on linguistic affinity for implementing their matching task (i.e., HMatch($\mathcal{C}$) and HMatch($\mathcal{I}$)). HMatch($\mathcal{C}$) performs contextual matching and implements techniques for comparing ontology elements based on their contexts. HMatch($\mathcal{I}$) performs instance matching by interacting with HMatch($\mathcal{C}$) in order to establish the mappings at the schema level and with HMatch($\mathcal{L}$) to exploit the linguistic matching techniques for instance matching. HMatch($\mathcal{S}$) performs structural matching to evaluate the structural similarity between two ontologies. Since it is dependent only from the structure of the ontologies to be matched, it works on the graph structure of the ontologies without interactions with other components. HMatch($\mathcal{M}$) component provides all the functionalities required for merging sets of mappings by means of mapping operations such as intersection, union, product and transitive closure. HMatch($\mathcal{V}$) is used for mapping validation and inference. This component takes in input an initial set of mappings which can be calculated by means of any other HMatch 2.0 component. The MappingManager is responsible for the storage, release, and post-processing of mappings produced by the other modules. A mapping m produced as output of a matching process is a 5-tuple of the form:

$$m = \langle E_1, E_2, \mathcal{R}, \mathcal{V}, \mathcal{S} \rangle$$

where, E_1 and E_2 denote two ontology elements (i.e., concepts, properties, individuals), $\mathcal{R}$ denotes a semantic relation (e.g., $\equiv$, $\sqsubseteq$) holding between E_1 and E_2, $\mathcal{V} \in [0, 1]$ is a confidence value associated with $\mathcal{R}$, and $\mathcal{S} \in [0, 1]$ denotes the level of similarity between E_1 and E_2 determined by the matching component. The confidence value $\mathcal{V}$ denotes the level of trust associated with m and it is differently computed by each matching component (e.g., in HMatch($\mathcal{C}$), $\mathcal{V}$ is determined by considering the number of matching elements in the context of two ontology concepts).

2.2 Matching process

The workflow of the matching process of HMatch 2.0 is shown in Figure 2. When two ontologies are submitted to the matching process, the first step is the configuration of the HMatch 2.0 execution. In this step, the designer can choose the components of HMatch 2.0 to be activated during the matching process and can set the set of parameters required in the matching process. HMatch 2.0 is highly configurable and almost every parameter intervening in the matching process can be set by the designer. However, HMatch 2.0 is provided with default values for each parameter, in order to simplify the configuration step. If any of the activated components requires linguistic similarity analysis, the linguistic affinity (LA) is calculated between the names of ontology concepts and properties. Then, each selected matching component is executed. As a result, one or more sets of mappings are produced. The designer can stop the process and store the

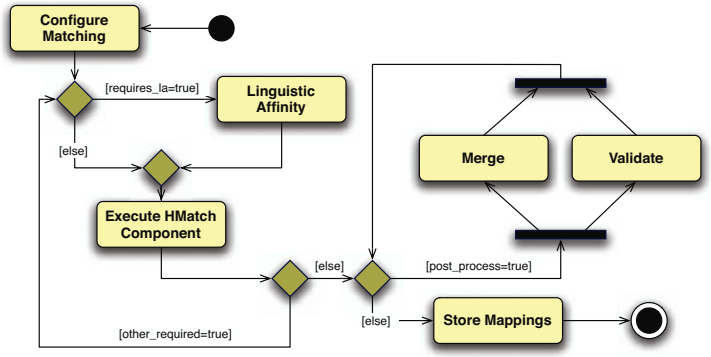


Fig. 2. Workflow of the matching process

mappings or he can perform a post-processing activity on the resulting mappings. During the mapping post-processing step, each set of mappings can be validated and new mappings can be inferred. Moreover, the different set of mappings obtained according to different similarity matchings can be merged into a comprehensive and coherent mapping set. The validation and merge step can be iterated as many times as required by the designer.

3 Ontology Matchmaking Functionalities

The HMatch 2.0 suite provides components for performing matching both at schema and instance level, namely HMatch($\mathcal{L}$), HMatch($\mathcal{C}$), HMatch($\mathcal{S}$), HMatch($\mathcal{I}$), and HMatch($\mathcal{M}$).

3.1 HMatch($\mathcal{L}$): Linguistic Matching

The Linguistic Matching component determines the level of semantic correspondence between terms used as names of ontology elements (i.e., concepts, properties, individuals). The degree of such a correspondence is calculated through a Linguistic Affinity function (LA) which returns a value in the range [0,1]. LA is used as the fundamental technique for establishing the similarity between two atomic ontology elements to be matched in all the situations when linguistic analysis is required. In HMatch 2.0, LA can be evaluated by means of three different approaches:

- **Syntactic**: using a string matching algorithm (i.e., QGram, i_Sub).
- **Semantic**: using a thesaurus or a lexical system (i.e., WordNet) of terms and terminological relationships and a notion of weighted distance between terms.
- **Combined**: using a function that combines syntactic and semantic measures.

3.2 HMatch($\mathcal{C}$): Contextual Matching

HMatch($\mathcal{C}$) determines the level of semantic affinity between concepts of two independent ontologies. A threshold-based mechanism is enforced to set the minimum level of semantic affinity required to consider two concepts as matching concepts. Given two concepts c and c', HMatch($\mathcal{C}$) calculates a semantic affinity value $SA(c, c')$ in the range $[0, 1]$ as the linear combination of a linguistic affinity value $LA(c, c')$ produced by exploiting HMatch($\mathcal{L}$) and a contextual affinity value $CA(c, c')$. The contextual affinity function of HMatch 2.0 provides a measure of similarity by taking into account the contextual features of the ontology concepts c and c'. The context can be differently composed to consider different levels of semantic complexity, and four matching models, namely, *surface*, *shallow*, *deep*, and *intensive*, are defined to this end. The context of a concept can include properties, semantic relations with other concepts, and property values. In the surface matching, only the linguistic affinity between the concept names of c and c' is considered to determine concept similarity. A detailed description of there matching models and of their evaluation is given in [4].

3.3 HMatch($\mathcal{S}$): Structural Matching

The goal of HMatch($\mathcal{S}$) is to evaluate the degree of similarity between two ontologies on the basis of their structure, by considering the RDF graph associated with the OWL representation of the two ontologies to be compared. More specifically, HMatch($\mathcal{S}$) is conceived to work only with structural information and without taking into account linguistic and contextual features of the two ontologies. This goal is important when the two ontologies do not provide significant linguistic information or the names used for ontology elements are missing or misleading. Another advantage of HMatch($\mathcal{S}$) is the fact that it allows to define mappings also for the elements which are not featured by a name, such as the so-called *anonymous classes* (i.e., quantified and cardinality restrictions, collections).

Goal of HMatch($\mathcal{S}$), in other terms, is to capture the *structural role* of an ontology element, that is the combination of information about the position of the element with respect to other elements in the ontology, the number and features of RDF triples in which it is involved, and the position within a RDF triple (i.e., subject, predicate, or object). In particular, HMatch($\mathcal{S}$) takes into account the following features for each element:

- The type of a feature, i.e., classes, properties, instances, anonymous classes.
- The number of RDF triples involving an element.
- The position within a triple (i.e., subject, predicate, or object).
- The type of the elements involved into the same RDF triples of a given element.
- The language constructs involved in the same RDF triples of a given element.

3.4 HMatch($\mathcal{I}$): Instance Matching

The goal of the instance matching component is to determine instances that represent the same real object or event and to help in disambiguating multiple

explanations of the same entity [6]. HMatch($\mathcal{I}$) evaluates the degree of similarity among different individuals by considering those assertions which provide a description of the individuals features. Consequently, the similarity of role filler values as well as the similarity of their direct types is evaluated. When two instances are compared, their similarity is proportional to the number of similar roles and role fillers they share. Moreover, for the similarity evaluation we associate a different weight with different properties of the instances, to capture the fact that some properties are more important than others in denoting the real object denoted by an instance, because they are relevant for object identification. For example, the name of a person is more important than his age for the goal of identifying the person.

The approach adopted in HMatch 2.0 is based on the idea of considering roles (referred also as properties) as connections between individuals and propagating similarity values through them. Each specification of an individual of the ABoxes is represented by means of a tree. In order to evaluate the degree of similarity of two individuals, the procedure computes a measure of similarity between datatype values and propagates these similarity degrees to the individuals of the higher level by combining the similarity among their role fillers. To this end, HMatch($\mathcal{I}$) provides a set of specific techniques devoted to the evaluation of similarity between datatype values. A function called *datatype role filler matching* is responsible of selecting the most suitable matching technique for each pair of datatype role fillers, according to the semantic meaning of the roles and to the datatype category.

3.5 HMatch($\mathcal{M}$): Mapping Analysis and Combination

Using HMatch 2.0 it is possible to collect several sets of mappings produced by the different components available both for concept and for instance matching. The different sets of mappings can be produced against the same ontologies or against a collection of different ontologies. Even in case of mappings collected against the same ontologies, the correspondences among ontology elements can be different moving from one set of mapping to another, because the different components analyze different features of the ontology elements. In order to obtain a comprehensive evaluation of the level of matching between two ontologies, HMatch 2.0 provides operations for combining different sets of mappings into a unique set and for dealing with the cardinality of a mapping set. Operations supported by HMatch($\mathcal{M}$) are defined over two mapping sets and are *intersection*, *union*, *product*, and *transitive closure*. Intersection and union are used to combine together the results obtained by different matching components against the same ontologies. From a conceptual point of view, the intersection has the goal of selecting those ontology elements that are similar with respect to more than one matching component at the same time, while union has the goal of selecting elements which are similar with respect to at least one matching component. Combining structural similarity with linguistic similarity, for example, is useful in case of elements labeled with different terms or terms that are not retrieved to be similar by linguistic matching techniques. On the other side, product and

transitive closure are used for combining together results obtained from different ontologies. The semantics of mapping operation is described in [7].

4 Evaluation results

In general, the evaluation of ontology matchmaking tools is based on the idea of using a benchmark constituted by several heterogeneous ontologies to be matched and a set of manually defined results, that is a set of expected mappings. Then, the matching tool to be evaluated is executed against the ontologies in the benchmark, in order to obtain a set of automatically retrieved mappings. On the basis of these two sets of mappings, conventional metrics are employed for the evaluation of the tool, namely precision, recall, and F-measure [8].

The evaluation of HMatch 2.0 has been performed over the 2006 and 2007 benchmarks of the Ontology Alignment Evaluation Initiative (OAEI), an international ontology matching contest held with the goal of comparing different ontology matching tools [5] [2]. We performed the evaluation of each component separately as well as the evaluation of HMatch($\mathcal{M}$)by applying the union and intersection operations. A summary view of these result is reported in Table 1. A detailed description of these results is given in [7, 5].

	HMatch($\mathcal{C}$)	HMatch($\mathcal{S}$)	HMatch($\mathcal{C}$) $\cap$ HMatch($\mathcal{S}$)	HMatch($\mathcal{C}$) $\cup$ HMatch($\mathcal{S}$)
Precision	0.92	0.81	0.99	0.81
Recall	0.60	0.72	0.35	0.76
F-Measure	0.73	0.77	0.51	0.79

Table 1. Evaluation results of HMatch 2.0

As a general remark, we note that the intersection is useful to increase precision (up to a very high level of 0.99), while union is useful to increase recall. This is because the general behavior of HMatch 2.0 is to find a quite low number of results, but very correct. Then, if we apply intersection, we reduce again the number of results, but we increase the probability to have them correct. On the opposite, if we take the union, we increase the number of results by affecting precision. More in detail, the results show that the union provides the best balance between precision and recall. The general conclusion is that intersection is supposed to be used when the precision of the results is much more important than the number of results retrieved. In all the other cases, union is the best solution in order to combine different components of HMatch 2.0.

5 Application scenarios

The HMatch 2.0 ontology matching suite provides a highly configurable matching environment where the various components can be dynamically selected accord-

[2] http://oaei.ontologymatching.org/2007/

ing to the application context that is considered for a given matching case. With respect to the complexity of the ontologies to be matched, we note that HMatch 2.0 is intended to work with OWL ontologies and it is compatible with all the OWL dialects (i.e., OWL Lite, OWL DL, OWL Full). By taking into account the elements that affects the matching process, the level of semantic complexity supported by HMatch 2.0 is $\mathcal{ALC}$. In Figure 3, we summarize the applicability of each HMatch 2.0 component in terms of i) the ontological features that are considered, and ii) the suggested application context.

	HMatch($\mathcal{L}$)	HMatch($\mathcal{C}$)	HMatch($\mathcal{S}$)	HMatch($\mathcal{I}$)
Ontological description	Poorly structured ontologies, with very simple concept descriptions (e.g., Web directories, glossaries)	Ontologies with semantically rich concept descriptions (e.g., OWL ontologies, DL specifications)	Ontologies with "nameless" concept descriptions (e.g., XML documents, Ontologies with generally labeled elements)	Ontologies with instances (e.g., Semantic Web ontologies)
Application context	Oriented to soft chema integration	Oriented to stable schema integration	Oriented to semi-structured document integration	Oriented to data/information integration

Fig. 3. Applicability of the HMatch 2.0 components

HMatch($\mathcal{L}$) is suited to work with poorly structured ontologies, such as Web directories, taxonomies, and glossaries, where properties, semantic relations, and instances are not available. Moreover, HMatch($\mathcal{L}$) is also used with richer ontology descriptions to perform an initial comparison and to provide a linguistic analysis as input for the execution of other HMatch 2.0 components. In general, HMatch($\mathcal{L}$) is suggested for generic matching scenarios where the linguistic component is relevant and/or is self-explanatory. In particular, soft schema integration is a typical application context where HMatch($\mathcal{L}$) is invoked to identify the corresponding labels used in different schemas of web sources. As a further application context, HMatch($\mathcal{L}$) can be used for supporting social tagging and classification of Web resources (i.e., *folksonomy*). In this respect, HMatch($\mathcal{L}$) has the role to suggest to the user the "right tag" for a given resource according to linguistic-based rather than popularity-based metrics. When semantically rich ontology descriptions are provided (e.g., OWL ontologies, DL specifications), HMatch($\mathcal{C}$) is the suggested component to use due to the high level of matching granularity enforced through its matching models (i.e., surface, shallow, deep, intensive). For this reason, HMatch($\mathcal{C}$) is suited for general-purpose matching scenarios where it can be properly combined with HMatch($\mathcal{L}$) to obtain stable integrated representation of the information. For instance, in schema integration applications, HMatch($\mathcal{C}$) can be used to refine the results of the linguistic matching and to provide a more accurate matching evaluation by taking into account contextual features of schema elements to be integrated. HMatch($\mathcal{S}$) is suggested when the ontology descriptions contain "nameless" concept descrip-

tions, that is ontologies with anonymous classes or with non-meaningful element names. For this reason, HMatch($\mathcal{S}$) is suited for those application contexts where the meaningfulness of the terminological part is not guaranteed and/or is not a core requirement. As an example, HMatch($\mathcal{S}$) can be adopted for semi-structured document integration (e.g., XML documents) where tag labels are often automatically generated by applications thus making ineffective the adoption of linguistic-based matching components. For what concern HMatch($\mathcal{I}$), it is suited to work with ontology instances (ABoxes). In particular, HMatch($\mathcal{I}$) is suggested for those application contexts where extensional matching is applicable, such as data/information integration.

We note that in real matching scenarios more than one HMatch 2.0 component can be executed according to the specific features of the ontology descriptions to be matched. In this respect, the results produced by each component can be combined by invoking HMatch($\mathcal{M}$) in order to return a single comprehensive set of matching results. Moreover, the combination of different HMatch 2.0 components can contribute to increase the quality of the matching results. For instance, intersection and union of HMatch($\mathcal{C}$) and HMatch($\mathcal{S}$) results provide better performance in terms of precision and recall, respectively, as discussed in Section 4.

5.1 A practical application to ontology evolution

The HMatch 2.0 ontology matching suite is actually adopted in the framework of the BOEMIE project where it is exploited for supporting multimedia ontology evolution. In BOEMIE, a novel methodology for ontology evolution is defined to enhance traditional approaches and to provide methods and techniques for evolving a domain ontology through acquisition of semantic information from multimedia sources such as image, video, and audio [9]. The BOEMIE methodology is characterized by the use of a reasoning-based engine with the role of providing a semantic interpretation of the extracted information and by the use of an ontology matching engine. In particular, HMatch 2.0 is used as a comprehensive matching service to support the BOEMIE evolution activities and tasks as shown in Table 2. According to the interpretation results, ontology evolution is performed i) through *population* by inserting new instances in the ontology, and ii) through *enrichment* by inserting new concepts and relation types in the ontology. Coordination activities are also defined in the BOEMIE methodology to log changes and to manage the different versions of the ontology produced with evolution.

6 Concluding Remarks

In this document, we have described the architecture and the main matching components implemented in the HMatch 2.0 ontology matchmaking suite. Instance matching is a challenging task and HMatch 2.0 is one of the few ontology

Activity	Task	HMatch 2.0 component	Goal
Population	Instance grouping	HMatch($\mathcal{I}$)	To group together instances referred to the same individual in the domain
Enrichment	Concept enhancement	HMatch($\mathcal{L}$), HMatch($\mathcal{C}$)	To suggest names for new concepts and properties
Coordination	Alignment	HMatch($\mathcal{C}$), HMatch($\mathcal{S}$)	To align a new version of the domain ontology with other external knowledge sources

Table 2. Usage of HMatch 2.0 components in BOEMIE

matching tools with instance matching functionalities. Moreover, we have introduced the idea of composing mappings obtained by applying different matching components. In such a way, the domain expert is capable of collecting separate mappings sets on the ground of different application purposes and of deriving a comprehensive similarity view of ontology elements. Our future work will be devoted (i) to increasing the recall results obtained with HMatch($\mathcal{C}$) starting from experimental data and working on the combination between HMatch($\mathcal{C}$) and HMatch($\mathcal{S}$), and (ii) to formalize the semantics of mapping operations by taking into account the different types of mappings and their relations.

References

1. Aumueller, D., Do, H., Massmann, S., Rahm, E.: Schema and Ontology Matching with COMA++. In: Proc. of SIGMOD 2005 - Software Demonstration, Baltimore, USA (2005)
2. Jian, N., Hu, W., Cheng, G., Qu, Y.: Falcon-AO: Aligning Ontologies with Falcon. In: Proc. of K-CAP Workshop on Integrating Ontologies, Banff, Canada (2005)
3. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer-Verlag (2007)
4. Castano, S., Ferrara, A., Montanelli, S.: Matching Ontologies in Open Networked Systems: Techniques and Applications. Journal on Data Semantics (JoDS) **V** (2006)
5. Castano, S., Ferrara, A., Messa, G.: ISLab HMatch Results for OAEI 2006. In: Proc. of ISWC Int. Workshop on Ontology Matching, Athens, Georgia, USA (2006)
6. Gu, L., Baxter, R.A., Vickers, D., Rainsford, C.: Record Linkage: Current Practice and Future Directions. Technical Report 03/83, CSIRO Mathematical and Information Sciences (2003)
7. Bruno, S., Castano, S., Ferrara, A., Lorusso, D., Messa, G., Montanelli, S.: Ontology Coordination Tools: Version 2. Technical Report D4.7, BOEMIE Project, FP6-027538, 6th EU Framework Programme (2007)
8. Salton, G.: Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley (1989)
9. Castano, S., Espinosa, S., Ferrara, A., Karkaletsis, V., Kaya, A., Melzer, S., Möller, R., Montanelli, S., Petasis, G.: Ontology Dynamics with Multimedia Information: The BOEMIE Evolution Methodology. In: Proc. of the ESWC Int. Workshop on Ontology Dynamics (IWOD 07), Innsbruck, Austria (2007)

Semantic Nearest Neighbor Search in OWL Ontologies

Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito

Dipartimento di Informatica, Università degli studi di Bari
Campus Universitario, Via Orabona 4, 70125 Bari, Italy
{fanizzi|claudia.damato|esposito}@di.uniba.it

Abstract. A *nearest neighbor* search procedure is presented, for retrieving resources in knowledge bases expressed in OWL. The procedure exploits a semi-distance for annotated resources, that is based on a number of dimensions corresponding to a committee of features represented by OWL concept descriptions. The procedure can retrieve resources belonging to query concepts expressed in OWL, by analogy with other training instances, on the grounds of the classification of the nearest ones w.r.t. the dissimilarity measure. Besides, it may also be able to suggest new assertions that are not logically entailed by the knowledge base due to open world semantics. In the experimentation, where we compare the performance of the procedure to running a reasoner, we show that it can be quite accurate and augment the scope of its applicability, improving w.r.t. previous prototypes that adopted other semantic measures.

1 Introduction

In the perspective of resource retrieval, purely logical approaches pursued so far, in the context of the Semantic Web, may fall short in terms of noise-tolerance and efficiency. Hence, *analogical* methods applied to multi-relational domains appear particularly well suited, since they are known to be more efficient and noise-tolerant, which is very important in contexts where knowledge is intended to be acquired from distributed sources. To this purpose, a relational distance-based framework for retrieving resources contained in semantic knowledge bases has been devised to infer inductively consistent class-membership assertions that may be not logically derivable due to the open-world semantics. The main idea is that similar individuals, by analogy, should likely belong to the extension of similar concepts.

Specifically, we present a retrieval procedure that constitutes a multi-relational extension [5] of the well-known *Nearest Neighbor* approach (henceforth, *NN*) [10]. These algorithms may be quite efficient because they require checking query-membership for a limited set of training instances on such concepts and making a decision on the classification of new instances.

From a technical viewpoint, extending the NN setting to work on multi-relational representations, such as concept languages like OWL, required suitable metrics whose definition could not be straightforward. In particular, a theoretical problem has been posed by the *Open World Assumption* (OWA) that is generally made in the target context, differently from typical databases settings where the *Closed World Assumption* (CWA) is the standard. Indeed the NN algorithms are devised for simple classifications

where classes are assumed to be pairwise disjoint which is quite unlikely in the Semantic Web context.

As pointed out in [3], most of the existing measures focus on the similarity of atomic concepts within hierarchies or simple ontologies. Moreover they have been conceived for assessing *concept* similarity. Conversely, for our purposes, a notion of similarity between *individuals* is required. Recently, dissimilarity measures for specific description logics concept descriptions have been proposed [3, 4]. Although they turned out to be quite effective for the inductive tasks of interest, they are still partly based on structural criteria (a notion of normal form) which determine their main weakness: they are hardly scalable to deal with standard languages, such as OWL-DL, commonly used for ontologies and knowledge bases.

A new semantic pseudo-metric [7] is exploited in order to overcome these limitations. Following the distance-induction method proposed in [9], the proposed measures are based on a committee of features (concepts) onto which individuals are projected for being compared. As such, these measures are not absolute, yet they depend on the knowledge base they are applied to. However, the measures are suitable for a wide range of languages, since they merely depend on the discernibility of the input individuals w.r.t. a fixed set of concepts. The choice of optimal committees may be performed in advance through randomized search algorithms [7].

Such measures have been integrated in the NN procedure presented in [4]. Essentially the classification of a resource w.r.t. a query concept is performed by selecting the closest resources in the knowledge base and then determining the membership through a weighted voting procedure based on the neighbor similarity.

The resulting system allowed for an experimentation of the method on performing instance retrieval with real ontologies drawn from public repositories comparing its predictions to the assertions that were logically derived by a standard reasoner. These experiments show that the novel measure considerably increases the effectiveness of the method with respect to past experiments where the same procedure was integrated with other dissimilarity measures [4].

The paper is organized as follows. The basics of the instance-based approach applied to the standard representations are recalled in Sect. 2. The next Sect. 3 presents the semantic similarity measures adopted in the retrieval procedure. Sect. 4 reports the outcomes of experiments performed with the implementation of the procedure. Possible developments are finally examined in Sect. 5.

2 Resource Retrieval as Nearest Neighbor Search

2.1 Representation and Inference

In the following sections, we assume that concept descriptions are defined in terms of a generic sublanguage based on OWL-DL that may be mapped to *Description Logics* with the standard model-theoretic semantics (see the handbook [1] for a thorough reference).

A *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ contains a *TBox* $\mathcal{T}$ and an *ABox* $\mathcal{A}$. $\mathcal{T}$ is a set of axioms that define concepts. $\mathcal{A}$ contains factual assertions concerning the resources, also known as individuals. Moreover, the *unique names assumption* may be made on

the ABox individuals, that are represented by their URIs. The set of the individuals occurring in $\mathcal{A}$ will be denoted with $\mathsf{Ind}(\mathcal{A})$.

As regards the inference services, like all other instance-based methods, our procedure may require performing *instance-checking* [1], which roughly amounts to determining whether an individual, say a, belongs to a concept extension, i.e. whether $C(a)$ holds for a certain concept C. Note that because of the OWA, a reasoner may be unable to give a positive or negative answer to a class-membership query. This service is provided proof-theoretically by a reasoner.

2.2 The Method

Query answering boils down to determining whether a resource belongs to a (query) concept extension. Here, an alternative inductive method is proposed for retrieving the resources that likely belong to a query concept. Such a method may also be able to provide an answer even when it may not be inferred by deduction, Moreover, it may also provide a measure of the likelihood of its answer.

In *similarity search* [10] the basic idea is to find the most similar object(s) to a query one (i.e. the one that is to be classified) with respect to a similarity (or dissimilarity) measure. We review the basics of the k-NN method applied to the Semantic Web context [4] context.

The objective is to induce an approximation for a discrete-valued target hypothesis function $h : \mathit{IS} \mapsto V$ from a space of instances *IS* to a set of values $V = \{v_1, \ldots, v_s\}$ standing for the classes (concepts) that have to be predicted. Note that normally $|\mathit{IS}| \ll |\mathsf{Ind}(\mathcal{A})|$ i.e. only a limited number of training instances is needed especially if they are prototypical for a region of the search space. Let x_q be the query instance whose class-membership is to be determined. Using a dissimilarity measure, the set of the k nearest (pre-classified) training instances w.r.t. x_q is selected: $NN(x_q) = \{x_i \mid i = 1, \ldots, k\}$.

In its simplest setting, the k-NN algorithm approximates h for classifying x_q on the grounds of the value that h is known to assume for the training instances in $NN(x_q)$, i.e. the k closest instances to x_q in terms of a dissimilarity measure. Precisely, the value is decided by means of a weighted majority voting procedure: it is simply the most *voted* value by the instances in $NN(x_q)$ weighted by the similarity of the neighbor individual.

The estimate of the hypothesis function for the query individual is:

$$\hat{h}(x_q) := \operatorname*{argmax}_{v \in V} \sum_{i=1}^{k} w_i \delta(v, h(x_i)) \tag{1}$$

where δ returns 1 in case of matching arguments and 0 otherwise, and, given a dissimilarity measure d, the weights are determined by $w_i = 1/d(x_i, x_q)$.

Note that the estimate function $\hat{h}$ is defined extensionally: the basic k-NN method does not return an intensional classification model (a function or a concept definition), it merely gives an answer for the instances to be classified. It should be also observed that this setting assigns a value to the query instance which stands for one in a set of pairwise disjoint concepts (corresponding to the value set V). In a multi-relational

setting this assumption cannot be made in general. An individual may be an instance of more than one concept.

The problem is also related to the CWA usually made in the knowledge discovery context. To deal with the OWA, the absence of information on whether a training instance x belongs to the extension of the query concept Q should not be interpreted negatively, as in the standard settings which adopt the CWA. Rather, it should count as neutral (uncertain) information. Thus, assuming the alternate viewpoint, the multi-class problem is transformed into a ternary one. Hence another value set has to be adopted, namely $V = \{+1, -1, 0\}$, where the three values denote, respectively, membership, non-membership, and uncertainty, respectively.

The task can be cast as follows: given a query concept Q, determine the membership of an instance x_q through the NN procedure (see Eq. 1) where $V = \{-1, 0, +1\}$ and the hypothesis function values for the training instances are determined as follows:

$$h_Q(x) = \begin{cases} +1 & \mathcal{K} \models Q(x) \\ -1 & \mathcal{K} \models \neg Q(x) \\ 0 & \textit{otherwise} \end{cases}$$

i.e. the value of h_Q for the training instances is determined by the entailment[1] the corresponding assertion from the knowledge base.

Note that, being based on a majority vote of the individuals in the neighborhood, this procedure is less error-prone in case of noise in the data (e.g. incorrect assertions) w.r.t. a purely logic deductive procedure, therefore it may be able to give a correct classification even in case of (partially) inconsistent knowledge bases.

It should be noted that the inductive inference made by the procedure shown above is not guaranteed to be deductively valid. Indeed, inductive inference naturally yields a certain degree of uncertainty. In order to measure the likelihood of the decision made by the procedure (individual x_q belongs to the query concept denoted by value v maximizing the argmax argument in Eq. 1), given the nearest training individuals in $NN(x_q, k) = \{x_1, \ldots, x_k\}$, the quantity that determined the decision should be normalized by dividing it by the sum of such arguments over the (three) possible values:

$$l(\textit{class}(x_q) = v | NN(x_q, k)) = \frac{\sum_{i=1}^{k} w_i \cdot \delta(v, h_Q(x_i))}{\sum_{v' \in V} \sum_{i=1}^{k} w_i \cdot \delta(v', h_Q(x_i))} \tag{2}$$

Hence the likelihood of the assertion $Q(x_q)$ corresponds to the case when $v = +1$.

3 A Semantic Pseudo-Metric for Individuals

As mentioned in the first section, various attempts to define semantic similarity (or dissimilarity) measures for concept languages have been made, yet they have still a limited applicability to simple languages [3] or they are not completely semantic depending also on the structure of the descriptions [4]. Moreover, for our purposes, we need a function for measuring the similarity of individuals rather than concepts. It can be observed that

[1] We use $\models$ to denote entailment, as computed through a reasoner.

individuals do not have a syntactic structure that can be compared. This has led to lifting them to the concept description level before comparing them (recurring to the notion of the *most specific concept* of an individual w.r.t. the ABox [1], yet this makes the measure language-dependent. Besides, it would add a further approximations as the most specific concepts can be defined only for simple DLs.

For the NN procedure, we intend to exploit a new measure that totally depends on semantic aspects of the individuals in the knowledge base.

3.1 The Family of Measures

The new dissimilarity measures are based on the idea of comparing the semantics of the input individuals along a number of dimensions represented by a committee of concept descriptions. Indeed, on a semantic level, similar individuals should behave similarly with respect to the same concepts. Following the ideas borrowed from [9], totally semantic distance measures for individuals can be defined in the context of a knowledge base.

More formally, the rationale is to compare individuals on the grounds of their semantics w.r.t. a collection of concept descriptions, say $\mathsf{F} = \{F_1, F_2, \ldots, F_m\}$, which stands as a group of discriminating *features* expressed in the OWL-DL sublanguage taken into account.

In its simple formulation, a family of distance functions for individuals inspired to Minkowski's norms L_p can be defined as follows [7]:

Definition 3.1 (family of measures). *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base. Given a set of concept descriptions $\mathsf{F} = \{F_1, F_2, \ldots, F_m\}$, a family of dissimilarity functions $d_p^{\mathsf{F}} : \mathsf{Ind}(\mathcal{A}) \times \mathsf{Ind}(\mathcal{A}) \mapsto [0, 1]$ is defined as follows:*

$$\forall a, b \in \mathsf{Ind}(\mathcal{A}) \quad d_p^{\mathsf{F}}(a, b) := \frac{1}{|\mathsf{F}|} \left[\sum_{i=1}^{|\mathsf{F}|} \mid \pi_i(a) - \pi_i(b) \mid^p \right]^{1/p}$$

where $p > 0$ and $\forall i \in \{1, \ldots, m\}$ the projection function *π_i is defined by:*

$$\forall a \in \mathsf{Ind}(\mathcal{A}) \quad \pi_i(a) = \begin{cases} 1 & F_i(a) \in \mathcal{A} \qquad (\mathcal{K} \models F_i(a)) \\ 0 & \neg F_i(a) \in \mathcal{A} \qquad (\mathcal{K} \models \neg F_i(a)) \\ 1/2 & otherwise \end{cases}$$

The superscript F will be omitted when the set of features is fixed.

The alternative definition for the projections, requires the entailment of an assertion (instance-checking) rather than the simple ABox look-up; this can make the measure more accurate yet more complex to compute unless a KBMS is employed maintaining such information at least for the concepts in F.

3.2 Discussion

It is easy to prove [7] that these functions have the standard properties for pseudo metrics (i.e. semi-distances [10]):

Proposition 3.1 (pseudo-metric). *For a given a feature set* F *and* $p > 0$, d_p *is a pseudo-metric.*

It cannot be proved that $d_p^{\mathsf{F}}(a, b) = 0$ iff $a = b$. This is the case of *indiscernible* individuals with respect to the given set of features F. To fulfill this property several methods have been proposed involving the consideration of equivalent classes of individuals or the adoption of a supplementary meta-feature F_0 determining the equality of the two individuals.

Compared to other proposed dissimilarity measures [3, 4], the presented functions do not depend on the constructors of a specific language, rather they require only (retrieval or) instance-checking for computing the projections through class-membership queries to the knowledge base.

The complexity of measuring he dissimilarity of two individuals depends on the complexity of such inferences (see [1], Ch. 3). Note also that the projections that determine the measure can be computed (or derived from statistics maintained on the knowledge base) before the actual distance application, thus determining a speed-up in the computation of the measure. This is very important for algorithms that massively use this distance, such as all instance-based methods.

The measures strongly depend on F. Here, we make the assumption that the feature-set F represents a sufficient number of (possibly redundant) features that are able to discriminate really different individuals. The choice of the concepts to be included – *feature selection* – is beyond the scope of this work (see [7] for a randomized optimization procedure aimed at finding optimal committees). Experimentally, we could obtain good results by using the very set of both primitive and defined concepts found in the knowledge base.

Of course these approximate measures become more and more precise as the knowledge base is populated with an increasing number of individuals.

4 Experimentation

4.1 Experimental Setting

In order to test the NN procedure integrated with the pseudo-metric proposed in the previous section, we have applied it to retrieval problems on random queries.

To this purpose, a number of OWL ontologies was selected, namely: FINITE STATE MACHINES (FSM), SURFACE-WATER-MODEL (SWM), part of SCIENCE and NEW TESTAMENT NAMES (NTN) from the Protégé library[2], the Semantic Web Service Discovery dataset[3] (SWSD) and the FINANCIAL ontology[4]. Tab. 1 summarizes the details of these knowledge bases.

For each ontology, 30 queries were randomly generated by composition of primitive or defined concepts. The performance was evaluated comparing the decisions made by the NN procedure to those returned by a standard reasoner[5] as a baseline.

[2] `http://protege.stanford.edu/plugins/owl/owl-library`

[3] `https://www.uni-koblenz.de/FB4/Institutes/IFI/AGStaab/Projects/xmedia/dl-tree.htm`

[4] `http://www.cs.put.poznan.pl/alawrynowicz/financial.owl`

[5] We employed PELLET v. 1.5. See `http://pellet.owldl.com`

Table 1. Data concerning the ontologies employed in the experiments.

knowledge base	DL language	#concepts	#object prop.	#data prop.	#individuals
FSM	$\mathcal{SOF}(D)$	20	10	7	37
SWM	$\mathcal{ALCOF}(D)$	19	9	1	115
SCIENCE	$\mathcal{ALCIF}(D)$	74	70	40	331
NTN	$\mathcal{SHIF}(D)$	47	27	8	676
SWSD	$\mathcal{ALCH}$	258	25	0	732
FINANCIAL	$\mathcal{ALCIF}$	60	17	0	1000

The parameter k was set to $\log |\mathsf{Ind}(\mathcal{A})|$ depending on the number of individuals in the ontology. Yet we found experimentally that much smaller values could be chosen, resulting in the same classification. We employed the simpler version of the distance (d_1^{F}) using all the concepts in the knowledge base for determining the set F.

4.2 Results

Standard IR measures. Initially the standard IR measures precision, recall, and F_1-measure were employed to evaluate the system performance. The outcomes are reported in Fig.2. For each knowledge base, we report the average values obtained over the 30 queries as well as their standard deviation and minimum-maximum ranges of values.

It is possible to note that precision and recall are generally quite good except in the experiment with the SWSD ontology where precision was significantly lower. Namely, SWSD turned out to be more difficult (also in terms of recall) for two reasons: a very limited number of individuals per concept was available and the number of concepts is larger than in other knowledge bases. For the other ontologies scores are quite high, as testified also by the F-measure values. The results in terms of recall are also more stable than those for recall as proved by the limited variance observed, whereas some queries turned out to be quite difficult w.r.t. the correctness of the answer.

The reasons for precision being less than recall are probably related to the OWA. Indeed, in a many cases it was observed that the NN procedure deemed some individuals as relevant for the query issued while the DL reasoner was not able to assess this relevance and this was computed as a mistake while it may likely turn out to be a correct inference when judged by a human agent.

Because of the problem issued by the OWA, in some cases it could not be (deductively) ascertained whether a resource was relevant or not for a given query. Thus explicitating both the rate of inductively classified individuals and the real nature of the mistakes would be needed. This leads to consider different indices.

Alternative measures. In previous works [4], we had employed the following indices for the evaluation:

- *match rate*: rate of individuals whose classification matched the reasoner decision;
- *omission error rate*: rate of individuals for which inductive method could not determine whether they were relevant to the query (or not) while they were actually relevant according to the reasoner;

Table 2. Experimental results in terms of standard IR measures: average ± standard deviation and [min.;max.] intervals.

	precision	recall	F-measure
FSM	89.22 ± 15.88 [28.60;100.00]	91.63 ± 12.41 [50.00;100.00]	90.26 ± 14.46 [36.39;100.00]
SWM	73.35 ± 11.66 [52.90;93.80]	89.56 ± 9.35 [73.30;97.50]	80.52 ± 10.55 [62.04;93.80]
SCIENCE	94.55 ± 6.03 [86.70;99.70]	97.12 ± 2.78 [93.50;99.70]	95.79 ± 4.45 [89.97;99.70]
NTN	78.73 ± 9.98 [34.60;95.60]	92.28 ± 4.58 [85.30;99.70]	84.63 ± 7.84 [49.23;97.61]
SWSD	55.30 ± 11.01 [31.90;74.10]	70.59 ± 10.37 [56.80;86.20]	61.51 ± 9.68 [41.03;79.69]
FINANCIAL	89.57 ± 19.48 [22.40;99.70]	97.80 ± 5.06 [84.70;100.00]	92.43 ± 15.47 [35.75;99.85]

- *commission error rate*: rate of individuals inductively found to be relevant to the query concept, while the reasoner assigned them to its negation (and vice-versa);
- *induction rate*: rate of individuals whose relevance (or irrelevance) relevant w.r.t. the query concept could be determined by the inductive method, while this classification could not be derived logically by the reasoner.

Tab. 3 reports the outcomes in terms of these new indices. Preliminarily, it is important to note that, in each experiment, the commission error was low or absent. This means that the search procedure is quite accurate: it did not make critical mistakes i.e. cases when an individual is deemed as an instance of a concept while it really is an instance of a disjoint one. Furthermore, the rate of omission errors was quite low, yet it is more frequent for the considered ontologies especially when few disjointness axioms were specified. A noteworthy difference was observed for the case of the FINANCIAL ontology for which we found the lowest match rate and the highest variability in the observed results over the various query concepts.

Comparing these outcomes to those reported in other works on the same task [4], where the highest average match rate observed was about 80%, we find a significant increase of the performance due to the accuracy of the new measure. Also the elapsed time (not reported here) was much less with the new measure: once the values of the projection functions are pre-computed, the efficiency of the classification, which depends on the similarity computation gains a lot of speed-up.

The usage of all concepts for the feature committee F made the measure quite accurate, which is the reason why the procedure resulted quite conservative as regards inducing new assertions. In many cases, it matched rather faithfully the reasoner decisions (the top k nearest neighbors had null distance w.r.t. the query instance). Namely, we found that a choice for lower values for k could have been made, for in many cases the decision on the correct classification was easy to make even on account of fewer (the closest) neighbor instances. This yielded also that the likelihood of the inference made (see Eq. 2) turned out quite high.

Table 3. Results with alternative indices: average ± standard deviation and [min.;max.] intervals.

	match r.	commission e.r.	omission e.r.	induction r.
FSM	94.51 ± 6.63 [73.00;100.00]	5.49 ± 6.63 [0.00;27.00]	0.00 ± 0.00 [0.00;0.00]	0.00 ± 0.00 [0.00;0.00]
SWM	85.38 ± 5.69 [75.70;98.30]	0.00 ± 0.00 [0.00;0.00]	2.68 ± 0.92 [0.90;4.30]	11.95 ± 5.37 [0.90;20.90]
SCIENCE	97.31 ± 1.97 [94.60;99.40]	0.00 ± 0.00 [0.00;0.00]	0.98 ± 0.61 [0.30;1.80]	1.71 ± 1.41 [0.30;3.60]
NTN	88.06 ± 6.95 [74.60;95.40]	0.00 ± 0.00 [0.00;0.00]	2.12 ± 0.77 [0.30;3.40]	9.83 ± 7.12 [4.30;24.30]
SWSD	85.40 ± 4.96 [74.50;92.20]	0.00 ± 0.00 [0.00;0.00]	4.76 ± 1.86 [2.70;8.70]	9.84 ± 3.97 [4.00;19.00]
FINANCIAL	93.34 ± 11.55 [54.80;99.70]	6.30 ± 11.55 [0.00;44.70]	0.01 ± 0.03 [0.00;0.10]	0.35 ± 0.06 [0.30;0.50]

Cases of induction are particularly interesting because they suggest new assertions which cannot be logically derived by a deductive reasoner and they might be used to *complete* a knowledge base [2], e.g. after being validated by an ontology engineer. Eq. 2 should be employed to assess the likelihood of the candidate assertions and hence decide on their inclusion in the ABox.

5 Conclusions and Outlook

This paper explored the application of a distance-based procedure for semantic search to knowledge bases represented in OWL. To this purpose, a novel family of language-independent semantic pseudo-metrics was exploited. Specifically, these measures were integrated in a nearest neighbor search procedure which can be employed for solving approximate retrieval problems.

This turns out to be more effective w.r.t. purely logical methods, especially in the presence of incomplete (or noisy) information in the knowledge bases. Experiments made on various ontologies showed that the method is quite effective and also robust since it seldom made commission errors during the various runs. As expected for an instance-based learning method, the overall performance depends on the number (and distribution) of the available training instances.

As regards the dissimilarity measures, we argue that more efficiency may be reached when statistics (on class-membership) are maintained by the KBMS [8]. Besides, so far, the subsumption relationships among concepts in the feature committee are not explicitly exploited, which might likely make similarity measurements more accurate.

Further developments can be also foreseen as concerns the choice of good feature committees. Namely, since measures are very dependant on this choice, some immediate lines of investigations arise: studying how to maintain limited-sized concepts committees, yet saving those sets which altogether are endowed of a real discriminating power.

Randomized optimization procedures can be used to learn maximally discriminating sets of features, by allowing the composition of concepts through the specific constructors made available by the representation language of choice [7]. This can be accomplished especially well when large sets of individuals are available for the ontologies. Namely, part of the entire data can be drawn in order to learn optimal feature sets, in advance with respect to the next stage.

As mentioned, the measures can be adopted in other instance-based machine learning methods which can be applied to several further tasks. For instance, the measures have been exploited in a hierarchical conceptual clustering algorithm where clusters would be formed grouping individual resources on the grounds of their similarity [6].

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[2] F. Baader, B. Ganter, B. Sertkaya, and U. Sattler. Completing description logic knowledge bases using formal concept analysis. In M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 230–235, Hyderabad, India, 2007.

[3] A. Borgida, T.J. Walsh, and H. Hirsh. Towards measuring similarity in description logics. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Working Notes of the International Description Logics Workshop*, volume 147 of *CEUR Workshop Proceedings*, Edinburgh, UK, 2005.

[4] C. d'Amato, N. Fanizzi, and F. Esposito. Reasoning by analogy in description logics through instance-based learning. In G. Tummarello, P. Bouquet, and O. Signore, editors, *Proceedings of Semantic Web Applications and Perspectives, 3rd Italian Semantic Web Workshop, SWAP2006*, volume 201 of *CEUR Workshop Proceedings*, Pisa, Italy, 2006.

[5] W. Emde and D. Wettschereck. Relational instance-based learning. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning, ICML96*, pages 122–130. Morgan Kaufmann, 1996.

[6] N. Fanizzi, C. d'Amato, and F. Esposito. A hierarchical clustering procedure for semantically annotated resources. In R. Basili and M.T. Pazienza, editors, *Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence, AI*IA2007*, volume 4733 of *LNAI*, pages 266–277. Springer, 2007.

[7] N. Fanizzi, C. d'Amato, and F. Esposito. Induction of optimal semi-distances for individuals based on feature sets. In D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, A.-Y. Turhan, and S. Tessaris, editors, *Working Notes of the 20th International Description Logics Workshop, DL2007*, volume 250 of *CEUR Workshop Proceedings*, Bressanone, Italy, 2007.

[8] I. R. Horrocks, L. Li, D. Turi, and S. K. Bechhofer. The Instance Store: DL reasoning with large numbers of individuals. In V. Haarslev and R. Möller, editors, *Proceedings of the 2004 Description Logic Workshop, DL 2004*, volume 104 of *CEUR Workshop Proceedings*, pages 31–40. CEUR, 2004.

[9] M. Sebag. Distance induction in first order logic. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming, ILP97*, volume 1297 of *LNAI*, pages 264–272. Springer, 1997.

[10] P. Zezula, G. Amati, V. Dohnal, and M. Batko. *Similarity Search – The Metric Space Approach*. Advances in database Systems. Springer, 2007.

Talia: A Framework for Philosophy Scholars

Michele Nucci[1], Stefano David[1], Daniel Hahn[2], and Michele Barbera[2]

[1] Semedia Group - 3mediaLabs
Università Politecnica delle Marche
s.david@univpm.it, mik.nucci@gmail.com
[2] NET7
Pisa
(barbera|hahn)@netseven.it

Abstract. In this paper we present Talia, a novel implementation of a semantic digital web library system, which is part of the Discovery project. Talia deploys Semantic Web technologies and uses computational ontologies for the organisation of knowledge, which can help the definition of a state-of-the-art research and publishing environment for philosophy. Talia will provide an innovative and flexible system which enables data interoperability and new paradigms for information enrichment, data-retrieval and navigation.

1 Introduction

The human sciences have been traditionally hesitant in picking up new technologies and publication channels, instead relying on traditional printed publications. However, the search for and the acquisition of manuscripts and other source material can be difficult, expensive and time-consuming. When researchers do not have access to the original writings of their field they have to rely on secondary material, which may not be accurate and may even lead to invalid results, as describe in [1].

The Discovery Project[3] will create a distributed digital online library for the special needs of philosophers. It will contain original texts and manuscripts by Wittgenstein, Nietzsche, modern and presocratic philosophers.

In addition, the philosophers will be able to publish and peer review new material online. This gives the research community immediate access to new results and removes the burden that the traditional publishing model put on the dissemination of content.

The Discovery library will be built using the Talia system, which is being developed specifically for the Discovery project. Talia will be a distributed digital library system, consisting of a *federation* of interoperable web sites (nodes), each of which will contain content on a specific topic.

Talia uses computational ontologies to organise information and Semantic Web technology[4], like RDF(S) and triple-stores. This technology allows Talia to

[3] http://www.discovery-project.eu/

[4] See http://www.w3.org/2001/sw/ for Semantic Web initiative, related technologies and other W3C standards.

provide excellent data interoperability; it also allows to create a flexible framework for information enrichment and data retrieval that suits the specific needs of research communities. Talia will be able to use different ontologies for each node.

A flexible user interface framework will allow the content providers to configure the navigation according to their own ontologies for each node. By using the Ruby on Rails[5] framework and standard semantic technology Talia will be able to provide a novel way to quickly develop and deploy digital libraries for specialised needs.

Compared to existing digital libraries, Talia will adapt much better to changing requirements and heterogeneous metadata. Compared to other digital library systems, Talia benefits from a much better integration between the semantic technology and the user interface framework.

The paper is organised as follows. In Section 2 we briefly present the Discovery project, in Section 3 we introduce computational ontologies and in Section 4 we describe some general use-cases and requirements which have guided the Talia design and development. Finally, in Section 5 we will present the Talia architecture, focusing on its *semantic software layer*, based on Semantic Web technologies.

2 The Discovery Project

The Discovery project aims at the creation of a federation of interoperable web sites designed to aid humanities' scholars working on digital collections. The content of the Discovery federation is related to ancient and modern philosophy, but the model can easily be applied in other fields. The technical infrastructure underlying Discovery consists of two main components:

- A federation of institutional archives or content providers;
- A peer-to-peer (P2P) network of personal desktop applications;

The federated web sites (nodes) represent the foundation of the Discovery approach. These domain-specific archives and *publishers* contain the digitalised content. Each node is maintained by a different institution and acts as a content-provider and as an Open Access publisher. Each is dedicated to a specific author or theme of ancient and modern philosophy.

The sites contain both reproductions of *primary sources* and *scholarly contributions*. Primary sources are the principal subject of research by a specific community; in the Nietzsche community, for example, these are the writings of Nietzsche himself. Scholarly contributions are scientific works that contribute to the research of the subject at hand. These could be essays on the primary sources, reviews of other contributions, or other research results.

[5] `http://www.rubyonrails.org/`

As Open Access publishers, the nodes accept new secondary sources, or *contributions* related to the primary sources of their scholarly community. Contributions are peer reviewed by the board of reviewers of the node and then published on the web site, side by side with their related primary sources.

One of the characteristic traits of Discovery is that the relations between primary and secondary sources are explicitly described and expressed in a machine readable way. A "Discovery structural ontology" has been created; it will be shared among all Discovery nodes. The structural ontology is the basic building block of the Discovery network as it represents the lowest common denominator of interoperability among the archives of the participating data providers.

Knowledge in Discovery will be organised by means of suitable computational ontologies, which will be used both as a means for organising knowledge concerning each site, and to relate knowledge from different sites.

3 Computational Ontologies

Ontologies have become popular in computer science as a way of organising information. This connotation of *ontology* differs to the traditional use and meaning it has in philosophy, where ontologies are considered as "A system of categories accounting for a certain vision of the world" [2]. There have been a lot of definitions of ontology in computer science (or computational ontologies), which extended and refined the one provided by Gruber [3], who defined an ontology as "a specification of a conceptualisation."[6]. We will stick to the following one, extracted from Guarino's definitions (see for example [4]): A computational ontology is *"A Formal, Partial Specification of a Shared Conceptualisation of a world (domain)"*. Intuitively, according to this definition, a computational ontology is a set of assumptions that defines the structure of a given world or domain of interest, allowing different people to use the same concepts to describe that domain.

This definition incorporates the most relevant characteristics for an ontology to be useful:

- *Formal.* An ontology needs a robust underlying theory for reasoning.
- *Partial specification.* It has a representative aspect (i.e., it presents and organises knowledge about a domain), but shall not pretend to describe every detail of the domain it should represent.
- *Shared.* An agreement shall be reached on an ontology by all interested parties, so that they can share the same ontology, and the same concepts defined in it, without the need to reinvent the wheel.
- *Conceptualisation of a world.* It should meaningfully represent a world of interest.

[6] A more up-to-date definition, together with additional readings can be found at `http://tomgruber.org/writing/ontology-definition-2007.htm`

4 Requirements and sample use cases

While the requirements for the Talia system vary significantly between the user communities, there are some general requirements that are valid for all Discovery partners. In the following we present some of the central features of Talia. These are the ones that every typical scholar using Talia will need.

4.1 General Requirements

- The users need to be able to access the primary sources (or their reproductions) online. Previously, these were often difficult and/or expensive to obtain.
- There should be a *context* to the documents that are being viewed. This means that the researcher should be able to quickly find documents that are related to the item at hand. Related, in this case, can mean for example "is cited by", "was written in the same period", "shares a common topic". This context depends heavily on the community that uses the archive.
- The user will not only be interested in relations to items in the current archive, but in relations to items in other archives as well.
- If an author wants to cite an item from the archive in her work, she must be able to refer to this item in a unique way. She also needs to have the confidence that the archive is *permanent* – so that she can cite a document without fear that it may be renamed, removed, or altered in the future.
- Researchers will be able to publish new content on the system. In order to be a viable alternative to traditional publication channels, the system must ensure at least the same level of quality. Thus, a peer review workflow has to be established that ensures the same standards as in printed publications.

4.2 Example Use Cases

One of the most striking requirements for Talia is that it needs to provide a unified platform for user communities with very diverse needs. Therefore, we will illustrate three simple use cases for three diverse user communities[7]. From these use cases, we can then explore the technical decisions made in the design of Talia.

"The Manuscript Comparers" The archive contains a lot of hand-written manuscripts by a certain philosopher. The users want to browse these like books, navigating by chapters and pages. Unfortunately, the philosopher's handwriting is unreadable for normal people, so the archive also contains transcriptions of the different paragraphs. The user wants to click on a paragraph in a manuscript and see the correct transcription. Finally, the philosopher often wrote different "versions" of the same thought in different places. The users are interested in comparing those versions, so they want to find all different versions of the current paragraph.

[7] For the sake of explanation, these use cases have been greatly simplified from the real world use cases

"The Topic Searchers" The archive contains the works of a different philosopher as an electronic text. The user wants to annotate each paragraph with the philosophical topics it deals with and then navigate the contents by topic. However, the users often disagree on the topics, so they also want to annotate the topic with their different opinions.

"The Movie Enthusiasts" The archive contains movie files concerning philosophical topics. The users want to view the videos and while watching there should be links to the current section of the video is about. All the videos refer to documents that are stored in other archives, and different parts of the videos can refer to different things.

5 Talia

The Talia platform stores digital objects, called *sources*, which are identified by their unique URLs. Each source can have one or more data files attached to it, and the system stores information about the source objects. Talia also provides a user interface framework that allows to build web interfaces for different requirements, using modular components called *widgets*.

The communication between different Talia servers will be done through a REST interface, an approach proposed by R. Fielding in [5].

Talia is developed using the Ruby on Rails[8] web application framework and it will be publicly available from the Talia web site[9].

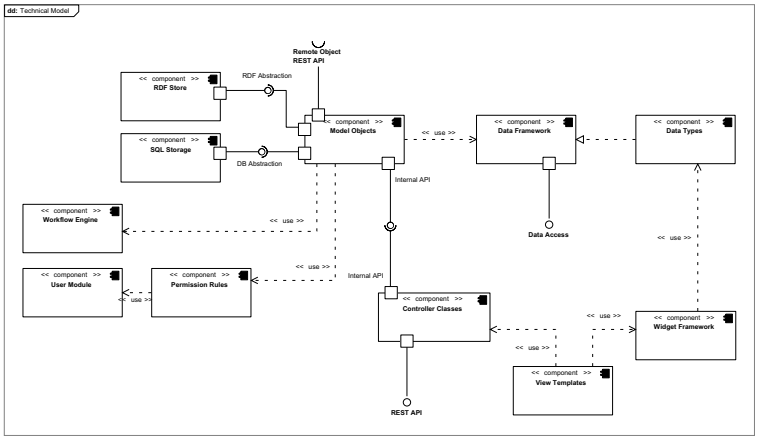


Fig. 1. Overview of the Talia system architecture

[8] http://www.rubyonrails.org
[9] http://www.talia.discovery-project.eu/

Architecture overview. Figure 1 provides an overview of the general architecture of the system and show the components of which Talia is composed:

Model Objects. The model classes provide an interface to the sources; they also contain a query mechanism to locate and retrieve sources. They can retrieve data from remote Talia nodes.

RDF Store. It contains the metadata on the sources. The data is stored as RDF triples, allowing to store any kind of descriptions about the sources.

SQL database. It contains all the information for which classic ACID requirements (see for example [6]) exist. This would contain things like the current workflow state of a source, user permissions, and the like.

Data Framework. It provides an API to access the actual data stored in the system. Different *datatypes* provide more advanced functionality for each document type, e.g., the ability to read lines from text files or a video stream from a multimedia file.

Controller Classes They contain the logic that connects the model objects with the user interface and drives the application. The controllers also provide an easy to use REST interface that allows other software to remotely interact with the library.

Widget Framework. It allows the development of widgets that can be used to build customised user interfaces. Widgets can have their own behavioural logic.

View Templates. The web site templates that render the user interface of the application. These use the data set up by the controllers, and can use widgets to place common interface elements.

Other The *Workflow Engine*, *User*, and *Permission Modules* contain logic that is need to manage the library. They are without the scope of this short introduction.

User interface. Talia will include its own framework for deploying *widgets*. These modular interface elements can be packaged independently and used as building blocks for the application's user interface. A number of common widgets will be shared by all archives, and there will be also specialised widgets for single archives that have specific needs. The archives may also modify the HTML rendering templates to customise their site's appearance.

Figure 2 shows an example of such a widget: The *context sidebar* presents the user with links to all documents related to the source currently viewed.

Dynamic Contextualisation. The *dynamic contextualisation* provides a method for different Talia nodes to exchange data. The mechanism functions as follows:

If a node adds a relation to a document on different ("foreign") node, it writes the information to its internal RDF store. Then it notifies the foreign node; and the foreign node will decide independently on whether or not to add the relation to its local RDF store.

Fig. 2. Context sidebar in the Talia application

"Foreign" documents will be presented in the navigation in the same way as local documents. If the user follows a link to a foreign document, she will be taken to the other archive's site and she will be able to continue to browse there. If that side has accepted the contextualisation request, a back link to the original site will also be provided.

5.1 Requirements for Talia's Semantic Layer

The most characteristic aspect of Talia is that, for the first time, at least in the field of humanities, all public interfaces will be based on Semantic Web standards, which enables interoperability within the Talia Federation and with other systems.

According to the original idea and to meet the requirements coming from the analysis of use-cases, Talia will have to include a *semantic software layer*, which will be based on the following general characteristics:

- *Making metadata remotely available.* The metadata content of the archives will be made available through interfaces similar to those of URIQA[10], which will allow automated agents and clients to connect to a Talia node and ask for metadata about a resource, retrieving its description and related metadata (e.g., the author, the resource type, etc.). This type of services enables numerous types of digital content reuse.
- *Querying the system using standard query-languages.* It is possible to directly query the Talia Federation using the SPARQL Semantic Web Query Language. SPARQL provides a powerful and standard way to query RDF repositories and enables the merging of remote data sets.
- *Transformation of encoded digital contents into RDF.* The semantic software layer will provide tools to transform textually encoded material, for example the transformation of manuscripts in TEI[11] format into RDF.
- *Managing structural metadata.* "Structural" metadata describes the structure of a document (e.g., the format of a document, its publisher, etc.). It will be necessary to provide tools to manage these kinds of metadata.

[10] `http://sw.nokia.com/uriqa/URIQA.html`
[11] `http://www.tei-c.org/`

The Talia semantic software layer will also provide facilities to *enrich* the *primary sources* and the *secondary sources* with semantic information, the type and format of which depends on the kind of source and on the user community that works on the data. For example, in some archives, different "versions" of the same paragraph may exist in different documents. In order to allow the users to follow the evolution of that particular philosophical thought, the relations between these different versions must be captured by the system.

Additional trouble may arise when archives use different formats for the annotation of documents. For this reason, Talia will have to provide dedicated ontologies and tools to describe information coming from different users and the relations among them.

5.2 The Semantic Layer in Talia

In order to fulfil the requirements of the use-cases in section 4.2 and to develop the semantic software layer in Talia, it was necessary to define its general architecture as well as to choose which of the already existing Semantic Web technology to use. This choice has been influenced by the programming language used to develop the Talia system.

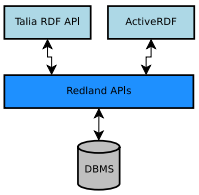


Fig. 3. General architecture of the Talia Semantic Layer

Figure 3 shows the general architecture of the Semantic Web Layer in Talia. We decided to use Redland RDF [7] as main tool to manipulate RDF data. Redland RDF provides a set of free software libraries to manage and manipulate whole RDF graphs, single triples, URIs, and literals. It also supports different storage systems, so we might choose to replace MySQL at any time without the necessity to change the Talia source code. For the first prototype of Talia we chose to use the MySQL DBMS.

On top of Redland RDF we use ActiveRDF [8], a library to access and manipulate RDF data in Ruby. ActiveRDF also provides an abstract query engine which is independent of the data source and the specific query language.

Currently ActiveRDF does not meet all requirements from the use-cases. It was therefore extended and an additional high-level API was developed. This API can interact directly with the low-level API provided by Redland. Since

ActiveRDF does not natively support the use of DBMS like MySQL or PostgreSQL, we designed and developed new dedicated data adapters. Moreover, a basic API was developed, which provides dedicated methods to delete single RDF statements on Talia sources, as well as methods to remove specific sources and their related data.

A flexible system like Talia requires facilities to easily work with properties and related values. New tools to manipulate RDF predicates have been developed, introducing a new dedicated syntax to speed up the creation, modification, and deletion of properties and groups of properties.

Digital contents of Talia archives structured by RDF can be exported in different ways. We provided Talia system with flexible and optimised tools, which currently allow RDF data export/import in RDF/XML and N-Triples formats. The feature for exporting RDF data could be easily extended included others formats like N3, plain-text, etc.

To organise information and support the semantic enrichment of the content provided by partners with different needs, it was decided not to develop a common ontology for the whole project. Instead, we defined only a very broad *structural ontology*, which contains only some general concepts and relations to link documents, and each Discovery content partner will develop its own *domain ontology* for the specific needs of each archive. For this reason we also developed basic tools to manage and load ontologies for the first prototype. These tools will be upgraded for the final version of Talia, including features which will allow to remove old ontologies from the system and to synchronise and update different versions of the same ontology.

6 Related Works

Talia is directly related to the Hyper platform, which was used for the HyperNietzsche[12] archive. HyperNietzsche and the Hyper platform are described in [1].

HyperNietzsche provides a way of navigating the digital Nietzsche archive. The software was specifically developed for the community of Nietzsche scholars, with many different features tailored for the needs of the Nietzsche researchers.

Hyper did not use semantic web technology extensively, and was a highly specialised solution. Modifying this code base for the needs of the Discovery partners will not be feasible; thus the Hyper platform will be retired during the Discovery project and will be superseded by Talia.

The authors are also aware of other semantic digital library projects, such as JeromeDL [9], BRICKS [10] and Fedora [11]. Especially the latter ones provide robust frameworks for digital library applications. However, the focus is mainly on providing an infrastructure; none of the projects offers the kind of strong integration between the user interface framework and the semantic library backend that is required for Discovery.

[12] `http://www.hypernietzsche.org/`

7 Conclusion and future work

We have presented a prototype of the Talia platform, an application which aims at easing scholarly research in philosophy. We have also presented, as a use case, the motivation for the development and implementation of Talia and the role played by computational ontologies and Semantic Web technologies. By using ontologies to organise information and to formally describe the relations among the digital contents it has been possible to develop a very adaptable, state-of-the-art research and publishing environment for the philosophy.

A lot of effort is currently put into the development of Talia, and the release of an alpha version is planned for the beginning of 2008. A preliminary date for a beta release is the end of June, 2008.

Acknowledgements

This work has been supported by Discovery, an ECP 2005 CULT 038206 project under the EC eContentplus programme.

References

1. D'Iorio, P.: Nietzsche on new paths: The hypernietzsche project and open scholarship on the web. In Fornari, C., Franzese, S., eds.: Friedrich Nietzsche. Edizioni e interpretazioni. Edizioni ETS, Pisa (2007)
2. Calvanese, D., Guarino, N.: Ontologies and Description Logics. "Intelligenza Artificiale - The Journal of the Italian Association for Artificial Intelligence" **3**(1/2) (2006)
3. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition **5**(2) (1993) 199–220
4. Guarino, N.: Formal Ontology and Information Systems. In Guarino, N., ed.: 1st International Conference on Formal Ontologies in Information Systems, IOS Press (1998) 3–15
5. Fielding, R.T.: Achitectural Styles and the Design of Network-based Software Architectures. PhD thesis, UC Irvine (2000)
6. Gray, J.: The transaction concept: Virtues and limitations. In: 7th International Conference on Very Large Data Bases, 19333 Vallco Parkway, Cupertino CA 95014, Tandem Computers (1981) 144–154
7. Beckett, D.: The Design and Implementation of the Redland RDF Application Framework. In: 10th International World Wide Web Conference (WWW2001), Hong Kong. (2001)
8. Oren, E., Debru, R., Gerke, S., Haller, A., Decker, S.: ActiveRDF: Object-Oriented Semantic Web Programming. In: 16th International World Wide Web Conference (WWW2007), Banff, Alberta, Canada. (8-12 May, 2007) 817–823
9. Kruk, S., Woroniecki, T., Gzella, A., Dabrowski, M., McDaniel, B.: Anatomy of a social semantic library. In: European Semantic Web Conference. Volume Sematic Digital Library Tutorial. (2007)
10. Risse, T., Knežević, P., Meghini, C., Hecht, R., Basile, F.: The bricks infrastructure - an overview. In: The International Conference EVA, Moscoww (2005)
11. Fedora Development Team: Fedora open source repository software: White paper. White paper, Fedora Project (2005)

Towards Social Semantic Suggestive Tagging

Fabio Calefato, Domenico Gendarmi, Filippo Lanubile

University of Bari,
Dipartimento di Informatica,
Via Orabona, 4, 70126 - Bari, Italy
{calefato,gendarmi,lanubile}@di.uniba.it

Abstract. The organization of the knowledge on the web is increasingly becoming a social task performed by online communities whose members share a common interest in classifying different types of information for a later retrieval. Collaborative tagging systems allow people to organize a set of resources of interest through unconstrained annotations based on free keywords commonly named tags. Suggestive tagging techniques support users in this organization process and have shown to be helpful also in fostering a quick convergence to a shared tag vocabulary.
In this paper, we propose a tag recommender which relies on the content analysis of the resource to be tagged, as well as on the personal and collective tagging history. The main contribution of this work is a model which combines semantic content analysis methods with existing suggestive tagging techniques. The expected benefit is the improvement of the user experience in social bookmarking systems, and more generally in collaborative tagging systems.

Keywords: collaborative tagging, folksonomy, recommender system, semantic web, content analysis, suggestive tagging, social bookmaking.

1 Introduction

The phenomenon of Web 2.0 [9] has led to the development of many tools, which have succeeded in making the task of knowledge organization more attractive to a broader audience. Tools for accomplishing this activity, such as collaborative tagging systems, harness the power of virtual communities and have been shown effective in gathering quickly large amounts of information directly generated by users.

Collaborative tagging systems allow people to organize a set of resources, by annotating them with tags through a browser. Tags can be regarded as free keywords used by people to label resources of interest. The activity of labelling is called tagging, as it consists of attaching one or more tags to the resource. Although this tagging activity is accomplished individually, while using the system, everyone can see who else is participating by observing others' tagging behaviours. This tight feedback loop makes these systems social and the result is a collection of annotations, also called folksonomy [14]. Unlike top-down centralized classification approaches, folksonomies have revealed a noteworthy ability in adhering to the personal way of

thinking [4]. The opportunity of using free tags with no restrictions allows users to express their own perspective on the annotated resource. Therefore, these annotations can become a reliable indicator of interests and preferences of the active participants in such systems.

To date, most collaborative tagging systems provide a limited support to users in the annotation process, as they typically recommend tags by arranging suggested tags in a tag cloud that emphasizes tags on the basis of their popularity: the bigger the font, the more used the tag. By suggesting to the user his/her most used tags, as well as the most popular tags in the whole community, this form of tag recommendation takes into account both the personal and social dimension of folksonomies. Nevertheless, this approach falls short of considering the semantic dimension for the content of the resource that is going to be annotated.

We acknowledge that suggesting meaningful tags to a user, according to personal and social interests, can enhance the user experience and augment the number of active participants in the annotation process. However, we argue that the content analysis of the resources can significantly improve the accuracy of suggestive tagging, thus, fostering a quick convergence to a shared tag vocabulary and limiting the tag synonymy issue [5].

In this paper, we propose a tag recommender which relies on the semantic analysis of the resource content which is going to be annotated, as well as on the personal and collective tagging history. Such an approach is able to address the typical cold-start problem affecting recommender systems [11]. In fact, our recommender system will be able of suggesting tags to users who have not yet tagged any resource, by putting forward tags which are popular in the community. Further, when there are resources not yet tagged by anyone in the community, our recommender will suggest tags which have been gathered through a semantic analysis of the resource content.

The main contribution of this work is a model which combines semantic content analysis methods with existing suggestive tagging techniques. The expected benefit is the improvement of the user experience in social tagging bookmarking systems, and more generally in collaborative tagging systems.

The remainder of the paper is structured as follows. Section 2 describes how the content analyzer works and how it is going to be integrated in the proposed tag recommender system. In Section 3 we present our model through four typical scenarios which can take advantage from a mix of semantic content analysis and traditional suggestive tagging. Section 4 surveys novel related work concerning suggestive tagging in folksonomies. Finally, Section 5 draws conclusions and points out some challenges we are going to address in the near future.

2 Content Analysis for Semantic Tag Suggestion

The idea behind applying content analysis for semantic suggestive tagging is to provide a user who wants to tag a resource, not only with relevant words extracted from a resource, but also with a set of synonyms. In this way, other than fostering tag convergence, we also increase the probability of suggesting tags that fit better to users' personal way of thinking, without affecting the meaning.

To make this possible, a word sense disambiguation (WSD) algorithm is needed, which can assing a word *w* occurring in a given resource (e.g., a web document), to the appropriate sense, according to the context (i.e., the set of words that precede and follow *w*). Then, once the appropriate sense of a word *w* is identified, a dictionary or a lexical ontology can be used to find its synonyms, and to provide the user with a set of recommended tags alternative to *w*.

META (MultilanguagE Text Analyzer) [3] is a tool developed at the University of Bari, which implements an algorithm to perform WSD on text documents in a variety of formats (e.g., pdf, doc). The tool has also been used by the Item Recommender system (ITR) to learn sense-based user profiles [12]. In addition to performing the basic content analysis tasks (e.g., stop-words elimination, stemming), META is also able to analyze different parts of text documents, called slots. For instance, when processing papers from a conference proceedings, META performs content analysis on the title, abstract, and body, separately. Furthermore, META relies on WordNet for obtaining a sense inventory. Thus, after performing the WSD, META returns the unique id in WordNet (called offset) of the correct sense identified, for each word extracted. Our idea is to use offsets to retrieve from the lexical ontology the whole set of synonyms (SYNSET, in short) for each relevant word extracted that will be suggested as a tag.

3 The Suggestive Tagging Model

In [1, 9] a generic collaborative tagging system is defined as a tripartite 3-uniform hypergraph $F=(N,E)$ where $N=U \cup T \cup R$ is the union of three disjoint sets of entities, namely a set of registered users (U), a set of applied tags (T), and a set of annotated resources (R). Furthermore, we define $E=\{(u,t,r) \mid u \in U, t \in T, r \in R\}$ as the set of all the annotations that compose the folksonomy. Given the above definitions, we can define a typical social bookmarking system as a folksonomy where R is replaced by a set of bookmarks B, pointing to resources in R.

For each entity within such system, we can also discern among different kinds of tags, users and bookmarks. Given a user u and a selected bookmark b we can identify three sets of tags:

- *PersonalTags(u)*, all the tags assigned by u to all bookmarks.
- *SocialTags(b)*, all the tags assigned by all users to b.
- *SemanticTags*(b), all the tags extracted by analyzing the content of the resource pointed by b.

Depending on the amount of tags adopted by a single user, we can also discriminate a user as *novice*, if he/she has no tags, or as *expert*, when he/she has started to annotate bookmarks using tags. The definition of novice includes hence both new users just registered to the system and users registered for a while but loath in using tags to save bookmarks. Finally, a bookmark can be categorized as *tagged* if it has been annotated with at least one tag, or *untagged* if there are no associated tags

According to the above definitions, we illustrate four scenarios which depict how a user can be supported by tag recommendations in his/her task of saving a bookmark (Table 1). Because of the huge number of registered users and the availability of

suggestive tagging features, we use del.icio.us[1] as the reference system for the proposed approach. In the following, we consider four users, namely John and Dexter (as novices), and Alice and Bea (as experts). We also assume the Collaborative Development Group Research page[2] as an untagged bookmark and SWAP 2007 home page[3] as a tagged one.

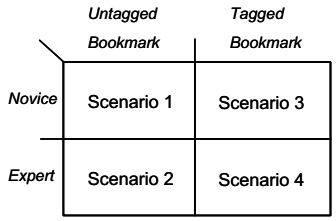


Table 1. Four exemplary scenarios

Scenario 1: A Novice user saving an Untagged Bookmark
John is going to save Collaborative Development Group Research page as a bookmark. Being a novice user, he has no tags yet. In this scenario del.icio.us cannot provide any suggestion because John has no personal tags and nobody else has saved this bookmark yet (Figure 1). However, even if no suggestions are available from either personal or social tags, according to our view, it is still possible to support John with tag recommendations by providing *Semantic Tags* as the output of the content analysis (Figure 2). In particular, META extracts the following words from the Collaborative Development Group Research page (for the sake of space, we limit to six the number of words extracted):

- software (occurring 27 times)
- distributed (21)
- 2007 (16)
- 2006 (10)
- conference (7)
- workshop (5)

In addition, META identifies the correct sense for the words extracted. For instance, for the word "conference" the sense extracted from the inventory is "prearranged meeting, especially with a formal agenda", while the other offset (i.e., "association of sport teams") is just skipped. Finally, the whole set of semantic tags is obtained by also retrieving from WordNet the SYNSET for "conference" (i.e., {meeting}).

[1] http://del.icio.us
[2] http://cdg.di.uniba.it/index.php?n=Research.HomePage
[3] http://www.swapconf.it/2007/

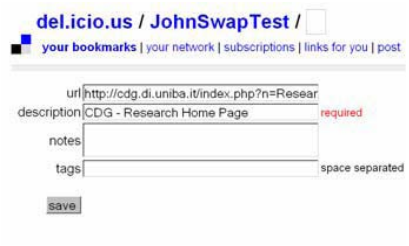


Figure 1. No tags suggested by del.icio.us

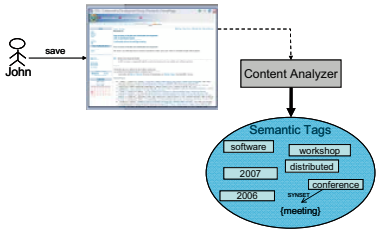


Figure 2. Semantic Tags as output of the content analysis

Scenario 2: An Expert user saving an Untagged Bookmark

Alice is an expert user and thus, she has already used some tags to annotate bookmarks in del.icio.us. Now she wants to save a bookmark never tagged before in the system. In such a case, del.icio.us can suggest only those tags which have been already adopted by Alice, even though most of them might be inappropriate (Figure 3). Instead, in this scenario, we argue that a hopefully more useful set of tags can be suggested to Alice by intersecting both Alice's *Personal Tags* (i.e., collaborative, Web2.0, 2007, conference, …) and the *Semantic Tags* extracted from the bookmark (i.e., software, distributed, 2007, conference).

We define the intersection of the two sets as the *Personal Semantic Tags(u,fb)*, i.e., all the tags from a user which have also been obtained from the content analysis of the resource pointed by a bookmark. If the set of Personal Semantic Tags is not empty, these tags will be suggested to Alice as recommended tags, in addition to the remaining *Personal Tags* and *Semantic Tags* (Figure 4).

Figure 3. Personal tags suggested by del.icio.us

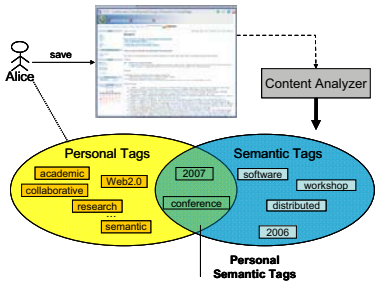


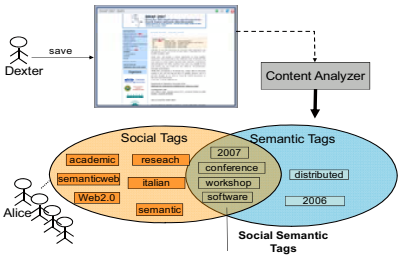
Figure 4. Personal Semantic Tags recommendation

Scenario 3: A Novice user saving an Untagged Bookmark

Dexter is a novice user who has been registered to del.icio.us from two months, but he has never used tags. Now he is going to save the SWAP 2007 home page that has already been tagged by Alice and other users. Typically, del.icio.us suggests only popular tags if the selected bookmark has been annotated by more than one user

(Figure 5). This time, a recommender that implements our approach might benefit from both *SemanticfiTags(b)* and *SocialfiTags(b)*. Assuming that the *SocialfiTags(b)* and the *SemanticfiTags(b)fi*sets are not disjoint, we define the intersection between these two sets as *SocialfiSemanticfiTags(b)* (i.e., 2007, conference, workshop, and software). As in the previous scenario, our social/semantic tag recommender would suggest tags belonging to the intersection as *RecommendedfiTags* and also the remaining *SocialfiTags* and *SemanticfiTags* (Figure 6).

Figuref5.ffPopularftagsfisuggestedfbyfi del.icio.usfi



Figuref6.ffSocialfiSemanticfiTagsfi recommendationfi

Scenariofi4:fiAnfiExpertfiuserfisfisavingfiafiTaggedfiBookmarkfi
Bea is an expert user who uses del.icio.us to save her bookmarks on a daily basis, and thus she has a large set of personal tags. She is now going to save and tag the SWAP 2007 home page, which has been also tagged by both Alice and Dexter, among the others. In such a scenario, del.icio.us would provide Bea with both *PopularfiTags*, i.e., the most used tags for that bookmark by other users, and *RecommendedfiTags*, i.e., the personal tags that have been also used by others for that bookmark (Figure 7). Other than suggesting these two sets of tags, according to our approach, it is also possible to exploit the *SemanticfiTags(b)* obtained through the content analysis of the SWAP 2007 home page. In this scenario, Bea is supported with four different kinds of tags recommendation:

- *PersonalfiSemanticfiTags(u,fib)*
- *SocialfiSemanticfiTags(b)*,
- *SharedfiTags(b,fiu):fi* those tags belonging to the intersection between *PersonalfiTags(u)* and *SocialfiTags(b)*
- *SemanticfiSharedfiTags(u,fib):fi*those tags belonging to the intersection of all the above available sets of tags, namely *PersonalfiTags(u),fiSocialfiTags(b),fi* and*fiSemanticfiTags(b)* (Figure 8).

We argue that the quality of tag recommendations provided to Bea could be significantly improved by presenting intersections which limit the information overload.

Figure 7. Recommended and popular tags suggested by del.icio.us

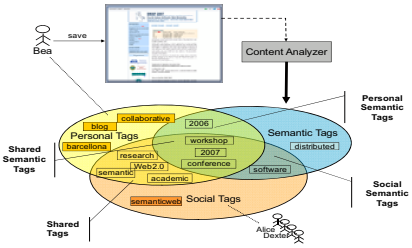


Figure 8. Four different kinds of tags recommendation

Finally, in the following table, we summarize all the recommended tags, according to each presented scenario. In particular, strongly recommended tags, i.e., those tags that, in our view, are hopefully more useful, are shown in bold (Table 2).

	Untagged Bookmark	*Tagged Bookmark*
Novice	Semantic Tags	**Social Semantic Tags** Semantic Tags Social Tags
Expert	**Personal Semantic Tags** Semantic Tags Personal Tags	**Semantic Shared Tags** **Shared Tags** **Social Semantic Tags** **Personal Semantic Tags** Semantic Tags Social Tags Personal Tags

Table 2. Recommended tags for each scenario

4. Related Work

Suggestive tagging within folksonomies is a rather novel field of research [8]. The evidence of such a novelty is the quite sparse literature related to the state of the art on tag recommendations.

One existing approach to tag suggestions is referred to as *selection of tags,* which indicates that systems select a small number of tags to display, among the sheer size of terms already associated to an item. With respect to this approach, Sen et al. [13] investigated how different algorithms for selecting tags to display, influence users' personal vocabularies while annotating movies in a movie recommendation system.

A similar approach was also proposed by Xu et al. [16], who defined a set of general criteria for a good tag suggestion algorithm, in order to identify the most appropriate tags, while eliminating noise and spam. These criteria, identified through a study of tag usage by real users in My Web 2.0, include high coverage of multiple facets to ensure good recall, least effort to reduce the cost involved in browsing, and high popularity to ensure tag quality.

Based merely on the social dimension of tagging systems is the work of Jaschke et al. [7], who presented two different algorithms for recommending tags. The first algorithm is based on collaborative filtering [11], whereas the second is based on the FolkRank algorithm, defined in [6], and exploits the graph structure of folksonomies. The comparison, performed using two datasets from real–life folksonomies, namely Last.fm and Bibsonomy, showed that the graph-based FolkRank algorithm outperforms collaborative filtering approaches.

Finally, following a similar approach to the one proposed in this paper, Byde et al. [2] described a tag recommender system based on two different resource similarity metrics, which take into account the tag used by one user to annotate resources and their content, respectively. Although this work was the first to introduce content-based methods for recommending tags, it failed to take into account the social dimension of folksonomies (i.e., the community tags) to compute the resource similarity, considering only the personal resources and tags.

5 Conclusions and Future Work

Suggestive tagging fulfils several needs: it helps users in the annotation process, fostering a quick tag vocabulary convergence, and enhances the likelihood of a resource to get tagged. However, current systems suggest tags only on the basis of personal recent use or because of their popularity among the community.

In this paper, we have described a model which combines semantic content analysis methods with existing suggestive tagging techniques. By exploiting semantics of content analysis, provided by the META tool, and social features, built-in in folksonomies, the proposed recommender can address tag recommendations even in borderline cases, such as a user which has never used tags previously or a resource with no associated tags. We also intend to extend META and adapt it for the purpose of performing the content analysis of web resources to be annotated. Web pages organize their content in the HTML <head/> and <body/> slots. The analysis of the content of the <head/> slot, in particular, can offer valuable insights for the purpose of suggesting tags to annotate a web resource. In fact, editing both the <title/>, and the *keywords* and *description* <meta/> slots can be thought as an accurate form of free annotation, because their content reflect just the personal view of the web page creator/maintainer on its whole content.

Our approach to suggestive tagging has been presented in the context of del.icio.us, the most popular social bookmarking system. As future work, we plan to complete the development of the proposed recommender system and perform an explorative experimentation within del.icio.us, having the existing suggested tagging feature as a control group.

Referencesfi

1. Abbattista, F., Calefato, F., Gendarmi D., Lanubile, F.: Shaping personal information spaces from collaborative tagging systems. In B. Apolloni et al. (Eds.): KES 2007/ WIRN 2007, Part III, LNAI 4694, Springer-Verlag Berlin Heidelberg, pp. 728–735, 2007.
2. Byde, A., Wan, H., Cayzer, S.: Personalized Tag Recommendations via Social Network and Content-based Similarity Metrics. In Proceedings of the International Conference on Weblogs and Social Media (ICWSM'07), March 2007.
3. Degemmis M., Lops P., and Semeraro G.: A content-collaborative recommender that exploits WordNet-based user profiles for neighborhood formation. User Modeling and User-Adapted Interaction, 17(3), pp. 217-255, July 2007.
4. Gendarmi D., Abbattista F., Lanubile F.: Fostering knowledge evolution through community-based participation. Proceedings of the 1st Workshop on Social and Collaborative Construction of Structured Knowledge at WWW'07. 2007.
5. Golder, S., Huberman, B.: Usage patterns of collaborative tagging systems. Journal of Information Science, 32(2), pp. 198-208, 2006.
6. Hotho, A Jäschke, R., Schmitz, C., Stumme, G.: Information retrieval in folksonomies: Search and ranking. In: Sure, Y., Domingue, J. (Eds.) ESWC 2006, LNCS, vol. 4011, Springer-Verlag Berlin Heidelberg, pp. 411–426, 2006.
7. Jäschke, R., Marinho, L., Hotho, A., Schmidt-Thieme, L., Stumme, G.: Tag Recommendations in Folksonomies. In J. N. Kok and J. Koronacki and R. López de Mántaras and S. Matwin and D. Mladenic and A. Skowron, (Eds.), Knowledge Discovery in Databases: PKDD 2007, 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, LNCS, vol. 4702, Springer-Verlag Berlin Heidelberg, pp. 506-514, 2007.
8. Marlow, C., Naaman, M., Boyd, D., Davis, M.: HT06, tagging paper, taxonomy, Flickr, academic article, to read. Proceedings of the Seventeenth Conference on Hypertext and Hypermedia. ACM Press, New York, NY, pp. 31-40, 2006.
9. Mika, P.: Ontologies are us: A unified model of social networks and semantics. Proceedings of the 4th International Semantic Web Conference, ISWC 2005, LNCS, Vol. 3729, Springer-Verlag Berlin Heidelberg, pp. 522-536, 2005.
10. O'Reilly T.: What is Web 2.0. Design Patterns and Business Models for the Next Generation of Software. September 2005.
11. Sarwar, B., Karypis, G., Konstan, J., and Reidl, J.: Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th international Conference on World Wide Web (WWW 2001). ACM Press, New York, NY, pp. 285-295, 2001.
12. Schein, A. I., Popescul, A., Ungar, L. H., Pennock, D. M.: Methods and metrics for cold-start recommendations. In Proceedings of the 25th Annual international ACM SIGIR Conference on Research and Development in information Retrieval. ACM Press, New York, NY, pp. 253-260, 2002.
13. Semeraro G., Degemmis M., Lops P., Basile P. Combining Learning and Word Sense Disambiguation for Intelligent User Profiling. In Proceedings of 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, pp. 2856-2861, 2007.
14. Sen, S., Lam, S. K., Rashid, A., Cosley, D., Frankowski, D., Osterhouse, J., Harper, F. M., Riedl, J.: Tagging, communities, vocabulary, evolution. Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work (CSCW 2006). ACM Press, New York, NY, 181-190, 2006.
15. Vander Wal, T. Folksonomy Definition and Wikipedia. November 2005.
16. Xu, Z., Fu, Y., Mao, J., Su, D.: Towards the Semantic Web: Collaborative Tag Suggestions. Proceedings of Collaborative Web Tagging Workshop at 15th International World Wide Web Conference (WWW 2006). 2006.

Okkam*4P*: A Protégé Plugin for Supporting the Re-use of Globally Unique Identifiers for Individuals in OWL/RDF Knowledge Bases*

Paolo Bouquet[1], Heiko Stoermer[1], and Xin Liu[2]

[1] University of Trento,
Dept. of Information and Communication Tech.,
Trento, Italy
{bouquet, stoermer}@dit.unitn.it
[2] JiLin University
Dept. of Computer Science and Technology
Chang Chun, China
xinliujlu@gmail.com

Abstract. In Protégé, any newly created RDF/OWL knowledge base refers to local instances through a *local* URI, which is obtained through the concatenation of the ontology URI, the hash sign # and a local identifier. However, this practice makes data-level integration quite hard, and definitely prevents the straightforward application of RDF graph merging for independently developed knowledge bases, even if they share the same OWL ontology. In this paper, we present a Protégé plugin which supports the systematic reuse of global identifiers for instances in RDF/OWL knowledge base. The plugin is an extension of the Protégé "Individuals" tab. The main difference is that, when an instance is created, the user has a chance of looking for an existing URI for the corresponding individual in a publicly available service called Okkam. The match between the newly created instance and the globally registered individuals is based on a comparison of features of the new and a a simple profile stored in Okkam for all individuals. The plugin is available and tested for Protégé 3.3.1 and 3.4 beta.

1 Introduction

One of the key ideas of the Semantic Web is that the use of a unique identifier (URI) for referring to the same resource will be the basis for enabling the integration of data across autonomous applications and independently created semantic repositories. However, nothing in the infrastructure of the Semantic Web supports content creators to reuse already existing URIs for referring to a

* This work was partially funded by the European Commission under the 6^{th} Framework Programme IST Integrated Project VIKEF - Virtual Information and Knowledge Environment Framework (Contract no. 507173, Priority 2.3.1.7 Semantic-based Knowledge Systems; more information at http://www.vikef.net). The authors would like to thank Daniele Zanoni for his work on the first prototype of Okkam*4P*.

resource which has already been referred to in other applications/repositories. This is true for any type of resource (including abstract resources, like classes and properties), but is especially bad for instances (individuals), as the *ex-post* discovery of identities between instances across knowledge bases is in general more difficult (and less investigated) than discovering mappings between ontology elements. The lack of systematic support for the reuse of URIs leads to a flooding of identifiers, which makes data integration on the Semantic Web very hard and error prone.

The issue of identity and identification in the (Semantic) Web has been discussed and analyzed from different perspectives in the past, in fact two scientific workshops have been dedicated to the topic[3], the proceedings of which are a great source of insight into the diverse points of view on the issue[4]. Additionally, works such as Kent's [9, 10] from a historical perspective, Gangemi and Presutti's [6, 7] as well as the efforts and discussions within the W3C [2, 1, 8], make evident that there is plenty of ambiguity about the use and semantics of a URI. There are several options of what a URI can identify, opinions about whether a URI should be de-referenceable or not (and how), how syntactically URIs should be constructed that refer to non-electronic objects, and – last but not least – the intuition that the uniqueness property of URIs which their name suggests is desirable, but in no way guaranteed.

This last point is the motivation of our work. Based on the availability of an open public service for supporting the global reuse of unique identifiers for individual instances called Okkam [3], which is described in Sect. 2, we are developing tool support for content creation. The global service is based on an open public repository which stores previously created identifiers for individuals, together with a simple profile. The idea is that this service can be used to look up for pre-existing identifiers of any newly created instance in a knowledge base; this process is based on an entity matching algorithm, which uses any available information about the new entity to match it with the profiles of individuals stored in the repository and thus to find candidate URIs for reuse.

The application we are going to present makes use of this service in the area of ontology editing. It aims at demonstrating the advantages of such an approach as a way to converge on common URIs for newly created semantic content. Indeed, a common practice in ontology editing is the creation of new (local) URIs for any newly created instance. Here we present a Protégé plugin, named Okkam*4P*, which supports the good practice of looking up for pre-existing URIs when editing a new RDF/OWL knowledge base. The plugin is an extension of the "individual" tab. The main difference is that, when an instance is created, the user has a chance of looking for a pre-existing URI for the corresponding individual in a publicly available service called Okkam. The match between the newly created instance and the stored individuals is based on an algorithm which compares the features of the new instance in the local knowedge base

[3] IRW in 2006 (`http://www.ibiblio.org/hhalpin/irw2006/`) and I^3 in 2007 (`http://okkam.dit.unitn.it/i3/`)

[4] see [5] for I^3 and the workshop website for IRW

with the profiles stored in OKKAM. The plugin is available and tested for the latest official release of Protégé, version 3.3.1, and the beta version 3.4; the experimental OKKAM service is accessible at `http://www.okkam.org`.

2 The Okkam*PUBLIC* Infrastructure

The work described in this paper relies on the existance of the OKKAM infrastructure, the initial idea of which was described in more detail in [4, 3]. As illustrated in Figure 1, at the heart of this infrastructure there is the central repository for entity identifiers, called OKKAM*PUBLIC*[5]. This repository can be imagined like a very large catalog, where semi-structured descriptions of entities are stored and associated to globally unique identifiers for these entities. It furthermore provides the functionality to add entities and their descriptions to the repository that have not existed there so far, and to retrieve their OKKAM identifiers for use in information systems.

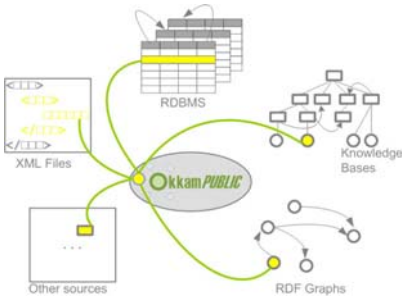


Fig. 1. Overview of the global OKKAM vision.

Figure 2 illustrates the standard use-case for the *okkamization*[6] of content, namely to query OKKAM*PUBLIC* for the existance of the entity at hand. This would usually be achieved through functionality provided by a client application – in this case Protégé – which accesses the OKKAM*PUBLIC* API, and presents (if available) a list of top candidates which match the description for the entity provided within the client application. If the entity is among these candidates, the client agent (human or software) uses the associated OKKAM identifier in the respective information object(s) *instead* of a local identifier. If the entity cannot be found, the client application can create a new entry for this entity in OKKAM and thus cause an identifier for the entity to be issued and used as described before.

[5] This service is currently under development at the University of Trento, and will be opened for public access in the near future.

[6] We call *okkamization* the process of assigning an OKKAM identifier to an entity that is being annotated in any kind of content, such as an OWL/RDF ontology, an XML file, or a database, to make the entity globally identifiable.

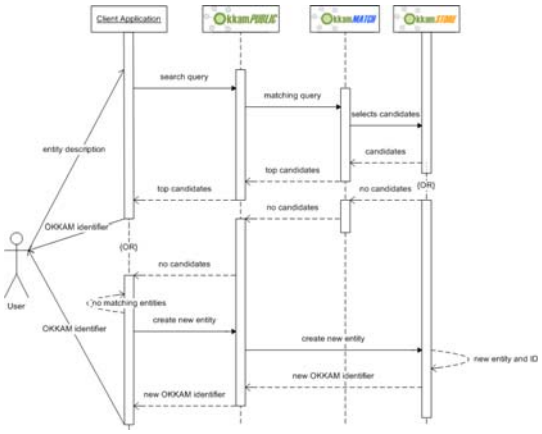


Fig. 2. Sequence diagram of the Okkam standard use case.

The large-scale, global service Okkam*PUBLIC* provides for the entity repository and a service infrastructure so that tools and applications can make use of this new technology. The current version of Okkam*PUBLIC* is a prototypical implementation of parts of a larger multi-tier architecture, namely a non-distributed version of the storage component Okkam*STORE* which in a later phase will move to a distributed layout, a preliminary version of the matching component Okkam*MATCH* which performs the search for entities, and a subset set of the developer API and toolkit Okkam*DEV* which is available[7].

The mechanisms inside Okkam which perform the matching between entity descriptions provided by the user or agent and the existing descriptions stored in the repository, display some specifics which should be mentioned at this point. One of the main characteristics of Okkam is that the description of an entity, which is necessarily used to distuingish this entity from all others in the repository, does *not* follow a fixed schema, i.e. Okkam is specifically not something like a knowledge base of entities; consequently, Okkam is not providing an ontological formalization of which attributes an entity has. The way to describe entities is extremely flexible and semi-structured, realised by way of key/value pairs which can contain arbitrary strings. The reasons for this decisions have been laid out in [4, 3], and basically go back to the point that there is an infinite variety of ways of how to model domains, for which reasons we decided to stay completely domain independent. As a consequence, the matching algorithms in Okkam*MATCH* can take as input *any* kind of description of an entity, e.g. the set of properties and values inferred from an ontology, and match it against existing data. This is how we achieve Okkam support without any dependence on, or knowledge of, an underlying schema.

[7] `http://www.okkam.org`

3 Okkam*4P* – Making Protégé an Okkam-empowered Tool

3.1 User Perspective

In our vision of a functioning Okkam infrastructure there is the notion of the so-called "Okkam-empowered tools", which are standard end-user applications (e.g. word processors, HTML/XML/OWL editors, web-based authoring environments – like blogs, forums, multimedia publishing and tagging applications, etc.) extended with functionalities which facilitate the creation of okkamized content through the use of the Okkam*PUBLIC* infrastructure. Protégé falls into this category. It is probably the most widely used editor for the creation of RDF/OWL knowledge bases (KBs), and provides vast extensibility through a plugin architecture, which makes it highly suitable for empowering it with Okkam functionality.

The plugin presented in this paper essentially assigns a global unique identifier called (the "Okkam ID") to a newly created individual, rather than relying on manual input of the user or the standard automatic mechanism of Protégé. To this end, it implements the use-case illustrated in Fig. 2: based on the data about an individual that are already provided in the KB developed by the user, it queries Okkam*PUBLIC* to see whether an identifier already exists which can be assigned to the new created individual, otherwise a new identifier would be created.

To use this plugin, the user selects an individual and right-clicks on it. A context menu will pop up, in which the item "Get Okkam ID" is the entry-point to the functions of the plugin, as illustrated in Fig. 3.

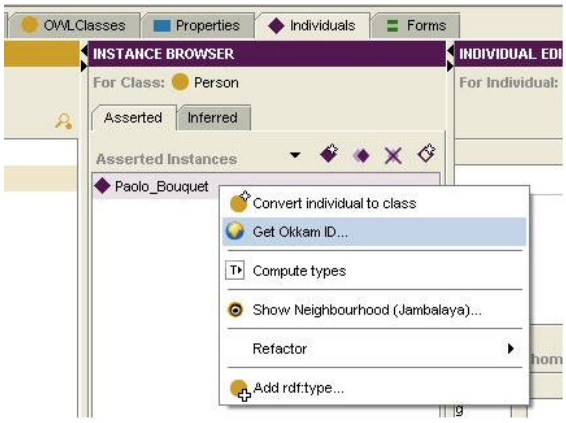


Fig. 3. Assigning a global identifier to an individual.

Once clicking on this menu, the plugin starts to collect the properties of this individual as specified in the KB, and presents a new dialog (see Fig. 4) displaying the information that is available for querying OKKAM*PUBLIC* in order to see whether an identifier for this entity already exists.

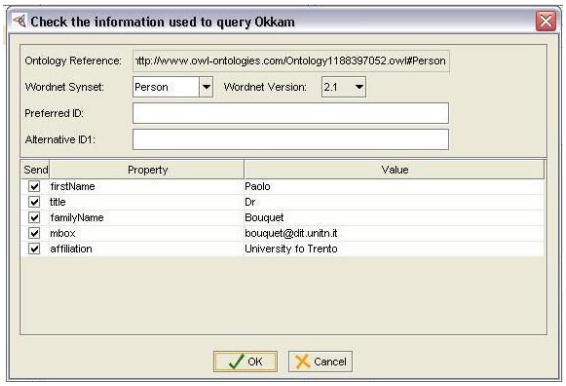


Fig. 4. The information of the chosen individual.

The properties that are gathered by the plugin to construct a query are the following:

- Ontology Reference: it is the reference of the ontology which the chosen individual belongs to. It is loaded automatically by this plugin, and it is read-only for users. If the ontology is publicly available, it can potentially be of use for the server-side matching mechanisms to improve search results for the individual.
- Wordnet Synset and Wordnet Version: provides a hint about a top-level class which the chosen individual belongs to. This has to be set by the user.
- Preferred ID and Alternative ID1: if the user wishes to use another identifier in other systems to identify the chosen individual, a user can input this identifier here. These two items are optional.
- Individual Properties: the plugin loads each property of the chosen individual automatically. The user can also deselect some properties which are thought to be unnecessary to find the OKKAM ID of the individual at hand.

After submitting this form, the plugin launches a thread to query OKKAM*PUBLIC* for matching entities by calling its web service. After searching, a list of entities that match the description for the new created individual will be visualized to the user, as illustrated in Fig. 5

The user now has the option to select one list entry as "the same" as the newly created individual and re-use the global identifier in the local KB (therefore the ID of the newly created individual will be replaced by the OKKAM ID in the

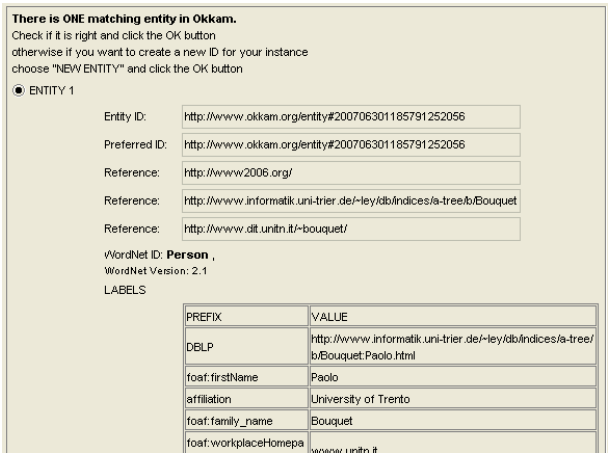


Fig. 5. Query result of with matching entities that already have an identifier in Okkam.

KB); otherwise the user can choose to create the individual as a new entity in Okkam*PUBLIC*, in which case the information selected in Fig. 4 will be inserted into Okkam repository, the new Okkam ID will be retrieved and assigned to the local individual.

3.2 Developer Perspective

The hierarchy of primary classes provided by and used in this plugin is illustrated in Fig. 6 in the appendix. In the following we describe the function of each class displayed in Fig. 6.

The class *OkkamPlugIn* is the most principal class. To extend the "Individuals" tab in protege, it needs to inherit the class *edu.stanford.smi.protegex.owl.ui.actions.ResourceAction*. This effects that the menu item "Get Okkam ID..." will appear in context-menu when the user right-clicks on a individual.

The class *okkamPanel* and *TopPanel* are used to compose the information window (see Fig. 4); the class *ResultPanel* is used to show the query result window(see Fig. 5). All of them inherit the class *javax.swing.JPanel* to present a window to users.

In this plugin, we make use of web services to interact with Okkam*PUBLIC*. The tasks of searching for matching entities and publishing a newly created entity are fulfilled by calling the webservice "EntitySearch" and "EntityPublicationWithURI" respectively. These webservices are reachable from the URL `http://okkam.dit.unitn.it:8081/OkkamCoreWebServices/services`. As complex queries can have a considerable runtime, in the initial version of Okkam*4P*, users would see nothing but a gray window until the result returned from the webservice. In the current version, we moved the plugin to a multi-threaded ar-

chitecture. Three classes which inherit class "java.lang.Thread" are new to this version.

The class *InquireThread* is used to call the webservice "EntitySearch", it is launched when the user submits the information to search for matching entities. The class *PublishThread* is used to call the webservice "EntityPublicationWithURI", it is launched when the user decides to publish a new entity to OKKAM*PUBLIC*. The class *DialogThread* is used to show a dialog during the process of searching or publishing, this dialog is meant as a user-friendly interface to inform the users that the process is running.

4 Benefits of the Approach

The vision of the OKKAM approach is the creation of what we call the *Web of Entities* (WoE): a global information space in which entities (as opposed to documents) are the main objects of discourse and thus the pivot for information access.

The pre-requisite for this WoE to function is the existence of suitable *okkamized* content, i.e. content in which identified entities (such as persons, events, locations, ...) are denoted by their globally unique OKKAM identifier, instead of a local identifier, as described in the introduction.

To achieve a substantial diffusion of okkamized content, a set of user-friendly OKKAM-empowered tools is necessary, because – as the rather slow adoption of Semantic Web technologies has shown – the mass of content creators (i.e. the users of the WWW) seem not to be extremely motivated to follow developments beyond the coding of HTML documents.

With OKKAM*4P* we are making the first and very important step towards the creation of such a suite of tools. We address the community that is "closest" to the issues addressed by the approach, and provide them with the means of creating okkamized RDF/OWL KBs. The aim is to prove that – with the systematic a-priori use of global identifiers for entities – the vision of RDF documents as a single, global, *decentralized* and *meaningful* knowledge base can in fact become reality, without having to deal with many of the difficulties of information integration, such as the ex-post alignment of entities.

5 Future Work and Conclusion

In this paper we have presented our ongoing work on OKKAM*4P*, a plugin for the creation of *okkamized* RDF/OWL knowledge bases in Protégé, and given a sketch of the underlying, globally available infrastructure OKKAM*PUBLIC*.

As regards the plugin, several improvements are scheduled in the near future. One is the general "elevation" of the tool to a more production-quality standard, including the usual aspects such as extended documentation, code improvements, etc. Secondly, as the plugin is currently implemented as an extension to the OWL part of Protégé, KBs developed in plain RDF(S) cannot benefit from

its functionality – a circumstance which we are currently investigating. Finally, additional features such as offline and batch operation, as well as automatic retrieval and assignment of OKKAM identifiers to existing KBs, are already in the design phase.

OKKAM*PUBLIC* itself will experience a great boost in the course of the European FP7 Integrated Project OKKAM, which has the aim and the means to implement the infrastructure briefly illustrated in Sect. 2 at a very large scale.

More information will be made available at `http://www.okkam.org`, the plugin itself is available from `http://www.okkam.org/projects/okkam4p/`.

References

[1] T. Berners-Lee, R. Fielding, and L. Masinter. *RFC 3986: Uniform Resource Identifier (URI): Generic Syntax*. IETF (Internet Engineering Task Force), 2005. http://www.ietf.org/rfc/rfc3986.txt.

[2] Tim Berners-Lee. Design Issues – Linked Data. Published online, May 2007. `http://www.w3.org/DesignIssues/LinkedData.html`.

[3] Paolo Bouquet, Heiko Stoermer, and Daniel Giacomuzzi. OKKAM: Enabling a Web of Entities. In *i3: Identity, Identifiers, Identification. Proceedings of the WWW2007 Workshop on Entity-Centric Approaches to Information and Knowledge Management on the Web, Banff, Canada, May 8, 2007.*, CEUR Workshop Proceedings, ISSN 1613-0073, May 2007. online http://CEUR-WS.org/Vol-249/submission_150.pdf.

[4] Paolo Bouquet, Heiko Stoermer, Michele Mancioppi, and Daniel Giacomuzzi. OkkaM: Towards a Solution to the "Identity Crisis" on the Semantic Web. In *Proceedings of SWAP 2006, the 3rd Italian Semantic Web Workshop, Pisa, Italy, December 18-20, 2006. CEUR Workshop Proceedings, ISSN 1613-0073, online http://ceur-ws.org/Vol-201/33.pdf*, December 2006.

[5] Paolo Bouquet, Heiko Stoermer, Giovanni Tummarello, and Harry Halpin, editors. *i3: Identity, Identifiers, Identification. Proceedings of the WWW2007 Workshop on Entity-Centric Approaches to Information and Knowledge Management on the Web, Banff, Canada, May 8, 2007.*, volume 249 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007. online http://CEUR-WS.org/Vol-249/.

[6] Aldo Gangemi and Valentina Presutti. Towards an OWL Ontology for Identity on the Web. In *Semantic Web Applications and Perspectives (SWAP2006)*, 2006.

[7] Aldo Gangemi and Valentina Presutti. A grounded ontology for identity and reference of web resources. In *i3: Identity, Identifiers, Identification. Proceedings of the WWW2007 Workshop on Entity-Centric Approaches to Information and Knowledge Management on the Web, Banff, Canada, May 8, 2007.*, 2007.

[8] Ian Jacobs and Norman Walsh. Architecture of the world wide web, volume one. Published online, December 2004. `http://www.w3.org/TR/webarch/`.

[9] William Kent. The Entity Join. In *Fifth Intl. Conf. on Very Large Data Bases, Rio de Janeiro, Brazil*, pages 232–238. Morgan Kaufman Publishers, 1979.

[10] William Kent. A Rigorous Model of Object Reference, Identity, and Existence. *Journal of Object-Oriented Programming*, 4(3):28–38, June 1991.

Appendix: Okkam*4P* Class Diagram

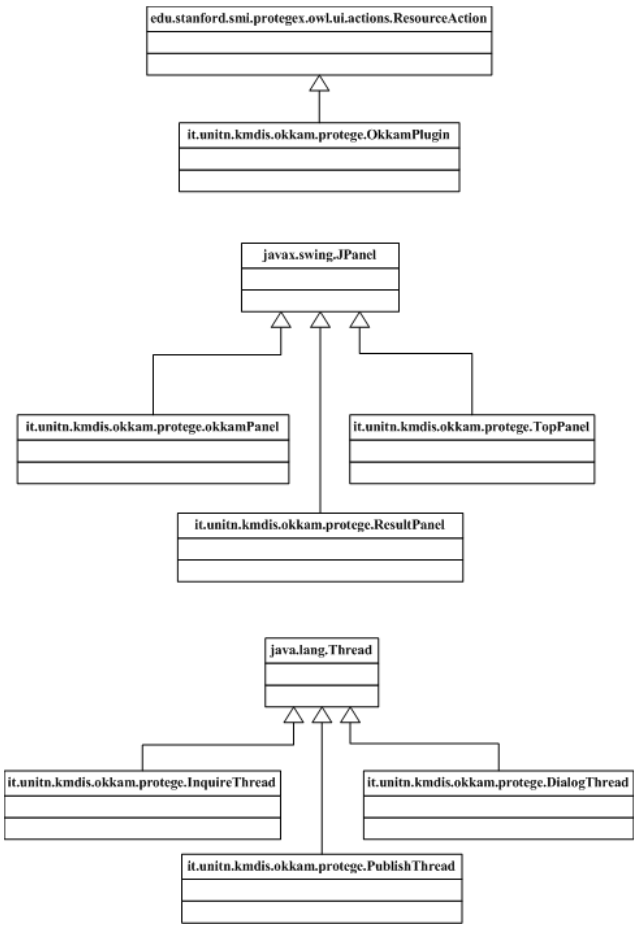


Fig. 6. UML class diagram showing the primary classes of Okkam*4P*.

Building Rules on top of Ontologies? Inductive Logic Programming can help!

Francesca A. Lisi and Floriana Esposito

Dipartimento di Informatica, Università degli Studi di Bari
Via E. Orabona 4, 70125 Bari, Italy
{lisi, esposito}@di.uniba.it

Abstract. Acquiring and maintaining Semantic Web rules is very demanding and can be automated though partially by applying Machine Learning algorithms. In this paper we show that the form of Machine Learning known under the name of Inductive Logic Programming (ILP) can help. In particular, we take a critical look at two ILP proposals based on knowledge representation frameworks that integrate Description Logics and Horn Clausal Logic and draw from them general conclusions that can be considered as guidelines for further ILP research of interest to the Semantic Web.

1 Introduction

The *logical* layer of the Semantic Web architecture is currently one of the major sources of research challenges in the field. Indeed, whereas the mark-up language OWL for *ontologies* is already undergoing the standardization process at W3C, the debate around a unified language for *rules* is still ongoing. Proposals like SWRL[1] extend OWL with constructs inspired to Horn Clausal Logic (HCL) in order to meet the primary requirement of the logical layer: 'to build rules on top of ontologies'. Since the design of OWL has been based on Description Logics (DLs) [1] (more precisely on the $\mathcal{SH}$ family of very expressive DLs [9]), rule languages for the Semantic Web will most likely follow the tradition of *old* hybrid Knowledge Representation (KR) systems such as $\mathcal{AL}$-log [7] and CARIN [11] that integrate DLs and HCL.

Acquiring and maintaining Semantic Web rules is very demanding and can be automated though partially by applying Machine Learning (ML) algorithms. The ML approach known under the name of Inductive Logic Programming (ILP) [22] seems particularly promising for the following reasons. ILP was born at the intersection of Concept Learning [19] and Logic Programming [17]. Thus it has been historically concerned with rule induction from examples within the KR framework of HCL and with the aim of prediction. The distinguishing feature of ILP, also with respect to other forms of Concept Learning, is the use of prior knowledge during the induction process. We claim that learning Semantic Web rules can be reformulated as learning rules by having ontologies as prior

[1] http://www.w3.org/Submission/SWRL/

knowledge. This may motivate an interest of the Semantic Web community in ILP. In this paper we take a critical look at the only two ILP attempts at learning rules within hybrid DL-HCL KR frameworks, the one for CARIN [28] and the other for $\mathcal{AL}$-log [12]. From the comparative analysis of the two we shall draw general conclusions that can be considered as guidelines for further ILP research of interest to the Semantic Web.

The paper is organized as follows. Section 2 provides the basic notions of DLs and HCL. Section 3 briefly describes different forms of integration of DLs and HCL. Section 4 first provides background information on the ILP methodological apparatus for non informed readers, then compares the two ILP proposals for hybrid DL-HCL formalisms. Section 5 concludes the paper with final remarks.

2 Logics behind Semantic Web Ontologies and Rules

Ontologies and rules for the Semantic Web are logically founded on Description Logics (DLs) and Horn Clausal Logic (HCL) respectively.

2.1 Description Logics

DLs are a family of decidable FOL fragments that allow for the specification of knowledge in terms of classes (*concepts*), binary relations between classes (*roles*), and instances (*individuals*) [2]. Complex concepts can be defined from atomic concepts and roles by means of constructors (see Table 1). E.g., concept descriptions in the basic DL $\mathcal{AL}$ are formed according to only the constructors of atomic negation, concept conjunction, value restriction, and limited existential restriction. The DLs $\mathcal{ALC}$ and $\mathcal{ALN}$ are members of the $\mathcal{AL}$ family. The former extends $\mathcal{AL}$ with (arbitrary) concept negation (also called complement and equivalent to having both concept union and full existential restriction), whereas the latter with number restriction. The DL $\mathcal{ALCNR}$ adds to the constructors inherited from $\mathcal{ALC}$ and $\mathcal{ALN}$ a further one: role intersection (see Table 1).

A DL knowledge base (KB) can state both is-a relations between concepts (*axioms*) and instance-of relations between individuals (resp. couples of individuals) and concepts (resp. roles) (*assertions*). Concepts and axioms form the so-called TBox whereas individuals and assertions form the so-called ABox[2]. The semantics of DLs is defined through a mapping to FOL. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for a DL KB consists of a domain $\Delta^{\mathcal{I}}$ and a mapping function $\cdot^{\mathcal{I}}$. In particular, individuals are mapped to elements of $\Delta^{\mathcal{I}}$ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (*Unique Names Assumption* (UNA) [25]). Also the KB represents many different interpretations, i.e. all its models. This is coherent with the *Open World Assumption* (OWA) that holds in FOL semantics. The main reasoning task for a DL KB is the *consistency check* that is performed by applying decision procedures based on tableau calculus.

[2] When a DL-based ontology language is adopted, an ontology is nothing else than a TBox. If the ontology is populated, it corresponds to a whole DL KB, i.e. encompassing also an ABox.

Table 1. Syntax and semantics of the DL $\mathcal{ALCNR}$.

bottom (resp. top) concept	$\bot$ (resp. $\top$)	$\emptyset$ (resp. $\Delta^{\mathcal{I}}$)
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
(abstract) role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
(abstract) individual	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
concept negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
concept intersection	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
concept union	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
value restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y\ (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
existential restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y\ (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
at least number restriction	$\geq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \lvert\{y \lvert (x,y) \in R^{\mathcal{I}}\}\rvert \geq n\}$
at most number restriction	$\leq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \lvert\{y \lvert (x,y) \in R^{\mathcal{I}}\}\rvert \leq n\}$
role intersection	$R_1 \sqcap R_2$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$
concept equivalence axiom	$C_1 \equiv C_2$	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$
concept subsumption axiom	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
concept assertion	$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$\langle a, b\rangle : R$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$

2.2 Horn Clausal Logic

The basic element in HCL is the *atom* of the form $p(t_i, \ldots, t_{k_i})$ such that each p is a predicate symbol and each t_j is a term. A *term* is either a constant or a variable or a more complex term obtained by applying a functor to simpler term. Constant, variable, functor and predicate symbols belong to mutually disjoint alphabets. A *literal* is an atom either negated or not. A *clause* is a universally quantified disjunction of literals. Usually the universal quantifiers are omitted to simplify notation. Alternative notations are a clause as set of literals and a clause as an implication. A *definite clause* is an implication of the form

$$\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_m$$

where $m \geq 0$ and α_i are atoms, i.e. a Horn clause with exactly one positive literal. The right-hand side α_0 and the left-hand side $\alpha_1, \ldots, \alpha_m$ of the implication are called *head* and *body* of the clause, respectively. Note that the body is intended to be an existentially quantified conjunctive formula $\exists \alpha_1 \wedge \ldots \wedge \alpha_m$. Furthermore definite clauses with $m > 0$ and $m = 0$ are called *rules* and *facts* respectively.

Definite clauses are at the basis of logic programming [17] and deductive databases [4]. In particular, the language DATALOG for deductive databases does not allow functors and recursion. A DATALOG *program* D is a set of DATALOG clauses. The predicates occurring in D are partitioned into two sets: the *extensional predicates* (EDB-predicates) and the *intensional predicates* (IDB-predicates). It is required that the predicate in the head of each rule in D be an IDB-predicate. Based on this distinction between extensional and intensional

predicates, a DATALOG program D can be divided into two parts, called extensional and intensional. The *extensional part*, denoted as $EDB(D)$, is the set of facts of D involving the extensional predicates, whereas the *intensional part* $IDB(D)$ is the set of all other clauses of D.

The *model-theoretic* **semantics** of DATALOG is based on the notion of Herbrand interpretation. Let D be a DATALOG program. The *Herbrand base* HB of D is the set of all atoms of the form $p(c_1, \ldots, c_k)$ such that p is a predicate of D and all the c_i are constants of D. We write EHB (resp. IHB) to denote the atoms of HB whose predicates are extensional (resp. intensional). An *Herbrand interpretation* for D is a subset of the Herbrand base HB. Let H be an Herbrand interpretation for D. A positive ground literal l is satisfied by H if $l \in H$. A negative ground literal $\neg l$ is satisfied by H if $l \notin H$. An Herbrand interpretation H for D is said to be a model of D if for every clause γ of D, for every ground instance $\gamma\prime$ of γ, at least one of the literals of $\gamma\prime$ is satisfied by H. The meaning of a DATALOG program D is the set of its models. The intersection of all the models of D is itself a model of D, and in particular is the so-called *least Herbrand model*, i.e. it is the subset of each Herbrand model of D. The corresponding proof-theoretic semantics of DATALOG is based on the *Closed World Assumption* (CWA).

Deductive **reasoning** with HCL is formalized in its proof theory. In clausal logic *resolution* comprises a single inference rule which, from any two clauses having an appropriate form, derives a new clause as their consequence. Resolution is sound: every resolvent is implied by its parents. It is also refutation complete: the empty clause is derivable by resolution from any set S of Horn clauses if S is unsatisfiable. The main reasoning task in DATALOG is query answering. A *query* Q to a DATALOG program D is a DATALOG clause of the form

$$\leftarrow \alpha_1, \ldots, \alpha_m$$

where $m > 0$, and α_i is a DATALOG atom. An *answer* to a query Q is a substitution θ for the variables of Q. An answer is correct with respect to the DATALOG program D if $D \models Q\theta$. The *answer set* to a query Q is the set of answers to Q that are correct w.r.t. D and such that $Q\theta$ is ground. In other words the answer set to a query Q is the set of all ground instances of Q which are logical consequences of D. Answers are computed by refutation.

3 Combining Ontologies and Rules

The integration of Ontologies and Rules for the Semantic Web follows the tradition of KR research on *hybrid systems*, i.e. those systems which are constituted by two or more subsystems dealing with distinct portions of a single KB by performing specific reasoning procedures [8]. The motivation for investigating and developing such systems is to improve on two basic features of KR formalisms, namely *representational adequacy* and *deductive power*, by preserving the other crucial feature, i.e. *decidability*. Indeed DLs and HCL are FOL fragments in-

comparable as for the expressiveness [2] and the semantics [26][3] but combinable under certain conditions. In particular, combining DLs with HCL can easily yield to undecidability if the interface between them is not reduced (*safeness*). A *safe* interaction between the DL and the HCL part of an hybrid KB allows also to solve the semantic mismatch between DLs and HCL [20,27].

$\mathcal{AL}$**-log** [7] is a safe hybrid KR system that integrates $\mathcal{ALC}$ [29] and DATALOG [4]. In particular, variables occurring in the body of rules may be constrained with $\mathcal{ALC}$ concept assertions to be used as 'typing constraints'. This makes rules applicable only to explicitly named objects. Reasoning for $\mathcal{AL}$-log knowledge bases is based on *constrained SLD-resolution*, i.e. an extension of SLD-resolution with a tableau calculus for $\mathcal{ALC}$ to deal with constraints. Constrained SLD-resolution is *decidable* and runs in single non-deterministic exponential time. Constrained SLD-refutation is a complete and sound method for answering *ground* queries.

A comprehensive study of the effects of combining DLs and HCL (more precisely, Horn rules) can be found in [11]. Here the family **Carin** of hybrid languages is presented. Special attention is devoted to the DL $\mathcal{ALCNR}$. The results of the study can be summarized as follows: (i) answering conjunctive queries over $\mathcal{ALCNR}$ TBoxes is decidable, (ii) query answering in a logic obtained by extending $\mathcal{ALCNR}$ with non-recursive DATALOG rules, where both concepts and roles can occur in rule bodies, is also decidable, as it can be reduced to computing a union of conjunctive query answers, (iii) if rules are recursive, query answering becomes undecidable, (iv) decidability can be regained by disallowing certain combinations of constructors in the logic, and (v) decidability can be regained by requiring rules to be *role-safe*, where at least one variable from each role literal must occur in some non-DL-atom. As in $\mathcal{AL}$-log, query answering is decided using constrained resolution and a modified version of tableau calculus. As opposite to $\mathcal{AL}$-log, the hybridization in CARIN is not safe.

4 Learning Rules on top of Ontologies with ILP

4.1 The methodological apparatus of ILP

ILP was born at the intersection between Logic Programming and Concept Learning. From Logic Programming it has borrowed the KR framework. From Concept Learning it has inherited the inferential mechanisms for induction, the most prominent of which is *generalization*. In Concept Learning generalization is traditionally viewed as search through a partially ordered space of inductive hypotheses [19]. According to this vision, an inductive hypothesis is a clausal theory and the induction of a single clause requires (i) structuring, (ii) searching and (iii) bounding the space of clauses. To serve the purposes of this paper we focus on (i) by clarifying the notion of *ordering* for clausal spaces. One such ordering is θ*-subsumption* [23]: Given two clauses C and D, we say that C θ-subsumes D

[3] Remind that the OWA holds for DLs whereas CWA is valid in HCL. Note that the OWA and CWA have a strong influence on the results of reasoning.

if there exists a substitution θ, such that $C\theta \subseteq D$. In θ-subsumption the *background knowledge* that figures prominently in ILP problem settings is left out of consideration. Yet combining the examples with what we already know often allows for the construction of a more satisfactory theory that can be glanced from the examples by themselves. Given the usefulness of background knowledge, orders have been proposed that reckon with it, e.g. Plotkin's *relative subsumption* [24] and Buntine's *generalized subsumption* [3]. Relative subsumption applies to arbitrary clauses and the background knowledge may be an arbitrary finite set of clauses. Generalized subsumption only applies to definite clauses and the background knowledge should be a definite program. Each of these orders is related to some form of deduction. It can be shown by using these two forms of deduction that generalized subsumption is a weaker quasi-order than relative subsumption. Also, it can be shown that both relative and generalized subsumption reduce to ordinary subsumption in case of non-tautologous clauses and empty background knowledge. Generalized subsumption is of major interest to this paper. It is called *semantic generality* in contrast to θ-subsumption which is a purely syntactic generality. In the general case, semantic generality is undecidable and does not introduce a lattice on a set of clauses. Because of these problems, syntactic generality is more frequently used in ILP systems. Yet for DATALOG generalized subsumption is decidable and admits a least general generalization.

Once structured, the space of hypotheses can be searched (ii) by means of refinement operators. The definition of refinement operators presupposes the investigation of the properties of the various quasi-orders. In Shapiro's sense [30], a refinement operator is a function which computes a set of specializations of a clause. Specialization is suited for the direction of search in his approach. His kind of refinement operator has been therefore called a *downward* refinement operator in ILP. Dually, operators can be also defined to compute generalizations of clauses. These can be applied in a bottom-up search, so they have been named *upward* refinement operators. A good refinement operator should satisfy certain desirable properties [32]. We shall illustrate these properties for the case of downward refinement operators but analogous conditions are actually required to hold for upward refinement operators as well. Ideally, a downward refinement operator should compute only a *finite* set of specializations of each clause - otherwise it will be of limited use in practice. This condition is called *local finiteness*. Furthermore, it should be *complete*: every specialization should be reachable by a finite number of applications of the operator. Finally, it is better only to compute *proper* specializations of a clause, for otherwise repeated application of the operator might get stuck in a sequence of equivalent clauses, without ever achieving any real specialization. Operators that satisfy all these conditions simultaneously are called *ideal*. It has been shown that ideal upward and downward refinement operators do not exist for both full and Horn clausal languages ordered by either subsumption or the stronger orders (e.g. implication). In order to define a refinement operator for full clausal languages, it is necessary to drop one of the three properties of idealness. Locally finiteness and completeness are usually considered the most important properties. This means

that locally finite and complete, but improper refinement operators can be defined for full clausal languages. On the other hand, in order to retain all the three properties of idealness, it seems that the only possibility is to restrict the search space. Hence, the definition of refinement operators is usually coupled with the specification of a declarative bias for bounding the space of clauses (iii). *Bias* concerns anything which constrains the search for theories [31]. Following [21] we will distinguish three kinds of bias: (a) *Language bias* that specifies constraints on the clauses in the search space; (b) *Search bias* that has to do with the way an ILP system searches its space of permitted clauses; (c) *Validation bias* that concerns the stopping criterion of the ILP system.

Induction with ILP generalizes from individual instances/observations in the presence of background knowledge, finding *valid hypotheses*. Validity depend on the underlying *setting*. At present, there exist several formalizations of induction in clausal logic. Two orthogonal dimensions are usually taken into account when classifying these formalizations [6]: the *scope of induction* (discrimination versus characterization) and the *representation of observations* (ground definite clauses versus ground unit clauses). *Discriminant induction* (also called *predictive induction*) aims at inducing hypotheses with discriminant power as required in tasks such as classification. In classification, observations encompass both positive and negative examples. *Characteristic induction* (also called *descriptive induction*) is more suitable for finding regularities in a given set of unclassified examples. This corresponds to learning from positive examples only. For a thorough discussion of differences between discriminant and characteristic induction see [18]. The second dimension affects the notion of *coverage*, i.e. the condition under which a hypothesis explains an observation. In *learning from entailment* (also called *learning from implications*), hypotheses are clausal theories, observations are ground definite clauses, and a hypothesis covers an observation if the hypothesis logically entails the observation. In *learning from interpretations*, hypotheses are clausal theories, observations are Herbrand interpretations (ground unit clauses) and a hypothesis covers an observation if the observation is a model for the hypothesis. A deeper investigation of learning from entailment and learning from interpretations can be found in [5].

4.2 ILP and DL-HCL formalisms

Learning in DL-HCL hybrid languages has very recently attracted some attention in the ILP community. Two ILP frameworks have been proposed that adopt a hybrid representation for both hypotheses and background knowledge.

In [28], the chosen language is CARIN-$\mathcal{ALN}$. The framework focuses on discriminant induction and adopts the ILP setting of learning from interpretations. The target concept is a unary DATALOG predicate, therefore hypotheses are represented as CARIN-$\mathcal{ALN}$ rules with a DATALOG literal in the head. The coverage relation of hypotheses against examples and the generality relation between two hypotheses are based on the existential entailment algorithm of CARIN. In particular, the generality relation is defined as an extension of Buntine's generalized subsumption [3]. Following [28], Kietz studies the learnability of CARIN-$\mathcal{ALN}$,

thus providing a pre-processing method which enables ILP systems to learn CARIN-$\mathcal{ALN}$ rules [10].

In [12], the representation and reasoning means come from $\mathcal{AL}$-log. Hypotheses are represented as constrained DATALOG clauses that are linked, connected (or range-restricted), and compliant with the bias of Object Identity (OI)[4]. Note that this framework is general, meaning that it is valid whatever the scope of induction (description/prediction) is. Therefore the literal in the head of hypotheses represents a concept to be either discriminated from others (*discriminant induction*) or characterized (*characteristic induction*). The generality relation for one such hypothesis language is an adaptation of generalized subsumption [3], named $\mathcal{B}$-subsumption, to the $\mathcal{AL}$-log KR framework. It gives raise to a quasi-order and can be checked with a decidable procedure based on constrained SLD-resolution [14]. Coverage relations for both ILP settings of learning from interpretations and learning from entailment have been defined on the basis of query answering in $\mathcal{AL}$-log [13]. As opposite to [28], the framework has been implemented in an ILP system [16]. More precisely, an instantiation of it for the case of *characteristic induction from interpretations* has been considered. Indeed, the system supports a variant of a very popular data mining task - frequent pattern discovery - where rich prior conceptual knowledge is taken into account during the discovery process in order to find patterns at multiple levels of description granularity. The search through the space of patterns represented as unary conjunctive queries in $\mathcal{AL}$-log and organized according to $\mathcal{B}$-subsumption is performed by applying an ideal downward refinement operator [15].

5 Final remarks

Building rules on top of ontologies for the Semantic Web poses several challenges not only to KR researchers investigating suitable hybrid DL-HCL formalisms but also to the ML community which has been historically interested in application areas where the Knowledge Acquisition bottleneck is particularly severe. In this paper, we have provided a brief survey of ILP literature dealing with hybrid DL-HCL formalisms. From the comparative analysis of [28] and [12], a common feature emerges. Both proposals resort to Buntine's generalized subsumption, being it a *semantic* generality relation. Note that in both the extension of [3] to hybrid DL-HCL formalisms is not trivial. Following the guidelines of [28] and [12], new ILP frameworks can be designed to deal with more expressive hybrid DL-HCL languages. The augmented expressive power may be due to a more expressive DL (than $\mathcal{ALC}$ and $\mathcal{ALN}$), or a more expressive HCL fragment (than DATALOG), or a looser integration between the DL and the HCL parts. We would like to emphasize that the *safeness* and the *decidability* of these formalisms are two desirable properties which are particularly appreciated both in ILP and in the Semantic Web application area.

[4] The OI bias can be considered as an extension of the UNA from the semantic level to the syntactic one of $\mathcal{AL}$-log. It can be the starting point for the definition of either an equational theory or a quasi-order for constrained DATALOG clauses.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
2. A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.
3. W. Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36(2):149–176, 1988.
4. S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.
5. L. De Raedt. Logical Settings for Concept-Learning. *Artificial Intelligence*, 95(1):187–201, 1997.
6. L. De Raedt and L. Dehaspe. Clausal Discovery. *Machine Learning*, 26(2–3):99–146, 1997.
7. F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. $\mathcal{AL}$-log: Integrating Datalog and Description Logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.
8. A.M. Frisch and A.G. Cohn. Thoughts and afterthoughts on the 1988 workshop on principles of hybrid reasoning. *AI Magazine*, 11(5):84–87, 1991.
9. I. Horrocks, P.F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
10. J.-U. Kietz. Learnability of description logic programs. In S. Matwin and C. Sammut, editors, *Inductive Logic Programming*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 117–132. Springer, 2003.
11. A.Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104:165–209, 1998.
12. F.A. Lisi. Principles of Inductive Reasoning on the Semantic Web: A Framework for Learning in $\mathcal{AL}$-log. In F. Fages and S. Soliman, editors, *Principles and Practice of Semantic Web Reasoning*, volume 3703 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2005.
13. F.A. Lisi and F. Esposito. Efficient Evaluation of Candidate Hypotheses in $\mathcal{AL}$-log. In R. Camacho, R. King, and A. Srinivasan, editors, *Inductive Logic Programming*, volume 3194 of *Lecture Notes in Artificial Intelligence*, pages 216–233. Springer, 2004.
14. F.A. Lisi and D. Malerba. Bridging the Gap between Horn Clausal Logic and Description Logics in Inductive Learning. In A. Cappelli and F. Turini, editors, *AI*IA 2003: Advances in Artificial Intelligence*, volume 2829 of *Lecture Notes in Artificial Intelligence*, pages 49–60. Springer, 2003.
15. F.A. Lisi and D. Malerba. Ideal Refinement of Descriptions in $\mathcal{AL}$-log. In T. Horvath and A. Yamamoto, editors, *Inductive Logic Programming*, volume 2835 of *Lecture Notes in Artificial Intelligence*, pages 215–232. Springer, 2003.
16. F.A. Lisi and D. Malerba. Inducing Multi-Level Association Rules from Multiple Relations. *Machine Learning*, 55:175–210, 2004.
17. J.W. Lloyd. *Foundations of Logic Programming*. Springer, 2nd edition, 1987.
18. R.S. Michalski. A theory and methodology of inductive learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume I. Morgan Kaufmann, San Mateo, CA, 1983.
19. T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

20. B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In S.A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *The Semantic Web*, volume 3298 of *Lecture Notes in Computer Science*, pages 549–563. Springer, 2004.
21. C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. Declarative bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 82–103. IOS Press, 1996.
22. S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence*. Springer, 1997.
23. G.D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.
24. G.D. Plotkin. A further note on inductive generalization. *Machine Intelligence*, 6:101–121, 1971.
25. R. Reiter. Equality and domain closure in first order databases. *Journal of ACM*, 27:235–249, 1980.
26. R. Rosati. On the decidability and complexity of integrating ontologies and rules. *Journal of Web Semantics*, 3(1), 2005.
27. R. Rosati. Semantic and computational advantages of the safe integration of ontologies and rules. In F. Fages and S. Soliman, editors, *Principles and Practice of Semantic Web Reasoning*, volume 3703 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2005.
28. C. Rouveirol and V. Ventos. Towards Learning in CARIN-$\mathcal{ALN}$. In J. Cussens and A. Frisch, editors, *Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 191–208. Springer, 2000.
29. M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
30. E.Y. Shapiro. Inductive inference of theories from facts. Technical Report 624, Dept. of Computer Science, Yale University, 1981.
31. P.E. Utgoff and T.M. Mitchell. Acquisition of appropriate bias for inductive concept learning. In *Proceedings of the 2nd National Conference on Artificial Intelligence*, pages 414–418, Los Altos, CA, 1982. Morgan Kaufmann.
32. P.R.J. van der Laag. *An Analysis of Refinement Operators in Inductive Logic Programming*. Ph.D. Thesis, Erasmus University, Rotterdam, The Netherlands, 1995.

Foaf-O-Matic - Solving the Identity Problem in the FOAF Network

Stefano Bortoli[1], Heiko Stoermer[1], Paolo Bouquet[1], and Holger Wache[2]

[1] University of Trento,
Dept. of Information and Communication Tech.,
Trento, Italy
{bortoli, bouquet, stoermer}@dit.unitn.it
[2] University of Applied Sciences Northwestern Switzerland
School of Business
Brugg, CH
holger.wache@fhnw.ch

Abstract. An issue that is equally arising both from social networks and the Semantic Web is the fact that, without the consistent use of the same identifier for an object across systems, it is unnecessarily hard to perform information integration. We are addressing this issue where both fields intersect, namely in the context of FOAF profiles, which describe the social network of a person with Semantic Web technology. We have developed a new, state-of-the-art web application for the generation of FOAF profiles, which has the integral characteristic of making use of the Okkam Infrastructure. This infrastructure enables the systematic, global use of unique identifiers for entities, and thus aims to solve the identity problem that exists in FOAF today.

1 Introduction

In the past few years, big efforts have been performed in the context of Semantic Web aiming at developing tools and solutions focused on mapping among ontologies and taxonomies, distributed reasoning, automatic knowledge extraction, building ontologies starting from databases and so on. In some sense, the Semantic Web relies on a silent underlying assumption: users throughout the Web will realize their sets of RDF assertions and then share them with others. Each set of RDF assertions represent a graph, where the nodes represent the resources and the edges correspond to properties. The integration between different graphs is obtained by merging the graphs by the means of collapsing the nodes describing the same resource. This process results in a graph in which the information regarding a resource is in some sense the sum of the information contained within the initial graphs. The integration among different graphs relies on the possibility to identify nodes describing the same resource.

When defining RDF assertions, the described resources are identified by Unique Resource Identifiers (URI). Nodes with the same URI in different graphs refers to the same resource, and so they may be collapsed when merging the

graphs they belong to. However, it is unclear how different developers working on their own ontologies and describing the same resource may be able to consistently use the same URIs for them. This is, in our opinion one of the biggest issues affecting the Semantic Web: the lack of "reusing and sharing" already defined identifiers for entities. Indeed, the merging of graphs relies on collapsing identifiers referring to identical resources, or, equivalently, assertions of identity binding different identifiers describing the same resource[3]. In this way, we assist on a proliferation in the creation of identifiers describing the same resource. We call this problem "*Pirandello's identity problem*", a more precise description of which is presented in section 2, and whose actual relevance in the area of social networks can be inferred from [5].

Under these premises, in this paper we try to prove the benefit arising from the use of Okkam [2] as means for sharing and reusing identifiers. In order to perform this experiment we chose to deal with one of the most successful applications of the Semantic Web, namely Friend-Of-a-Friend (FOAF). The experiment lead to the development of a new application integrating Okkam and providing functionalities for the creation of FOAF RDF profiles.

2 FOAF and Pirandello's Problem of Identity

Luigi Pirandello (1867-1936) was an Italian dramatist, novelist, and short story writer awarded the Nobel Prize in Literature in 1934. He is author of "One, No one and One Hundred Thousand" [7], a novel in which the protagonist discovers how all the persons around him have constructed in their mind a specific view of him, and how none of these views correspond to the image he has of himself. The problem experienced by the protagonist of the book of Pirandello can be used as a metaphor to explain the nature of the problem we are aiming to solve.

In a sense, every time that a new identifier is created, a new view about a resource (entity) is created. This multiplication of identifiers makes the concept of identifier itself weaker. Indeed, if an identifier for a resource is recognized by a single agent (the creator), or simply within a limited context, then we can state that this resource does not exist, or cannot be recognized for what it is, in an external context. The URI identifying a resource has, or should have, the good property of being unique, but, as long as it can be created arbitrarily, it cannot fulfil this property. Thus, creating different instances characterized by different URIs identifying the same resource leads to a parting of the associated information, and therefore a consistent decrease in the capability of making inferences and extracting knowledge. It is worthy to underline that this problem has a subtle difference from the "identity crisis" problem analyzed by Pepper [6]. Indeed, Pirandello's *identity problem*, as we mean it, describes the lack of reuse of identifiers referring to the same resource, and the weak identifying property of an arbitrarily created URI.

In order to show the concreteness of the described problem, we chose to analyze a limited context referring to a real world application: namely the Friend-of-

[3] This is, for example, the approach pursued by the Linked Data Initiative [1]

a-Friend (FOAF) project[4]. The FOAF initiative regards the definition of a set of specifications and tools based on the W3C's RDF language [4] that allow agents (people, organization, groups etc.) to describe themselves, their place of work, their main interests, education institutes etc. Furthermore, the set of properties associated to a FOAF agent are conceived to state some relationship involving other agents. The most important and used is the "*foaf:knows*" object property relating FOAF Person resources. In simple words, by means of this object property it is possible to state who is friend of whom and share this knowledge on the web in a machine readable way.

The vocabulary of FOAF is reasonably expressive, although still in evolution, and allows to express different types of information describing a person [5].

Analyzing the set of properties describing a FOAF *person* entity, it becomes clear that the best identifier *currently* available is the unique code obtained by encoding a person's email address in the field (*/foaf:mbox_sha1sum*). Indeed, an email address is uniquely identifying a mailbox of a person. Furthermore, often people use the same email address for long periods of time and this fact make the email address useful to identify persons along this period.

Any FOAF file describing a person represents an RDF graph. Every single graph is supposed to be merged with other graphs collapsing the nodes identifying the same person. Namely, if in two graphs somewhere the same unique code derived by the mail address is used, then both graphs contain some kind of information about the same person, therefore the graphs can be merged enlarging the network of "friendship". By means of this procedure it is possible to build a bigger graph containing all the information stated by the respective social network.

Analyzing superficially this process, everything seems to be at the right place, but going a little bit deeper some problems arise. The problems are related mainly to the weakness of the use of the email address based code as identifier. Indeed, an email address is not a good identifier for the following reasons:

- people change email address (change work/study institution, choose better provider, drop over-spammed[6] email address, etc...)
- people use more than one email address depending on the context of use (work, on-line gaming and shopping, 'night activities', family and friends relationship, etc...);
- email addresses can act as proxies for more than one person.

The facts listed above raise the following problem: different actors could use different email address to identify the same person (agents). Thus, a complete merging of all the information regarding a person is no more even possible. Despite the analyzed context is pretty simple and circumscribed, it reflects the more general Pirandello's identity problem affecting the semantic web evolution.

[4] The web page of the project is: `http://www.foaf-project.org`

[5] For more detail about the FOAF vocabulary see *http://xmlns.com/foaf/0.1/*

[6] over-spamed means that this address is a constant target of spam email

The weakness of the identifier generated on the base of the email address can be tackled and resolved by means of adding globally unique identifiers that are not dependent from the context of use. Thereby, there is the need of a tool that supports the creation of this kind of identifiers, namely Okkam.

3 The Okkam*PUBLIC* Infrastructure

The work described in this paper relies on the existance of the Okkam infrastructure, the initial idea of which was described in more detail in [3, 2]. As illustrated in Figure 1, at the heart of this infrastructure there is the central repository for entity identifiers, called Okkam*PUBLIC*[7]. This repository can be imagined like a very large catalog, where semi-structured descriptions of entities are stored and associated to globally unique identifiers for these entities. It furthermore provides the functionality to add entities and their descriptions to the repository that have not existed there so far, and to retrieve their Okkam identifiers for use in information systems.

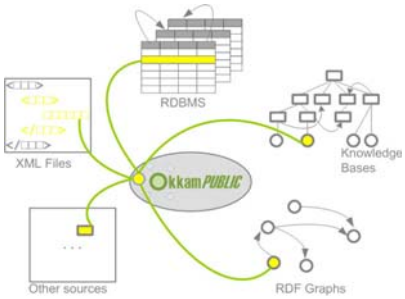


Fig. 1. Overview of the global Okkam vision.

Figure 2 illustrates the standard use-case for the *okkamization*[8] of content, namely to query Okkam*PUBLIC* for the existance of the entity at hand. This would usually be achieved through functionality provided by a client application – in this case Foaf-O-Matic – which accesses the Okkam*PUBLIC* API, and presents (if available) a list of top candidates which match the description for the entity provided within the client application. If the entity is among these candidates, the client agent (human or software) uses the associated Okkam identifier in the respective information object(s) *instead* of a local identifier. If the entity cannot be found, the client application can create a new entry for this

[7] This service is currently under development at the University of Trento, and will be opened for public access in the near future.

[8] We call *okkamization* the process of assigning an Okkam identifier to an entity that is being annotated in any kind of content, such as an OWL/RDF ontology, an XML file, or a database, to make the entity globally identifiable.

entity in Okkam and thus cause an identifier for the entity to be issued and used as described before.

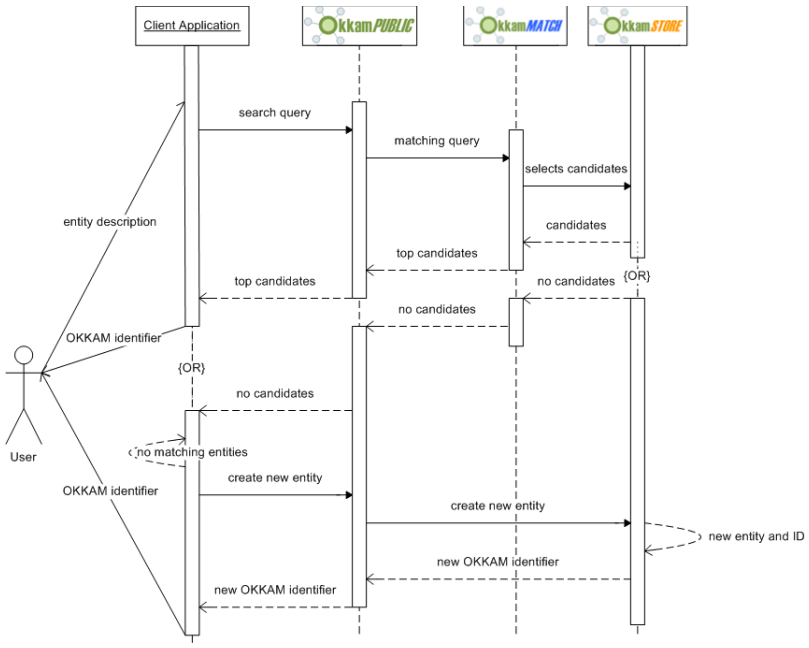


Fig. 2. Sequence diagram of the Okkam standard use case.

The large-scale, global service Okkam*PUBLIC* provides for the entity repository and a service infrastructure so that tools and applications can make use of this new technology. The current version of Okkam*PUBLIC* is a prototypical implementation of parts of a larger multi-tier architecture, namely a non-distributed version of the storage component Okkam*STORE* which in a later phase will move to a distributed layout, a preliminary version of the matching component Okkam*MATCH* which performs the search for entities, and a subset set of the developer API and toolkit Okkam*DEV* which is available[9].

The mechanisms inside Okkam which perform the matching between entity descriptions provided by the user or agent and the existing descriptions stored in the repository, display some specifics which should be mentioned at this point. One of the main characteristics of Okkam is that the description of an entity, which is necessarily used to distuingish this entity from all others in the repository, does *not* follow a fixed schema, i.e. Okkam is specifically not something like a knowledge base of entities; consequently, Okkam is not providing an ontological formalization of which attributes an entity has. The way to describe

[9] `http://www.okkam.org`

entities is extremely flexible and semi-structured, realised by way of key/value pairs which can contain arbitrary strings. The reasons for this decisions have been laid out in [3, 2], and basically go back to the point that there is an infinite variety of ways of how to model domains, for which reasons we decided to stay completely domain independent. As a consequence, the matching algorithms in OKKAM*MATCH* can take as input *any* kind of description of an entity, e.g. the set of properties and values inferred from an ontology or RDF graph, and match it against existing data. This is how we achieve OKKAM support without any dependence on, or knowledge of, an underlying schema.

4 Foaf-O-Matic – A Solution Approach

In the previous sections we have illustrated that what is missing in a FOAF Person description is a unique and sole identifier.

Thus, the problem described in Sect. 2 looks to be a perfect example of a real-world case in which OKKAM could play an important role in terms of information integration, enhancing the merging of RDF graphs describing the same person or its social network.

The approach applied for tackling the analyzed problem is to provide a tool allowing users to create/integrate FOAF person descriptions with identifiers contained in, or generated by, OKKAM. Thus, what is needed is a new application extending the functionalities provided by the foaf-a-matic application[10]. In order to underline the historical relation with the former application, this new web-based tool has been named FOAF-O-MATIC[11] (with the 'O' underlining the integration with OKKAM.)

It is important to notice that the aim of creating the FOAF-O-MATIC application is not only to replace the slightly 'obsolete' foaf-a-matic application and providing a pretty layout and new description fields. The focal point of the new application is to allow users to integrate OKKAM identifier within their FOAF document in a user-friendly way. In this way, it will be possible to merge more precisely a wider number of FOAF graphs describing a person's social networks, enhancing the integration of information and reach more easily the goal of the FOAF initiative.

A view of the new layout of the application is given in figure 3. As it is possible to see from the figure, the main layout is split in two columns: the left one for the *foaf:PrimaryPerson* description, and right one for the friend management. On the top of this two columns facilities to upload already defined FOAF files are presented. At the bottom, a "generate FOAF" button is present that trigger the generation and visualization of the FOAF file in a text area.

Without going too much into details, the FOAF-O-MATIC is meant provide the following set of functionalities:

[10] `http://www.ldodds.com/foaf/foaf-a-matic` — The foaf-a-matic is a tool that allow the definition of foaf rdf description by means of a simple form fulfillment.

[11] `http://www.okkam.org/projects/foaf-o-matic/`

Fig. 3. FOAF-O-MATIC screenshot.

- **Upload a FOAF file**. This functionality is meant to allow the upgrade of already defined FOAF descriptions and enhancing it with OKKAM identifiers. The file can be loaded providing either its Web URL, loading the file from the file system as is possible to see in the area marked with 1 in figure 3.
- **Describe the *foaf:PrimaryPerson* aka 'yourself'**. This functionality supersedes foaf-a-matic by providing of a wider choice of description fields some under testing FOAF properties. For a matter of dimension, the input for has been split in three collapsible panels presenting in the top part the standard description fields, in the middle part some extra information fields (i.e. *birthday*), and in the bottom part some *chat-id* related information fields (i.e. *yahooChatId*). A view of this part of the application is presented in figure 3 in the area marked with 2.
- **Add and describe friends**. This functionality is meant to allow users to provide a description of the friends they want to add to their social network. The information provided will be used to inquire OKKAM and retrieve a list of candidate entities corresponding to the described friends. If no entities will be found in OKKAM a newly created entity identifier will be provided. If none of the candidate entities match the user requirement in terms "recognition" a new identifier will be provided as well. "Okkamized" entities[12] will be marked in a special way. A view of this part of the application is presented in figure 3 in the area marked with 3 and 4. Notice that an OKKAM identifier is now part of the description of the described friend.
- **Select one Okkam entity for each described person**. This functionality is meant to allow the user to choose which is the entity representing the described person among the one matching such description within OKKAM, if any. The chosen entity identifier is used in the definition of the RDF FOAF file as value of *rdf:about* attribute of the described person. The list of candidate OKKAM entities is presented in a pop-up panel. The user can select the correct entity by pressing the "Select" button associated to the entity, or to state that none of the retrieved entities correspond to the describe person by pressing the button "None".
- **Retrieve the new FOAF description**. The FOAF RDF description containing the informations provided by the used is presented in a text area below the description areas. The FOAF RDF description containing the information provided and integrating an OKKAM identifier where chosen, is generated every time the "generate FOAF" button is pressed. The content of the file reflect the present state of the description provided by the user.

4.1 Foaf-O-Matic Development Framework

The framework used for the development of FOAF-O-MATIC is ICEFaces[13] open source project. ICEfaces is the most widely distributed enterprise Ajax[14] framework on the market today, providing a rich library of Ajax components. The

[12] entities which has been assigned an OKKAM identifier

[13] `http://www.icefaces.org/`

[14] Asynchronous JavaScript and XML - `http://www.ajaxprojects.com/`

main benefit of Ajax is that it gets rid of the usual submit/reload mechanism of Web forms and enables the creation of very user-friendly interfaces comparable to modern desktop windowing systems.

The primary goal behind the ICEfaces architecture is to provide a familiar Java Enterprise development model, and completely isolate them from the complexities of low-level Ajax development in JavaScript. The key to the ICEfaces architecture is a server-centric application model, where all application logic is developed in pure Java, and executes in a standard Java Application Server runtime environment.

The ICEfaces Framework is an extention to the standard JSF[15] framework, with the key difference in ICEfaces relating to the rendering phase. In standard JSF, the render phase produces new markup for the current application state, and delivers that to the browser, where a full page refresh occurs. With the ICEfaces framework, rendering occurs into a server-side DOM and only incremental changes to the DOM are delivered to the browser and reassembled with a lightweight Ajax Bridge.

5 Future Work and Conclusion

In this paper we have presented Foaf-O-Matic, an extended service for the creation of FOAF profiles, which relies on the Okkam infrastructure for issueing the "friends" with globally unique identifiers, and thus solving a-priory some of the issues of social network applications illustrated for example in [5].

The application we propose within the FOAF context is only one example of the potential application on top of an infrastructure – Okkam – which appeases the Pirandello's identity problem affecting the semantic web. But much more applications can also benefit from this infrastructure, in fact each application domain where some information about the same "thing" is distributed over different platforms is a candidate for Okkam improvements.

For the next steps we plan to extend Foaf-O-Matic in order to get some experience with Okkam and its matching algorithms. The benefit of the FOAF application is that there are many FOAF files distributed over the Internet which provide a good training base for the matching algorithm. With the FOAF application we want to tune Okkam's and Foaf-O-Matic's algorithms. With that experience we explore further application and improve Okkam over time and application domains. Also a scalable architecture with a fuzzy entity identification are subject of investigation.

The Okkam infrastructure itself will experience a great boost in the course of a European FP7 Integrated Project to start in early 2008 – consequently named Okkam – which has the aim and the means to implement the infrastructure briefly illustrated in Sect. 3 at a very large scale.

More information will be made available at `http://www.okkam.org`.

[15] JavaServer Faces - `http://java.sun.com/javaee/javaserverfaces/`

References

1. Tim Berners-Lee. Design Issues – Linked Data. Published online, May 2007. `http://www.w3.org/DesignIssues/LinkedData.html`.
2. Paolo Bouquet, Heiko Stoermer, and Daniel Giacomuzzi. OKKAM: Enabling a Web of Entities. In *i3: Identity, Identifiers, Identification. Proceedings of the WWW2007 Workshop on Entity-Centric Approaches to Information and Knowledge Management on the Web, Banff, Canada, May 8, 2007.*, CEUR Workshop Proceedings, ISSN 1613-0073, May 2007. online http://CEUR-WS.org/Vol-249/submission_150.pdf.
3. Paolo Bouquet, Heiko Stoermer, Michele Mancioppi, and Daniel Giacomuzzi. OkkaM: Towards a Solution to the "Identity Crisis" on the Semantic Web. In *Proceedings of SWAP 2006, the 3rd Italian Semantic Web Workshop, Pisa, Italy, December 18-20, 2006. CEUR Workshop Proceedings, ISSN 1613-0073, online http://ceur-ws.org/Vol-201/33.pdf*, December 2006.
4. Patrick Hayes. *RDF Semantics*, February 2004. http://www.w3.org/TR/rdf-mt/.
5. Tim O'Reilly. *The Social Network Operating System*, October 2007. online `http://radar.oreilly.com/archives/2007/10/social_network_operating_system.html`.
6. Steve Pepper and Sylvia Schwab. Curing the web's identity crisis: Subject indicators for rdf. Published online., December 2003. `http://www.idealliance.org/papers/dx_xml03/papers/05-01-05/05-01-05.pdf`.
7. Luigi Pirandello. *One, None and a Hundred Thousand*. E. P. Dutton & Co., Inc., New York, 1st edition, 1933. Translated from the Italian by Samual Putnam.

Semantic Content Annotation and Ontology Creation to Improve Pertinent Access to Digital Documents

Rocío Abascal-Mena[1] and Béatrice Rumpler[2]
[1] Universidad Autónoma Metropolitana - Cuajimalpa, José Vasconcelos 131, Col. San Miguel Chapultepec, Del. Miguel Hidalgo, 11850, México, D.F., México
[2] INSA de Lyon – LIRIS, 7 Avenue J. Capelle Bât 502 – Blaise Pascal, F69621 Villeurbanne cedex, France
mabascal@correo.cua.uam.mx, Beatrice.Rumpler@insa-lyon.fr

Abstract. In order to serve the needs of their current and future users' digital libraries must provide access to the relevant data. Since recent developments are still behind user needs, describing data using metadata has proven to be crucial for building digital libraries and for providing effective access to the information. This paper describes the use of concepts, extracted from the document itself, to annotate documents using them like "metadata tags". In order to suggest new relationships and new terms to seek, we have built also an ontology based on the concepts extracted from the theses. We present the process followed to add new semantic metadata into the digital theses and the methodology followed for the construction of the ontology based on the new metadata.

Keywords: metadata, knowledge markup, ontology, semantic annotation.

1 Introduction

Although there have been substantial advances in the way to structure information, users must still assess the pertinence of documents presented by the web. Generally, users need to get only parts of the pertinent documents rather than the complete documents. It is fastidious to read and evaluate several documents. For this reason, many pertinent documents are always unknown by users. Therefore, we try to propose a solution to enable a better access to pertinent documents or parts of documents in digital libraries.

Our work is situated within the context of a digital library, CITHER of INSA, Lyon. It concerns the online publishing of scientific theses, which is included in this study. As in other digital libraries, we encountered the same difficulties to find pertinent information in the CITHER system. During a search session, it is impossible to extract the pertinent contents of several theses. To evaluate the pertinence of a thesis, users must read several parts of the document. Furthermore, a document may be too long for a quick evaluation. A promising way to solve this problem is to use metadata in order to *"annotate"* and to describe, in a better way, the content of the documents. In our proposal, we have decided to extract the concepts, that best describe the theses, to use them as metadata for *"semantic tags"*. Of course, manual extraction of con-

cepts is a long time-consuming and is an expensive task. Tools for automating the extraction of concepts can overcome these limitations. Another promising way can be to use an ontology based on these concepts used like *"semanticfitags"*. In our approach, an ontology is the description of concepts and their relations. We propose the construction of an ontology from digital theses by following a certain methodology.

In our context, which is a digital library that publishes scientific theses, the introduction of new semantic information into documents has clearly for purpose to ameliorate information retrieval. In order to insert new semantic information into digital theses, we have used a tool able to extract concepts from a given document. Section 2, describes how we have chosen this tool. Afterward, we present the system developed to make annotations. Once digital theses are annotated, a search session is based on the new *"semanticfitags"*. In order to expand users request and to give to users also the possibility to chose documents that are closer to the pertinent document, we have decide to construct an ontology. The ontology is composed by the terms of a domain, which become, in our proposition, *"semanticfitags"* used to annotate theses. In addition, the ontology is composed by the identification of relations between terms. The identification of relations among concepts and the methodology followed to construct our ontology is described and illustrated in Section 3. Section 4 shows the integration, in the CITHER system, of the semantic annotations and the ontology in order to give the user the pertinent information. Afterward, we present a brief summary of related work in Section 5. Conclusions and further research are proposed at the end.

2fffMethodologyftofAnnotatefDigitalfDocumentsfi

In large document collections, such as digital libraries, it is very important to have mechanisms able to only select the information requested. The use of *keywords* to represent documents is a promising way to manipulate information in order to classify documents like pertinent or not pertinent.

Annotation is the process of adding semantic markup to documents, but determining which concepts are tied to a document is not an easy task. To address to this problem, several methods are proposed to extract concepts from a given document. In the field of extraction of concepts there are two main approaches: "*keyphrasefiassignment"* and "*keyphrasefextraction"*. By the term "*keyphrase"*, we mean a phrase composed by two or more words, which describes in a general way, the content of the document. "*Keyphrases"* can be seen like *"keyficoncepts"* which are able to classify documents into categories. *"Keyphrasefiassignment"* uses a controlled vocabulary to select concepts or phrases that best describe the document, instead "*keyphrasefextraction"* choose concepts from the document itself.

Our approach consists in taking a document as input to automatically generate a list of concepts as output. In general, this work could be called "*keyphrasefigeneration"* or "*conceptfigeneration"*. However, the tool used in our work performs "*conceptfiextraction"* which means that the concepts extracted always appear in the body of the input document.

2.1 Concept Extraction

In order to choose one tool for the extraction of concepts able to extract the higher number of pertinent concepts, we have evaluated four tools: (1) TerminologyExtractor of Chamblon Systems Inc., (2) Xerox Terminology Suite of Xerox, (3) Nomino of Nomino Technologies and (4) Copernic Summarizer of NRC. To evaluate the output list generated by each tool, we have compared this list with one referring list which contained concepts generated manually. The measure of performance and the method followed for scoring concepts are described in [1]. The results obtained indicate that Nomino is the most interesting tool for our approach because of the high number of pertinent concepts that it can extract.

Nomino is a search engine distributed by Nomino Technologies [15]. Nomino adopts a morphosyntactic approach. The morphological analyzer makes *"steeming",* which means that the prefix and the suffix are removed to make one single word. Nomino applies empirical criteria to filter the noise associated to the extracted concepts. These criteria include frequency and category, as well as stop lists. Nomino produces two types of interactive index, which contain all the concepts that most accurately summarize the content of a given document. One of the index created is very general, however the other one contains the most interesting concepts for Nomino. This index is based on two principles: the *"gain fi to fi express"* and the *"gain fi to fi reach"*. The *"gain fi to fi express"* classifies concepts according to their location in the given document. For example, if a paragraph is only concerned by one concept then it will be classified as important. The *"gain fi to fi reach"* classifies concepts according to the frequency of apparition. So, if a word is very rare, it will be selected as important. For example, if in a given document we find *"computer fi software"* and *"developing fi computer fi software"*, the second phrase is going to be selected as important because it is more complete and describes the document better. Instead, if the frequency of *"computer fi software"* is higher then both phases will appear in the concept list.

2.2 A fi Tool fi to fi Annotate fi Documents

Since manually annotation can be time consuming and induce to error, we have developed a tool to add easily knowledge into documents by making selections from one proposed list.

To exploit concepts extracted by the remarkable index of Nomino, we have proposed a tool to *"annotate"* documents [1]. The task we consider here is to take a document as input, in XML format, and to automatically add into it the Nomino's concepts by the way of tags. Usually when the paragraph containing the concept is identified then it is surrounded by a simple tag such as "*<concept-name>"* and "*</concept-name>"* at the end. This annotation scheme is very simple and so it can be easily applied to a text also by using a XML editor. However, by using the annotation tool, users can validate concepts proposed by Nomino or even propose other concepts to be aggregate. This tool allows the management of Nomino's concepts, the indexation and the extraction of pertinent paragraphs of the document according to some

search criteria. During a search session, the system is going to be focus in XML tags in order to retrieve the paragraph(s) containing pertinent information.

New work is taking place in order to improve the annotation tool. In the next paragraph, we describe this work, which concerns the construction of an ontology able to expand requests or categorize documents.

3 Methodology to Construct the Ontology

Gruber has defined ontology like *"an explicit specification of a conceptualization. A conceptualization is defined by concepts and other entities that are presumed to exist in some area of interest and the relationships that hold among them"* [6]. An ontology in the artificial intelligence community means the construction of knowledge models [2], [6], [12], [19] which specify concepts, their attributes and inter-relationships. A knowledge model is a specification of a domain that focuses on concepts, relations and reasoning steps characterizing the phenomenon under investigation.

Our ontology is composed of two elements: the "*domain terms"* and the "*relations"* among them. The "*domain terms"* are words or groups of words that are used to characterize a specific field. The "*relations"* among these domain terms are of type associative and hierarchic. Two main approaches can be taken when building an ontology. The first one relies on a *"top-down method"*. Someone may use an existing ontology and specify or generalize it to create another one. The second way to build an ontology is by using a *"bottom-up method"*. This method consists on extracting from the appropriate documents all the elements needed to compose an ontology. We believe that this method is accurate in our case because it does not exist yet an ontology of our domain. This method relies on two main stages: the extraction of domain terms (Section 3.1.1) and the identification of relations among these domain terms (Section 3.1.2).

Various methodologies exist to guide the theoretical approach chosen, and numerous tools for building ontology are available. The problem is that these procedures have not coalesced into popular development styles or protocols, and tools have not yet matured to the degree one expects in other software practices. Examples of methodologies followed for ontology building are described in [4], [8], [10]. In general, the following steps can define the methodology for the ontology building: (1) "*ontology capture"* and (2) *"ontology coding"*. The "*ontology capture"* consists in the identification of concepts and relations. The "*ontology coding"* consists in the definition of concepts and relations in a formal language. These two steps are going to be described in the following paragraphs in order to present the construction of our ontology.

3.1 The Ontology Capture Phase

The ontology capture phase consists in designing the overall conceptual structure of the domain. This will likely involve identifying the domain's principal concrete con-

cepts (Section 3.1.1) and their properties and identifying relationships among concepts (Section 3.1.2).

3.1.1 Concepts Extraction

This section reports on our methodology used towards defining concepts to describe the content of theses. The backbone of our ontology is a hierarchy of concepts, which had been extracted of the theses themselves.

The concepts of the ontology are used to automatically categorize documents and thus to allow a thematic access to documents. The problem of retrieving concepts and their structure come from the using of tools able to retrieve candidate concepts. Like described in Section 2, we have used Nomino for concept extraction. Given a document or a group of documents, Nomino constructs a specific index, which contains phrases composed by two or more words that are supposed to define the field. These concepts are called CNU (Complex Noun Units), series of structured terms composed by nominal groups or prepositional groups [5]. We used the CNU Nomino results as a starting point to construct our ontology. The use of NLP tools (Natural Language Processing), like Nomino, often produces *"errors"* that have to be corrected by a specialist of the field. Some of these *"errors"* include phrases that are not concepts or phrases that do not really describe the document. The *"errors"* found in our work, by using Nomino, were generally about the kind of: (1) verbs frequently used (e.g. *"called"*), (2) abbreviations of names (e.g. *"j."*), (3) names of people, cities, etc., (e.g. *"John"*), and also (4) phrases that were composed like CNU concepts but that they were not interesting (*"next phase of the field development"*).

Until now, we have not talk yet about the corpus used to make the ontology. The corpus used was composed of scientific documents. Once, these documents were analyzed by Nomino, we have obtained 78 possible concepts to be included in the ontology. We have gotten concepts like: *"information research"*, *"information system"*, *"research system"*, *"remote training"*, *"abstract ontology"*, *representation of ontology"*, etc.

The next step to construct the ontology is to define the relations between the concepts. In the next paragraph, we describe the process used to find relations by using Nomino's results as input.

3.1.2 Extraction of Semantic Relations

With regard to the acquisition of semantic relationships, there exist several approaches for acquiring semantic information. Once concepts have been retrieved, by using Nomino, they must be structured. One of the best-used techniques to discover relations among terms in documents relies on the number of terms co-occurrences. This technique identifies terms that often occur together in documents.

Different techniques exist to identify relations among terms; they are based on contexts of their co-occurrences. The idea is that two similar terms do not necessarily often occur together, as described above, but occur in similar contexts, they often appear surrounded by the same words. A first method based on this principle is described in [16]. This method represents the contexts in which words occur using a variety of lexical features that are easy to identify in large corpora. These contexts are

then converted into similarity or vector spaces which can then be clustered using a variety of different algorithms. A second method relying on this idea of similarity of contexts of terms occurrences is the one described by [11]. This method combines various text-based aspects, such as lexical, syntactic and contextual similarities between terms. Lexical similarities are based on the level of sharing of word constituents. Syntactic similarities rely on expressions in which a sequence of terms appears as a single syntactic unit. Finally, contextual similarities are based on automatic discovery of relevant contexts shared among terms.

In our approach, we use a NLP tool called LIKES [17], which is able to extract relations among concepts. LIKES (Linguistic and Knowledge Engineering Station) extracts concepts by looking to those concepts that are repeated in the document. LIKES, based on statistic principles, is a computational linguistic station with certain functions able to build terminologies and ontologies. The concepts extracted by Nomino have been paired in order to find relations between them. Thus, we have paired manually all the concepts. These pairs have also been compared in the opposite way, for example for the pair *"knowledge/language"* it has been also evaluated: *"language/knowledge"*. In this way, instead of having for example 200 pairs of concepts, we are going to have 400 pairs of concepts. Identifying relations by using LIKES it is an intense work because it takes a long time to process the corpus and to visualize the possible relationships. Furthermore, sometimes the relationships found are not very pertinent.

LIKES allows the representation of relationships in order to find similar relations in other pairs of concepts. One example of phrases that contained some relationship among the pair of concept *"knowledge//language"* is the following (we have kept the same sentence structure in English as in French language):

- A core of *knowledge* **is represented by all** *languages*;
- Other *knowledge* **is represented by some** *languages*;
- *Knowledge* **is represented in all** *languages*.

In the next paragraph we present the phase of the ontology coding where we are going to explain how we use the relations, identified by LIKES, to model a formal ontology.

3.2 The Ontology Coding Phase

The ontology coding is defined as the structuring of the domain knowledge in a conceptual model [20]. In our case the concepts are extracted by using Nomino, in some formal language.

To represent concepts and their relationships we have chosen Protégé. Protégé is a knowledge-engineering tool that enables developers to create ontology and knowledge bases [7], [9], [12], [18]. In this way, having the concepts extracted by Nomino and the relationships among concepts identified by LIKES we have used Protégé to model the ontology. Some relationships among concepts were missing and so we have added some relations like *"has"* or *"kind-of"*. Thus, we have constructed a domain ontology able to represent the main concepts included in the corpus. To have a clear idea of

how the ontology is seen, we represent, in the Figure 1, some concepts and their relationships, especially for the concepts *"language"* and *"knowledge"*.

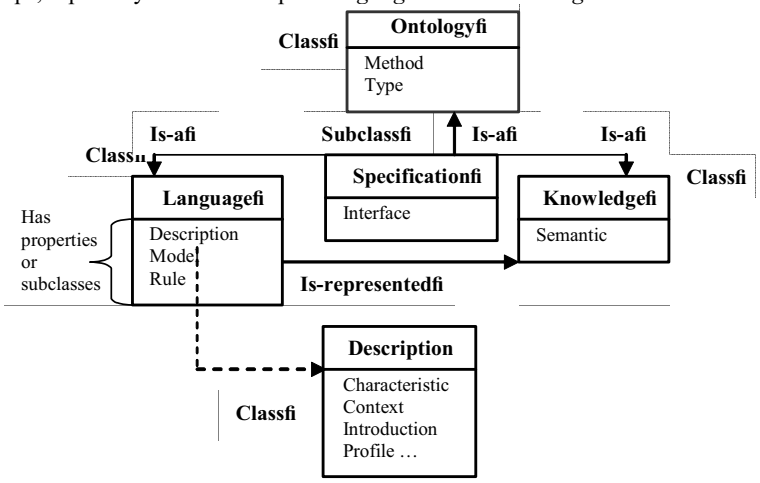


Fig.fi1. The classes or concepts *"Knowledge"* and "*Language"* are modeled in order to show their relationships among other classes or subclasses like *"Specification"fi*

The Figure 1 shows that the class *"Specification"* is a subclass of the classes *"Language"*, *"Ontology"* and *"Knowledge"*. Therefore, *"Specification"* is going to be included in these three classes. The relationship *"Is-represented"* is the one that we have found by using LIKES. Subclasses included in each class also have properties or subclasses. For example in the class *"Knowledge"* we found the subclasses *"Description"* that itself has the subclasses: *"Characteristic"*, *"Context"*, *"Introduction"*, *"Profile"*, *"Theme"*, *"Proprieties"*, etc. In our ontology, we have represented not only concepts with their relations but also "*slots"*. A *"slot"fi*is an attribute of a class in an ontology. For example, we have the relationship *"Is-represented"*, which can have some values or value types that are typically string type. Some of the values for the *"Is-represented"* relationship are: *"byfull"*, *"infisome"* and *"infull"*.

4fffiSemanticfiAnnotationfiandfiOntologyfiIntegrationfi

The initial CITHER project proposes the online access to the scientific doctoral theses of the INSA of Lyon, since January 1997. It allows the consultation, the conservation of the theses and the promotion of the research of laboratories. The distribution of theses, in PDF (Portable Document Format) format is done by the way of a server. However, by using the PDF format, it is not easy to automatically exploit the content of the theses. Therefore, we decided to use the XML format to store the theses. The new CITHER system, using XML documents, is under development. The new system's architecture was designed to satisfy the users' requirements. These re-

quirements include selecting pertinent information during a search session. We will briefly summarize the workflow and describe the associated functions.

Information Capturing. After the theses are scanned, they are annotated by including "metadata tags". These "metadata tags" come from the concepts extracted from the thesis itself. These "metadata tags" describe the semantic content. During this phase, we use NLP tool to extract concepts from the documents. The meta-information discovered during "*pre-processing"* is then stored with the corresponding documents in the repository. The storage is carried out by the "*XML content manager"*, which adds new information to the theses. Indeed, the domain knowledge contained in the "*ontology manager"* is based on the meta-information contained in the theses.

User Request. Given a search term, the ontology is used to recommend closer terms and to significantly enrich the request of the user. Users will be able to navigate between terms in order to choose pertinent documents. Once, terms are chosen by users, the "XML content manager" search in the "metadata tags" to find and retrieve pertinent fragments of the theses. By this way, if the fragments are pertinent for the user, this one can decide to retrieve the complete thesis.

5 Related Work

The terms are linguistic representation of concepts in a particular subject field [14]. Like this, applications in automatic extraction of concepts, called terms in many cases, include specialized dictionary construction, human and machine translation, indexing in books and digital libraries. Work in this area has been follow in order to produce tools for automatic extraction.

The University Michigan Digital Library (UMDL) ontology [21] delineates the process of publications using six formal concepts: *"conception"*, *"expression"*, *"manifestation"*, *"materialization"*, *"digitization"* and *"instance"*. Each of these concepts is related to other concept by using: *"has"*, *"of"*, *"kind-of"* or *"extends"* relationship. An ontology in the domain of the digital library is presented in the work of [2]. This ontology tries to represent the way in which new work is expressed. As a result, using the ontology researchers will no longer need to make claims about the contributions of documents (e.g. *"this is a new theory"*, *"this is a new model"*, *"this is a new notation"*, etc), or contest its relationships to other documents and ideas (e.g. *"it applies"*, *"it extends"*, *"it predicts"*, *"it refutes"*, etc).

Some of the methods used to specify ontologies in digital library projects include vocabularies and cataloguing codes such as Machine Readable Cataloguing (MARC). Other projects are based on the use of thesauri and classifications in order to describe different components of a document like the title, the name of the auteur, etc. In this way, some algorithms can make use of already existing thesauri in order to provide the user with useful suggestions in the integration of ontologies [3].

6ffi Conclusionfi and ff Furtherfi Researchfi

We have presented an approach to improve the document retrieval by using the semantic content. Our approach has a double advantage, first, it can exploit the entirely content of digital theses by using semantic annotations and it can provide other alternatives to the user requests. We have noticed that by adding related words (concepts words) in a document, it increases the number of relevant documents identified during a search session. In addition, ontologies can be used to support the operation and growth of a new kind of digital library, implemented as a distributed intelligent system [21]. In consequence, an ontology can be used to deduce characteristics of content being searched, and identify operations that are appropriate and available to access content or manipulate it in other ways. We have constructed an ontology by following a methodology. As long as there are not tools able to construct automatically ontologies from documents, the process carried out by using NLP tools will be fastidious and need the help of field experts. The extraction of relations by hand is very complex and by using NLP tools we have noticed that it still remains relations to be instantiated by the expert of the field. It is evident that there are still some needs in the ontology construction domain but at this moment, we are able to build ontology to support an entire domain. The construction of our ontology is only the first step to make a better access to the information in the digital library.

Further research should investigate the use of dictionaries or thesaurus in digital libraries to detect similar and not identical terms. The use of synonyms to complete our ontology could be another attempt.

Referencesfi

1. Abascal-Mena, R., Rumpler, B. Adaptive Hypertext Annotations for a Digital Library. International Transactions on Computer Science and Engineering. Vol. 32, No. 1. pp. 7-14. ISSN: 1738-6438, ISBN: 89-953729-5-8. July 2006. (2006)
2. Benn N., Buchingham S., Domingue J. Integrating Scholarly Argumentation, Texts and Community: Towards an Ontology and Services.5th Workshop on Computational Models and Natural Argument. IJCAI'05: International Joint Conference on Artificial Intelligence, Edinburgh, July 2005. (2005)
3. De Bo J., Spyns P., Meersman R. Assisting Ontology Integration with Existing Thesauri. On the Move to Meaningful Internet Systems 2004: CoopIS, DOA and ODBASE. Lecture Notes in Computer Science. pp. 801-818. ISSN: 0302-9743. (2004)
4. Fortuna B., Mladenic D., Grobelnik M. Semi-Automatic Construction of Topic Ontology. Proceedings of ECML/PKDD Workshop KDO 2005. October 2005. (2005)
5. Golebiowska, J.: SAMOVAR – Knowledge Capitalization in the Automobile Industry Aided by Ontologies. In Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000). Juan_les-Pins, France, October 2, (2000)
6. Gruber, T. R.: A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, 5, 2, pp. 199-220 (1993)

7. Hogeboom M., Lin F., Esmahi L., Yang C. Constructing Knowledge Bases for E-Learning Using Protégé 2000 and Web Services. 19th Conference on Advance Information Networking and Applications (AINA 2005). Vol. 1. pp. 215-220. ISSN: 1550-445X. (2005)
8. Kim J-M., Choi B-I, et al. A Methodology for Constructing of Philosophy Ontology Based on Philosophical Texts. Computer Standards & Interfaces. Vol. 29. Issue 3. March 2007. pp. 302-315. (2007)
9. Musen, M. A., Fergerson R. W, Grosso W. e., Noy N. F., Crubézy M., Gennari J. H.: Component-Based Support for Building Knowledge-Acquisition Systems. In Proceeding of the Conference on Intelligent Information Processing (IPP 200) of the International Federation for Information Processing World Computer Congress (WCC 2000), Beijing, (2000)
10. Nanda J., Simpson T. W., Kumara S. R. T. A Methodology for Product Family Ontology Development Using Formal Concept Analysis and Web Ontology Language. Journal of Computing and Information Science in Engineering. June 2006. Vol. 6. No. 2. pp. 103-113. (2006)
11. Nenadic G., Ananiadou S. Mining Semantically Related Terms From Biomedical Literature. ACM Transactions on Asian Language Information Processing (TALIP). Vol. 5. pp. 22-43. ISSN: 1530-0226. (2006)
12. Noy N. F., Musen M. A. Ontology Versioning in a Ontology Management Framework. IEEE Intelligent Systems. July/August 2004. Vol. 19. No. 4. pp. 6-13. ISSN: 1094-7167. (2004)
13. Noy, N. F., Fergerson, R. W., Musen, M. A.: The Knowledge Model of Protégé-2000: Combining Inter-operability and Flexibility. In Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France, October 2, (2000)
14. Pazienza M. T., Pennacchiotti M., Vindigni M., Zanzotto F. M. AI/NLP Technologies Applied to Spacecraft Mission Design. Proceedings of the 18th International conference on Innovations in Applied Artificial Intelligence. Lecture Notes in Computer Science. pp. 239-248. (2005)
15. Plante, P., Dumas, L., Plante, A.: Nomino version 4.2.22 updated the 25 July 2001. http://www.nominotechnologies.com. (2001)
16. Purandare A., Pedersen T. Discriminating Among Word Meanings By Identifying Similar Contexts. Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04). (2004)
17. Rousselot, F., Frath, P.: Terminologie et Intelligence Artificielle. In Traits d'Union, G.Kleiber and N. Le Queler, dir., Presses Universitaires de Caen. pp. 181-192, (2002)
18. Taboada M., Martínez D., Mira J. Experiences in Reusing Knowledge Sources Using Protégé and Prompt. International Journal of Human-Computer Studies. Vol. 62. pp. 597-618. (2005)
19. Vallet D., Fernández M., Castells P., Mylonas P., Avrithis Y. A Contextual Personalization Approach Based On Ontological Knowledge. International Workshop on Context and Ontologies (C&O 2006) at the 17th European Conference on Artificial Intelligence (ECAI 2006), (2006)
20. Wache H., Vogele T., Visser U., et al. Ontology-Based Integration of Information – A Survey of Existing Approaches. Proceedings of IJCAI 2001. Workshop "Ontologies and Information Sharing". (2001)
21. Weinstein, P.: Seed Ontologies: Growing Digital Libraries as Distributed, Intelligent Systems. In Proceedings of the Second International ACM Digital Library Conference, Philadelphia, PA, USA, July, (1997)

RELEVANT$^{\text{News}}$: a semantic news feed aggregator*

Sonia Bergamaschi[1], Francesco Guerra[2], Mirko Orsini[1], Claudio Sartori[3], and Maurizio Vincini[1]

[1] DII-Università di Modena e Reggio Emilia
via Vignolese 905, 41100 Modena
firstname.lastname@unimore.it
[2] DEA-Università di Modena e Reggio Emilia
v.le Berengario 51, 41100 Modena
firstname.lastname@unimore.it
[3] DEIS - Università di Bologna
v.le Risorgimento 2, 40136 Bologna
claudio.sartori@unibo.it

Abstract. In this paper we present *RELEVANT*$^{\text{News}}$, a web feed reader that automatically groups news related to the same topic published in different newspapers in different days. The tool is based on *RELEVANT*, a previously developed tool, which computes the "relevant values", i.e. a subset of the values of a string attribute. Clustering the titles of the news feeds selected by the user, it is possible identify sets of related news on the basis of syntactic and lexical similarity.
RELEVANT$^{\text{News}}$ may be used in its default configuration or in a personalized way: the user may tune some parameters in order to improve the grouping results. We tested the tool with more than 700 news published in 30 newspapers in four days and some preliminary results are discussed.

1 Introduction

Many newspapers publish their news in Internet. A recent research from the Italian Institute of statistics[4] shows that there is an increasing trend of mastheads publishing their contents on the Net often joining to the paper edition an Internet edition with special and more complete information [5]. Internet newspapers may update their contents frequently: thus there is not a daily issue but the news are continuously updated and published. As a consequence, hundreds of thousand of partially overlapping news are daily published.

The amount of information daily published is so wide that is unimaginable for a user. On the other hand, the availability of news generates new updated information needs for people. The RSS technology supports Internet users in staying updated: news

* This work was partially supported by MIUR co-funded project NeP4B (http://www.dbgroup.unimo.it/nep4b) and by the IST FP6 STREP project 2006 STASIS (http://www.dbgroup.unimo.it/stasis).

[4] http://www.istat.it

[5] Istat report about the Italian online newspapers (years 2005-2006), available at http://culturaincifre.istat.it/

are published in the form of RSS feeds that are periodically downloaded by specific applications called feed readers. In order to improve the users' selection of the interesting feeds from different newspapers, publishers group feeds in categories.

The RSS technology and the news classification in categories does not solve all the "news overload" issues. First, the categories are not fixed, and then the same topic may be called in different sites in different ways. Consequently, a user that wants to be updated about a specific topic has to manually browse the categories of potentially all the newspapers looking for interesting news. Then, the amount of news feeds daily published is so wide that automatic tools are required. If we consider the feeds published only by the five main Italian newspapers in one day, more than one thousand of news are available in their websites [6]. Such news are partially overlapping, since different newspapers publish the same information in different news. RSS feeds from different newspapers may carry the same information in different places, and therefore can confuse the reader. A great improvement might be to show groups, and leave to the reader the optional task of drilling down the group, if necessary, to compare the different flavours of the same information given by the different sources.

This work relies on *RELEVANT*[7] [1], a tool for calculating the "relevant values" among the string values of an attribute. The tool has been conceived for improving the user's knowledge of the attributes of database tables: by means of clustering techniques, *RELEVANT* provides to the user a synthetic representation of the values of the attribute. In particular, *RELEVANT* takes into account syntactic, dominance and lexical relationships for defining similarity measures among the attribute values. Such measures are then exploited for producing clusters of values, which are related. *RELEVANT* is independent of the attribute domain: a set of parameters allows the user to tune the relevance of the similarity measures and the clustering thresholds in order to produce best results.

In this paper we propose *RELEVANT*$^{\text{News}}$, a web feed reader with advanced features, since it couples the capabilities of *RELEVANT* and of a feed reader. By applying *RELEVANT* to the titles of the feeds, we can group related news published by different newspapers in different times in semantically related clusters. In particular, each cluster contains news related under the following dimensions: 1) Spatial perspective: the news with the similar titles published in different newspapers; 2) Temporal perspective: the news with the similar titles published in different times.

Several feed readers have been proposed in the literature (see section 5 for related works), but at the best of our knowledge *RELEVANT*$^{\text{News}}$ is the only lexical knowledge based feed aggregator.

The outline of the paper is the following: in order to ease the comprehension of our news aggregation technique, section 2 recalls the description of the *RELEVANT* prototype together with a detailed description of our technique for computing relevant values. Section 3 shows the *RELEVANT*$^{\text{News}}$ architecture, and in section 4 we discuss some preliminary results. Section 5 shows some related works and, finally, section 6 introduces future work.

[6] We considered the feeds on average available in the newspapers "Il Corriere della Sera", "La Repubblica", "La Gazzetta dello Sport", "Il sole 24 ore", "La Stampa" in a week of analysis.

[7] See http://www.dbgroup.unimo.it/relevant for more references.

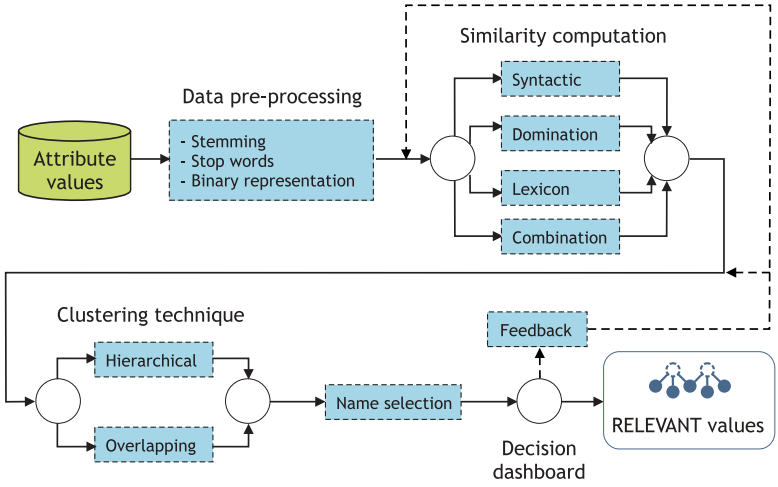


Fig. 1: The *RELEVANT* functional architecture

2 The *RELEVANT* prototype

RELEVANT is based on the idea that analyzing an attribute domain, we may find values which may be clustered because *strongly related*. Providing a name to these clusters, we may refer to a relevant value name which encompasses a set of values. More formally, given a class C and one of its attributes At, a **relevant value** for it, rv^{At} is a pair $rv^{At} = \langle rvn^{At}, values^{At} \rangle$. rvn^{At} is the name of the relevant value set, while $values^{At}$ is the set of values referring to it.

Figure 1 shows the functional flow diagram, including the following tasks:

1. **Data pre-processing**: like most cluster tasks with non-numeric attributes, the problem is to find an effective representation of the points (i.e. the attribute values) in a space, and to devise a suitable similarity function to be exploited by the clustering algorithm. After usual stemming, we build a binary representation of the attribute values, exploiting three different kinds of measures: 1) *syntactic*, mapping all the words of the attribute values in an abstract space, and defining a syntactic similarity function in such space; 2) *dominance*, expressed by the root elements described later on; and 3) *lexical*, which identifies semantically related values expressed with a different terminology.
   The syntactic similarity is based on the assumption that words related to the same object may have the same etymology and then share a common root. Thus, we group different attribute values sharing common words. However, syntactically similar values may refer to different objects. As a consequence, a similarity computation based only on a syntactical method may generate clusters containing elements with different meanings, but in conjunction with the similarities described below, it provides satisfactory results.

A similarity measure may be extracted from the Dominance relationships between the attribute values. Considering two attribute values a_1 and a_2, we say that a_1 dominates a_2 if the meaning of a_1 is more "general" than a_2. Any partial order on attribute values could be used to define dominance. We observed that it is frequent to have string domains with values composed of many words and including abbreviations and that the same word, or group of words, may be further qualified (i.e. specialized) with multiple words in many ways. Thus, we approximate the dominance between attribute values, a semantic property, with the *Contains* function, a syntactic property. *Contains* is a function based on string containment: $Contains(X,Y) = true$ iff $stem(X) \supseteq stem(Y)$, where X and Y are sets of words and *stem* is a *stemming operator*. The dominance is a partial order and can be represented by an oriented graph. Dominance is useful to build clusters of values around *root elements*. A root element is an attribute value with only outgoing edges in the domination graph, and can be taken as a representative of the cluster composed by the nodes recursively touched by its outgoing edges.

Finally WordNet is exploited for providing lexical similarity. In WordNet, English words are grouped into sets of synonyms (synsets), each one expressing a distinct concept. Synsets are described with a definition (a *gloss*) and are interlinked by means of conceptual/semantic and lexical relations. Since a word may be associated to different synsets due to the polysemy, a user is generally requested to manually select the appropriate synset for each term. On the other hand, by exploiting the WordNet lexical similarity it is possible to group different values which refer to semantically related synsets. Two different values, sharing one or more synsets are potentially similar. We can thus compute similarity on the basis of the shared synsets.

2. **Similarity Computation**: two tasks are enabled: the selection of the metrics for computing the similarity between pairs of attribute values and the selection of the similarity measures to be used (syntactic, dominance, lexical or a combination of the three). Concerning the first task, RELEVANT considers the choice of the metrics as a parameter, which can be chosen by the integration designer and changed to compare different settings. The tool implements some of the metrics commonly adopted in information retrieval (Simple Matching, Russel & Rao measure, Tanamoto Coefficient, Sorensen measure, Jaccard's Similarity [10]). Due to the sparseness of the binary matrix, the Jaccard similarity measure, which only considers the positive values in both the attribute value representations[8], is set as default. Concerning the second task, the user may balance the weight of the different similarity measures by setting specific parameters.
3. **Clustering technique**: this module implements some clustering algorithms to compute the set of relevant values on the basis of the selected similarity computation. The designer may choose between a classical clustering algorithm (generating partitions), and an overlapping clustering algorithm to compute *values*. At present we implemented a hierarchical clustering algorithm (see [4]) and an overlapping one

[8] Let us define B_{11} as the total number of times a bit is ON in both bit strings, B_{00} as the total number of times a bit is OFF in both bit strings, and L as the length of the bit string, the Jaccard Measure is defined as $B_{11}/(L - B_{00})$

based on “poles”, which are a rough partition of the domain (see [2]). Poles can be either the output of the hierarchical clustering algorithm or the set of root elements.

4. **Name selection**: The simplest way to detect a list of rvn_i candidates, i.e. the maximal values among *values*, is to use the *Contains* function. The integration designer may select the most appropriate name among them.
5. **Decision dashboard**: the integration designer may interact with *RELEVANT* in two ways: with the simple or advanced mode. The simple mode uses some default parameters and provides four sliders: the first to select the precision of the relevant values set. The slider ranges from *rough*, producing a small number of large relevant values where an attribute value may belong to different relevant values, to *accurate*, generating a large number of small relevant values each of them containing closely related attribute values. The second/third and fourth slider allow the selection of the weight of the similarity computation method. In the advanced mode, the designer may set among about hundred different configurations and clustering thresholds.
6. **Feedback**: The system provides a feedback on the results of a run that may be exploited to refine the relevant values set. A set of standard quality measures allows the tuning activity:
   - **countRV**: number of relevant values obtained for the configuration;
   - **average, max_elements, variance**: the descriptive statistics over the number of elements;
   - **count single**: number of relevant values with a single element;
   - **Rand Statistic index, Jaccard index, Folkes and Mallows index [6]**: compute the closeness of two sets of clusters evaluating couples of values that belong to the same cluster in both the sets;
   - **silhouette [9]** (only if the hierarchical clustering algorithm is used): calculates a width for each cluster based on the comparison of its tightness and separation;
   - **overlapping degree** (only if the overlapping clustering algorithm is used): number of elements which are in more than one relevant value.

   In the simple mode, the tuning activity is automatic: *RELEVANT* iteratively computes sets of relevant values to obtain a set of relevant values according to the slider indication. In the advanced mode, the designer may autonomously change the settings obtaining a set of relevant values satisfying his requirements.

3 *RELEVANT*$^{\text{News}}$ architecture

RELEVANT$^{\text{News}}$ is a web application including three components:

- A **feed aggregator** is in charge of collecting the feeds selected by the user;
- A **RSS repository**: *RELEVANT*$^{\text{News}}$ requires a database for sharing feeds published in different days by different newspapers;
- ***RELEVANT*** computes and groups similar news.

The *RELEVANT*$^{\text{News}}$ functional architecture is composed of four steps (see figure 2):

1. **selection of the news feeds**: a simple graphical user interface allows the user to select the interesting news feeds (by means of their URL) and to setup the updating policy, i.e. how frequently the feed has to be checked for new items;

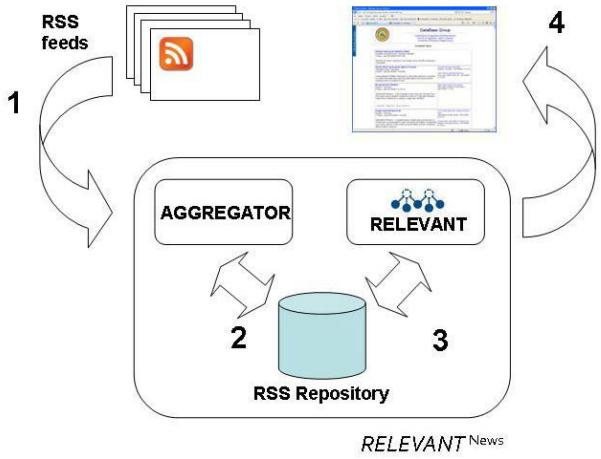


Fig. 2: The *RELEVANT*$^{\text{News}}$ functional architecture

2. **repository population**: a database supports the collection of the feeds. Thus it is possible to provide to the user news that are related to a topic, but are no longer published. The user may select a deadline for the maintenance of the news;
3. **news clustering**: by means of *RELEVANT* similar news are grouped, and for each cluster, a news, representative of the cluster, is selected. Concerning the clustering process, a simple graphical interface may allow the user to parametrize the algorithm settings, establishing the dimension of the clusters (big clusters with loosely related news, or small clusters containing strictly related information), and tuning the weight of the different similarities (lexical, dominance and syntax). Concerning the selection of the news representative of the cluster (the *relevant news*), the user may choose: a) the name extracted by *RELEVANT* ; or b) the last published news;
4. **Relevant news publication**: a web interface shows the news in terms of title, source, date and content. In case of clustered news, the relevant news is visualized together to the list of cluster related news.

In figure 3, a screen-shot of the *RELEVANT*$^{\text{News}}$ interface is shown. Each box contains a different news. In case of similarities, the relevant news text is shown in the box and the cluster related ones can be reached through a link in the bottom box.

4 Preliminary results

We tested *RELEVANT*$^{\text{News}}$ analyzing 730 news from 30 feed providers, published from the 1st to 4th of October 2007. The limited number of feeds allows us to evaluate the results by means of quality indexes provided by the tool, and of some qualitative, user-supplied evaluations. Since a gold standard for news does not exist, and different clusters for the same set of feeds may be provided by a domain expert due to the different

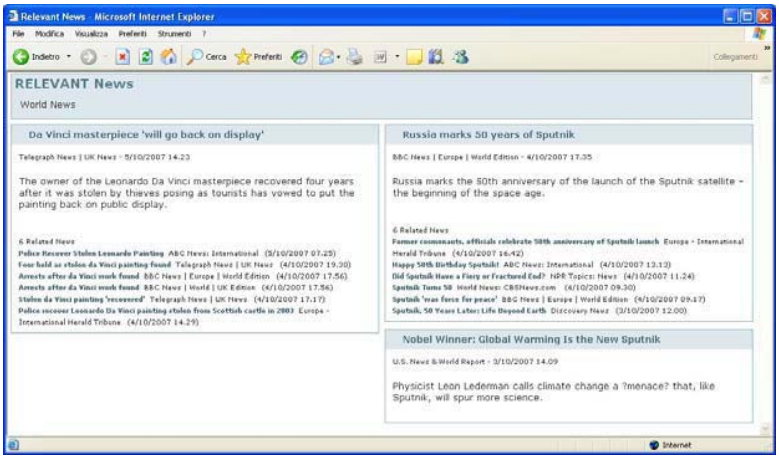


Fig. 3: *RELEVANT*$^{\text{News}}$ screen-shot

grouping criteria may be adopted, in the following, we will analyze several settings producing different clusters of news. Later on, after a brief explanation of the dataset, we will discuss some results and some numerical evaluations of 12 configurations.

4.1 Dataset analysis

The case study is defined by choosing 16 different news publishers with similar RSS feeds topic, i.e. world news, U.S. news and Europe news, analyzing altogether 30 RSS feeds. In particular, we considered 9 newspapers (6 from U.S., Chicago Tribune, New York Times, Wall Street Journal, Time, USA Today and U.S. News, and 3 European, Daily Telegraph, The Guardian and International Herald Tribune), 5 TV Network (4 from U.S., ABC News, CNN, CBS and Discovery Channel, and BBC from U.K.) and 2 International Press Agencies (Reuters and NPR).

During the period of time in analysis (4 days), each publisher provided on average 45 news, with a peak of Daily Telegraph, 141 news and ABC, 89 news, while Chicago Tribune only 5 news. Since the news topics are partially overlapping, the news are also partially overlapping: the same information may be published in different feeds at the same moment.

4.2 Evaluation of the results

Since it is not possible to compare the results produced by *RELEVANT*$^{\text{News}}$ with a gold standard, we will discuss and compare the results computed with different settings. In particular, we considered three different thresholds for the clustering algorithm (since the optimal number of clusters is not known , we considered thresholds producing respectively 300, 450 and 600 clusters) and two different tunings of the similarity parameters. Concerning these parameters, we evaluated a “lexical configuration”, where the

lexical similarity among the news titles assumes a main role, and a "syntactic configuration", where the syntactic similarity is the main similarity measure. We did not take into account a "dominance configuration" as its application in the analyzed dataset is not significant, i.e. only few title are "contained" in other titles.

	SYN_300	LEX_300	SYN_450	LEX_450	SYN_600	LEX_600
# single	241	217	388	348	527	511
Max elem	268	78	30	20	9	7
Avg elem	2.49	2.40	1.55	1.61	1.22	1.24
Variance	16.17	7.31	2.13	1.96	0.81	0.81
Silhouette	0.31	0.34	0.35	0.35	0.45	0.49

Table 1: Qualitative results

The results computed by the *RELEVANT* feedback module are summarized in table 1. The Silhouette values highlight a good clustering process in all the settings (ranges from -1, worst to +1, best). Another interesting information is provided by the "count single" value, that in all the settings is closed to the number of obtained cluster (almost 80% of the computed clusters contains only one element in all the settings). These values, which may be symptom of weakness of the tool are due to news for which no significant similarity has been found. The analysis of the dataset confirms that the observed news are related to general/generic topics from the world, and in the period of time in observation no event with a worldwide importance happened. Thus, we may suppose that clusters similar to the ones computed by $RELEVANT^{News}$ may be produced by a human reader.

In table 2, the clusters obtained considering two different configurations are analyzed. Qualitative analysis shows that lexical similarities improve the results. Table 2.a, where the clusters are computed with the syntactic configuration, shows that the news related to the recover of a stolen Leonardo da Vinci painting are grouped in two clusters. On the other hand, in Table 2.b the news are grouped in the same cluster, due to the lexical similarities among the news titles. Similar considerations may be done for news titles represented in Tables 2.c and 2.d. In this case, it is interesting to observe that the syntactic configuration produces three clusters, but the first news is correctly not included in a overall cluster (see Table 2.d) in the lexical configuration, since it refer to a different topic.

5 Related Work

There is a rich literature about metadata extraction and clustering techniques both in the area of Semantic Web, where metadata support automatic applications to understand web-site contents and in the area of Information Retrieval, where they allow document classification. For some references about this topic, see [1].

Several aggregators have been developed and implemented. Most of them are available as commercial products and their internal mechanisms are not known [9]. It is possible to group them in three different categories:

[9] See http://www.dmoz.org/Computers/Software/Internet/Clients/WWW/Feed_Readers/ for a non complete set of aggregators.

(a) News related to the "Da Vinci" stolen grouped with the syntactic configuration

#1	Arrests after da Vinci work found
	Da Vinci masterpiece "will go back on display"
	Four held as stolen da Vinci painting found
	Stolen da Vinci painting "recovered"
#2	Police recover Leonardo painting stolen from Scottish castle in 2003
	Police Recover Stolen Leonardo Painting

(b) News related to the "Da Vinci" stolen grouped with the lexical configuration

#1	Arrests after da Vinci work found
	Da Vinci masterpiece "will go back on display"
	Four held as stolen da Vinci painting found
	Stolen da Vinci painting "recovered"
	Police recover Leonardo painting stolen from Scottish castle in 2003
	Police Recover Stolen Leonardo Painting

(c) News related to the "Sputnik" grouped with the syntactic configuration

#1	Nobel Winner: Global Warming Is the New Sputnik
#2	Did Sputnik Have a Fiery or Fractured End?
	Former cosmonauts, officials celebrate 50th anniversary of Sputnik launch
	Happy 50th Birthday Sputnik!
	Sputnik "was force for peace"
#3	Russia marks 50 years of Sputnik
	Sputnik Turns 50
	Sputnik, 50 Years Later: Life Beyond Earth

(d) News related to the "Sputnik" grouped with the lexical configuration

#1	Nobel Winner: Global Warming Is the New Sputnik
#2	Did Sputnik Have a Fiery or Fractured End?
	Former cosmonauts, officials celebrate 50th anniversary of Sputnik launch
	Happy 50th Birthday Sputnik!
	Sputnik "was force for peace"
	Russia marks 50 years of Sputnik
	Sputnik Turns 50
	Sputnik, 50 Years Later: Life Beyond Earth

Table 2: A clustering example

1. **Simple readers** provide only a graphical interface for visualizing and collecting RSS feeds from different newspapers. Simple functions supporting the user in reading are provided (e.g. search engine, different ordering, association of news to a map, ...);
2. **News classifiers** show the news classified on the basis of criteria sometimes decided by the user. Simple classifications may exploit the categories and/or the keywords provided by the web sites;
3. **Advanced aggregators** provide additional features for supporting the user in reading, clustering, classifying and storing news.

There are several interesting proposals of advanced aggregators in literature. In [5], Velthune, a news search engine is proposed. The tool is based on a naive classifier that classifies the news in few categories. Unlike this approach, *RELEVANT*$^{\text{News}}$ computes clusters of similar news on the basis of their title. Classifying thousands of news in few categories produces large sets of news belonging to the same category that are not easily readable by a user. In [7] the authors propose an aggregator, called RCS (RSS Clusgator System), implementing a technique for temporal updating the contents of the clusters. NewsInEssence [8] is an advanced aggregator that computes similar news on the basis of a TF*IDF clustering algorithm, and provides to the reader a synthesis of them. Although *RELEVANT*$^{\text{News}}$ does not provide any synthesis, it implements a parametrized clustering algorithm based on syntactic/lexical/dominance relationships, that may be properly tuned for improving the creation of the clusters. Finally, the idea of

RELEVANT$^{\text{News}}$ may be compared with Google News [10] where each news is associated with a list of related information. Differently from us, Google News does not allow the user to select the newspapers. All the newspapers are analyzed and the news are provided to the user on the basis of a collaborative filtering [3].

6 Conclusion and future work

In this paper we proposed *RELEVANT*$^{\text{News}}$ a news feed reader able to group similar news by means of data mining and clustering techniques applied to the feed titles. As usual in data analysis, the startup phase requires the setting of several critical parameters. Nevertheless, for a given parameter setting, the technique calculates the relevant news without any human intervention. Moreover the parameters and similarity metrics selection determine the quality of the relevant value news. Therefore, the designer has to carefully evaluate the results and possibly change some parameters in order to improve the result quality.

Future work will be addressed on developing new techniques suitable for the feed domain. In particular, the preliminary results demonstrated that the dominance is a too restrictive condition: it is not frequent for news titles to be contained in other news titles. Moreover, some other techniques to compute the similarity may be exploited. For example, we are studying a similarity based on term frequency-inverse document frequency (TD*IDF), which takes also into account the word-spread. The idea is that unusual words and specific terms may be related to the same news.

References

1. S. Bergamaschi, F. Guerra, M. Orsini, and C. Sartori. Extracting relevant attribute values for improved search. *IEEE Internet Computing*, pages 26–35, Sep-Oct 2007.
2. G. Cleuziou, L. Martin, and C. Vrain. PoBOC: An overlapping clustering algorithm, application to rule-based classification and textual data. In *Proceedings of the 16th ECAI conference*, pages 440–444, 2004.
3. A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In Williamson et al. [11], pages 271–280.
4. B. S. Everitt. *Cluster Analysis.* Edward Arnold and Halsted Press, 1993.
5. A. Gulli. The anatomy of a news search engine. In Allan Ellis and Tatsuya Hagino, editors, *WWW (Special interest tracks and posters)*, pages 880–881. ACM, 2005.
6. M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *J. Intell. Inf. Syst.*, 17(2-3):107–145, 2001.
7. X. Li, J. Yan, Z. Deng, L. Ji, W. Fan, B. Zhang, and Z. Chen. A novel clustering-based rss aggregator. In Williamson et al. [11], pages 1309–1310.
8. D. R. Radev, J. Otterbacher, A. Winkel, and S. Blair-Goldensohn. Newsinessence: summarizing online news topics. *Commun. ACM*, 48(10):95–98, 2005.
9. P.J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20:53–65, 1987.
10. C. J. van Rijsbergen. *Information Retrieval.* Butterworth, 1979.
11. C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors. *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*. ACM, 2007.

[10] http://news.google.com/

An Approach to Decision Support in Heart Failure

Sara Colantonio [1], Massimo Martinelli [1], Davide Moroni [1], Ovidio Salvetti [1],
Francesco Perticone[2], Angela Sciacqua[2],
Domenico Conforti[4], Antonio Gualtieri[4]

[1] Inst. of Information Science and Technologies,ISTI-CNR, Pisa, Italy
{name.surname}@isti.cnr.it,
[2] University Magna Graecia, Catanzaro, Italy
perticone@unicz.it, angela.sciacqua@inwind.it,
[4] University of Calabria, Cosenza, Italy
mimmo.conforti@unical.it; gualtieri.antonio@gmail.com

Abstract. Chronic heart failure is a severe clinical syndrome among the most remarkable for prevalence and morbidity in the developed western countries. The European STREP project HEARTFAID aims at realizing an innovative platform of services which will improve the processes of diagnosis, prognosis and therapy provision in the heart failure domain. The core of the platform intelligence is a Clinical Decision Support System, designed by integrating innovative knowledge representation techniques and hybrid reasoning methods, and including advanced tools for the analysis of diagnostic data. In this paper we discuss how we are using semantic web technologies for implementing a real, significant clinical scenario, covering the clinical course of a heart failure patient.

Keywords: Decision Support Systems, Ontologies, Reasoning.

1 Introduction

Heart Failure (HF) is a complex clinical syndrome resulting from any structural or functional cardiac disorder and which impairs the ability of the ventricle to fill with or eject blood. In its chronic form, HF is one of the most remarkable health problems for prevalence and morbidity, especially in the developed western countries, with a strong impact in terms of social and economic effects [1].

The European STREP project HEARTFAID ("A knowledge based platform of services for supporting medical-clinical management of the heart failure within the elderly population") aims at defining efficient and effective health care delivery organization and management models for the "optimal" management of HF care.

The HEARTFAID platform (HFP) has been conceived as an integrated and interoperable system, able to guarantee an umbrella of services that range from the acquisition and management of raw data to the provision of effective diagnostic support to clinicians. Specifically, the core of HFP is the Clinical Decision Support

System (CDSS), which has been carefully designed by combining innovative knowledge representation formalisms, robust and reliable *reasoning* approaches, based on *learning* and *inference*, and innovative methods for diagnostic images and biomedical signals processing and analysis [2]. Semantic web technologies have been selected as the most advanced tools for knowledge formalization, re-using and sharing, for inferential reasoning, and for the integration and easy access to a number of services.

This paper aims at showing the approach based on semantic web technologies we have designed and at presenting and discussing the current results of the implementation activity, which will be finalized in 2009.

We will start by briefly reviewing the use of semantic web technologies for decision support, and then introduce the HEARTFAID approach, by describing CDSS peculiarities and main functionalities. Current results obtained on a real scenario are finally presented and discussed.

2 Clinical Decision Support and Semantic Web Technologies

The development of computerized applications for supporting health care givers is an old but still alive quest, started more than 45 years ago, in the early 1960s, and with ascending and descending periods of interest and growth. In their daily activity, health care practitioners have to continually face a wide range of challenges, ranging from making difficult diagnoses, to improve patients' quality of life, to saving money, which can benefit from effective support of computerized applications [3].

A number of systems have evolved for supporting medical decision by supplying a variety of services, from information retrieval and reporting, scheduling and communications, to cost-effectiveness, error prevention, safety, and improvement of health care quality. The most common realizations include electronic medical records [4], computerized alerts and reminders [5], clinical guidelines formalizations [6], provider order entries [7], diagnostic support [8].

The key element of all CDSS typologies is the corpus of *knowledge* meant as the necessary expertise and know-how for bringing health care to effect. Representing knowledge is then the primary task of CDSS development and concerns understanding, designing, and implementing ways of formally coding the knowledge necessary for deriving other knowledge, planning future activities, and solving problems that normally require human expertise. Suitable *languages* or *formalisms* are used for knowledge representation and result into a *Knowledge Base* (KB) of the clinical expertise. Usually, the formalism is *symbolic* and the KB contains *statements* or *expressions* of one of the following formalisms: (i) rule based; (ii) frame based; (iii) network based; and (iv) logic based [9]. Workflow based representation is also becoming well-known, especially for modeling guidelines [10], [11]. Moreover, in recent years ontologies are emerging as a powerful knowledge representation formalism which is conceptually equivalent to the frame based and to first order logic approach [12], [13].

The KB is exploited by a *reasoning engine* which processes available information for formulating new conclusions and answering questions. *Inferential* reasoning is

employed for inferring new knowledge from the KB by deduction, induction or abduction.

Semantic Web Technologies (SWT) are gathering more and more attention within the sphere of clinical decision support, thanks also to the always wider computerization of clinical systems and to the increasing availability of internet connections. More significantly, they represent instruments for viable solutions to the key problems of CDSS development, such as data integration, knowledge representation, reasoning and intelligent agents. Such significance is testified, first of all, by the rise of several ontology-like formalizations of medical domain, e.g. the Systematized Nomenclature of Medicine (SNOMED) [14], the Unified Medical Language System (UMLS) [15] or the Medical Subject Heading (MeSH) [16], to name a few. Moreover, a number of systems have been developed by using SWT, e.g. for assisting decision support in breast cancer management [17], or for modelling clinical practice guidelines [18].

The W3C has issued several recommendations about SWT, trying to establish format standards. Worthy of mention is the *Semantic Web Technology Stack* [19] which suggests a list of instruments such as the Web Ontology Language, OWL [20], for defining ontologies, as the actual de-facto standard semantic markup language for this task, and SPARQL as a language for querying ontologies [21]. Moreover, it recognizes the importance of rules for filling representation lacks of ontologies and in this context a new standard is under development, i.e. the Rule Interchange Format (RIF) [22], which will allows rules to be translated between different rule languages and thus transferred between rule systems. As an evidence of the use of SWT, the RIF working group debated a use case within the medical field.

It is important to underline that, although being powerful knowledge representation and management instruments, SWT are unsufficient to solve important problems such as handling time events and constraints, and uncertainty. So far, ad hoc strategy are usually developed when needed.

3 The HEARTFAID Case: a Semantic Web Approach to Support Decision in Heart Failure

HEARTFAID aims at deploying an effective platform of services to support HF routine practice. All the functionalities and services supplied by the entire HFP fall into three macro "contexts": (i) *biomedical data collection and management*, devoted to the acquisition and storage of continuous flows of information within healthcare structures, during patient hospitalization and outpatient visits, from analysis laboratories, and within a homecare program by *telemonitoring* patients' conditions; (ii) *knowledge-based decision support*, whose main goal is supporting, at decisional level, the HF health care operators, by making more effective and efficient all the processes related to diagnosis, prognosis, therapy and health care personalization of the HF patients; and (iii) *end-user applications*, which provide the doorway to a multitude of end-user utilities and services, such as accessing an electronic health record (EHR), querying the CDSS, applying advanced models and methods for the extraction of new knowledge, and so forth.

Our focus is on CDSS which is meant for an overall support of HF management. A careful investigation about the needs of HF practitioners and the effective benefits assured by decision support was performed: four problems have been identified as highly beneficial of HEARTFAID CDSS point-of-care intervention, i.e. diagnosis, prognosis, therapy and follow-up. An accurate analysis of the corpus of knowledge highlighted these problems mainly relied to the domain know-how and the clinical guidelines. Nevertheless, the solution of some of them seemed still debated in the medical community, due to the lack of validated and assessed evidences, e.g. prognosis. In such cases, making a decision requires an investigation on the hidden, complex, often non-linear correlations among data, together with high-level analytical processing functions. The knowledge needed for the solution should, then, be acquired directly from data (*inductive knowledge*) and stored in a model (e.g. *Artificial Neural Networks*, *Support Vectors Machines*), able to induce *sub-symbolic* knowledge from a data-driven processing [23].

HEARTFAID CDSS was then designed for incorporating different reasoning models and according to a multilevel conceptualization scheme for distinguishing among (i) the *knowledge level*, corresponding to all the information needed by the system for performing tasks, e.g. data, domain knowledge, computational decision models; (ii) the *processing level*, consisting of the system components that are responsible of tasks accomplishment by using the knowledge level; (iii) the *end-user application level*, including the system components whose functionalities are specifically defined for interacting with the user. This separation assures a high level of flexibility, since any change of the formalized knowledge will not affect the processing level.

Moreover, the knowledge level was modeled by integrating a formalization of symbolic knowledge and computational reasoning models required by those difficult HF decision problems, such as prognosis assessment and early detection of patient's *decompensation*.

In detail, the CDSS architecture consists of the following components (Fig. 1):

- *Domain Knowledge Base*, consisting of the domain knowledge, formalized from the european guidelines for the diagnosis and treatment of chronic HF and the clinicians' know-how;
- *Model Base*, containing the computational decision models, signals and images processing methods and pattern searching procedures;
- *Meta Knowledge Base,* composed by the strategy knowledge about the organization of CDSS tasks.
- *Brain*, the system component endowed with the reasoning capability;
- *Explanation Facility*, providing means to explain conclusions taken.

The Brain was modeled by functionally separating a *meta level*, devoted to task accomplishment and organization, and an *object level*, responsible of actually performing tasks, by reasoning on the computational and domain knowledge. A *Strategy Controller* was inserted for performing the meta level functionalities, by orchestrating the two components of the object level, i.e. the *Inference Engine* and the *Model Manager*.

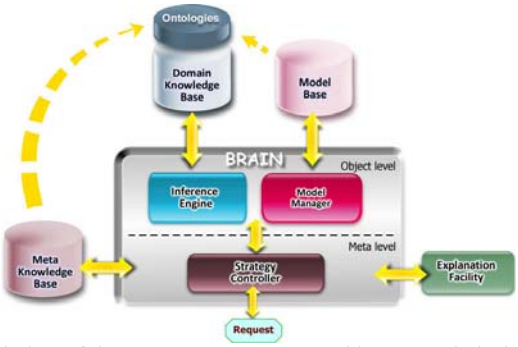


Fig. 1. The general view of the HEARTFAID CDSS architecture – dashed arrows correspond to reference to the ontologies, while the others denote a direct communication.

For modeling the Domain Knowledge Base, ontologies combined with rules were chosen as representation formalism since also assures easy re-use and sharing of knowledge. After a preliminary ontology, mainly corresponding to a structured terminology of the domain, we began to develop a new ontology by inserting relevant properties, classes and relations for a coherent and comprehensive formalization, also in accordance to standard medical ontologies, such as UMLS. To have a modular, less complicated developing, and more performing system we are developing some core and upper level ontologies. This was worthwhile and possible because in most cases the decisional support does not require the reasoner to take into account all about the patients and domain information (examples of core ontologies are *Therapy*, *MinnesotaQuestionnaire*, *Ecocardiocram* and example of upper ontologies are *DiagnosticProcedure*, *Suggestion*). A fragment of this ontology is shown in Fig. 2.

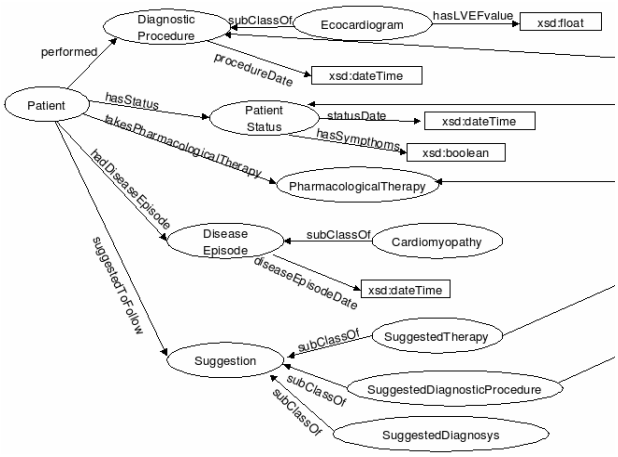


Fig. 2. Some relevant classes and properties of the ontology

Rules were used to fill the logical lacks of ontologies axioms and were elicited from the ESC guidelines and with a strong interaction with the clinicians. An example

rule for therapy suggestion, elicited in a natural language implication format: "*If a patient has Left Ventricle Ejection Fraction <= 40% and he is asymptomatic and is assuming ACE Inhibitors or ARB) and he had a myocardial infarction then a suggestion for the doctor is to give the patient Betablockers*". Of course this first set of rules was incomplete caused by logic jumps so now we are revising the entire flow of reasoning by refining rules and ontologies when not rewriting them.

An inference engine was devised for the corresponding inferential reasoning processes, by induction and deduction on the formalized knowledge for assessing patients' status, formulating diagnosis and prognosis, assisting therapy planning, and patients' monitoring. Instruments selected for development are discussed in the successive section.

HFP was devised for supplying a number of services and conceived for consciously distributing the work load among the various components. This means that, to avoid burdening the CDSS, other components were inserted for aiding the effectiveness of the support services. Their development was distributed among the partners of the project. A sketch of the platform with the components interacting with the CDSS is shown in Fig. 3. An EHR module was inserted for suitably organizing, visualizing and managing patients' data, stored into the platform Repository. In particular, a dedicated repository for storing examination images was conceived in accordance to the DICOM standard. An Agenda module was included for managing patients' care planning, e.g., scheduling new visits, prescribing new examinations and so forth. The User Interface was designed as a fundamental component, responsible for all the interactions and communications with the users.

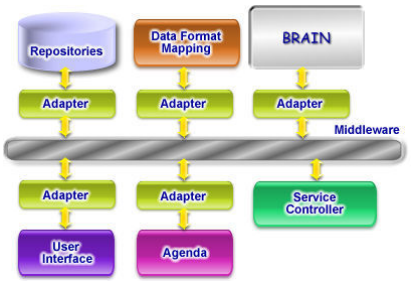


Fig. 3. A sketch of HFP with the components that interact with the CDSS.

The different components of the platform were seen as *resources*, by *virtualizing* the operations required for their management. When involved, the different components are dynamically integrated for supplying sophisticated but much flexible applications. The responsible of guaranteeing integration and interoperability among all the HFP components was defined as the platform Middleware, which includes all the adapters necessary for the virtualization. For simplifying the provision of different services, a Service Controller was comprised for managing platform events and invoking the other components. In this perspective, the CDSS component was designed as a resource able to offer a number of functionalities and to interact with the other resources for performing its tasks. Each decision-making problem has to be translated into a *request* or a *class of requests* committed to the CDSS, which is then activated *on-demand*. The system handles every request according to a specific policy

encoded in the Meta KB, and interacting, when needed, with the other platform components.

4 Summary of Results on a Real Scenario

In accordance to W3C recommendations, we selected OWL for defining the ontologies and Protégé and Swoop as editors. For defining the rules of the knowledge base, we chose SWRL [24], the Semantic Web Rule Language combining OWL and RuleML [25], which is a submission to W3C that extends the set of OWL axioms to include Horn-like rules. For realizing the reasoning component, Jena [26] was selected as a Java programming environment that uses OWL, SPARQL and includes a rule-based inference engine. To improve the reasoning capability of the latter and also to use SWRL we also used Bossam [27] and Pellet [28] depending on the SWRL builtins offered.

A use case scenario was chosen in order to simulate a real case that is partial if related to the entire problem but it let us understand better the work and the rules to be implemented. According to what stated in the previous section, we developed several core ontologies: Fig. 4 shows a fragment of the Therapy core ontology.

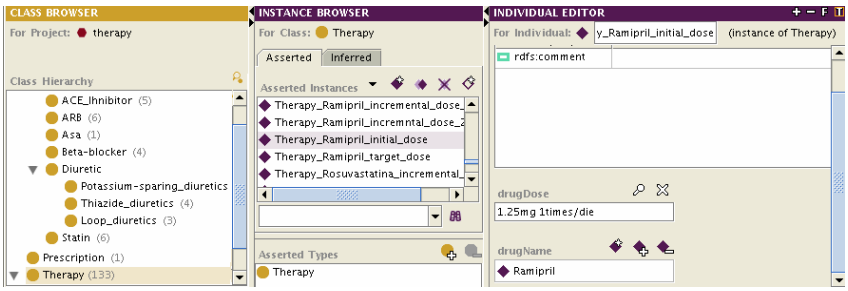


Fig. 4. A Fragment from the therapy core ontology

The set of rules elicited using a logic implication format in natural language has been simply transformed into a set of SWRL rules. An example rule is shown in Fig. 5 as it has been defined in Protégé.

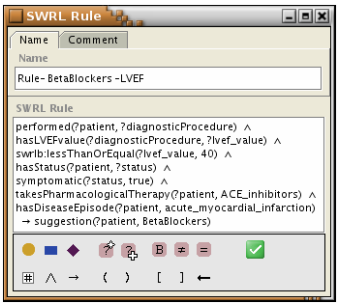


Fig. 5. A rule developed in SWRL.

Strongly typed data are stored into heterogeneous (relational and XML) and distributed databases by clinicians, patients and devices. Instead of using sets of rules codified into a programming language for transforming data into ontological format, we preferred to maintain the typed data and transform them into OWL using XML mapping files (D2R and XQuery). This allowed us to maintain independence of ontologies and rules from programming and from the inference engine choice using standards.

Currently the meta-level is only devoted to task accomplishment and sub-tasks organization, and for managing the requests from and towards the other HFP components. After all the object models completion, the strategy level will be used to foresee prognosis by merging the results coming from the inference and the other models (Support Vector Machines, Bayesian and Neural Networks).

Let us now consider the use case and how it is being developed. We consider a 65 years old patient, former smoker, suffering from hypertension from several years. The patient is enrolled in the HFP and, in particular, the telemonitoring services offered by the platform are activated. Information about the patient's status are sent to the HFP through the use of devices (blood pressure, oxygen saturation,...) and by web forms filling. For example the patient periodically answers a questionnaire through his web-based user interface and sends the information to the HFP interface that checks missing values. Then the Service Controller stores this information into the repository, gets historical data and opportunely invokes the specific CDSS service responsible of handling the request. The CDSS analyzes data and answers supplying the current patient's status, e.g. worsening of symptoms, and a set of suggested actions the clinician should undertake, e.g. schedule a new visit, change the New York Heart Association (NYHA) class, and change the therapy and so forth.

Then the Service Controller stores results into the repository and, according to the CDSS conclusions, if patient situation is worsening alert the clinician on duty, sending a sms or an email on the base of severity, for example suggesting to perform a visit.

When the doctor on duty logs in the HFP, the list of patients is displayed ordered by their severity status and the timestamp of the last related event (Fig. 6 - left).
Then he chooses to analyze the patient's situation and the change in his status is shown along with the list of suggested actions, for instance as a list of operations that can be selected. He then approves the schedule of the visit and the Service Controller forwards the request to the Agenda that opportunely records it and informs the patient. At the hospital, during the visit, the physician inserts his observations into the patient's record and decides to approve the change of the NYHA class (Fig. 6 - right): he selects the corresponding action within the list and the Service Controller takes care of registering the change in the patient's record. An ECG is then performed for further investigations. Once the information obtained by the ECG have been recorded, a request for its interpretation is sent by the Service Controller to the CDSS, which suggests performing an echocardiography as displayed in the recommended actions list. A change of therapeutic strategy is decided by the clinician supported by the CDSS (Fig. 7).

Future activities will consist in finalizing the platform implementation by concluding the realization of the Domain Knowledge Base, the algorithms contained

in the Model Base and the Brain, in particular of its meta level in order to integrate all the object models and the interface.

Acknowledgements

This research work is supported by the European Community, under the Sixth Framework Programme, Information Society Technology – ICT for Health, within the STREP project "HEARTFAID" (IST-2005-027107), 2006-2009.

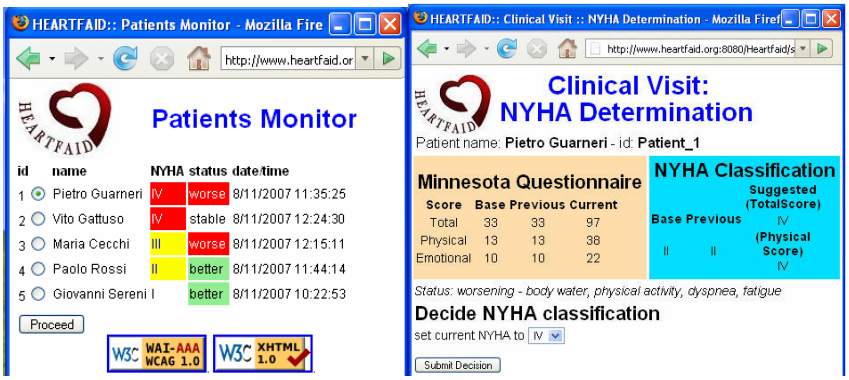


Fig. 6. On the left, the patients monitor; on the right, the NYHA determination

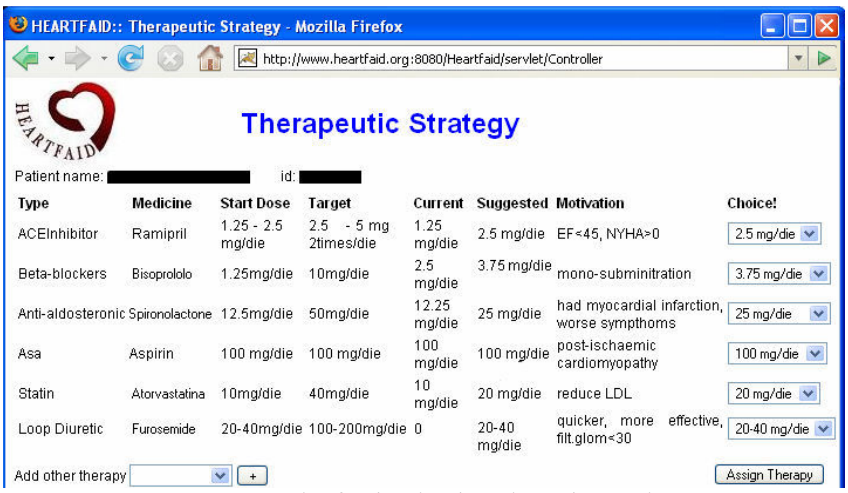


Fig. 7. CDSS suggestion for changing the patient's therapeutic strategy

References

1. ESC Swedberg, K. et al., The Task Force for the diagnosis and treatment of CHF of the European Society of Cardiology, "Guidelines for the diagnosis and treatment of Chronic Heart Failure: full text (update 2005)", *European Heart Journal* (2005) 45 pages.
2. VV.AA.. Functional Specifications of Data Processing and Decision Support Services. Deliverable D15, European STREP project HEARTFAID IST-2005-027107 (2007).
3. Greenes, D. (2007). *Clinical Decision Support: The Road Ahead.* Academic Press.
4. Poissant, L., Pereira, J., Tamblyn, R. Kawasumi, Y. The impact of electronic health records on time efficiency of physicians and nurses: a systematic review. *J Am Med Inform Assoc.* 12 (5) (2005) 505-16.
5. Johnson, P.D., Tu, S., Booth, N., Sugden, B., Purves, I.N. Using scenarios in chronic disease management guidelines for primary care. In *AMIA Symp*. (2000) 389-93.
6. GEM *E2210-02 Standard Specification for Guideline Elements Model (GEM)-Document Model for Clinical Practice Guidelines* ASTM International (2003).
7. Kaltschmidt, J., Gallin, S., Haefeli, W.E. Essential functional requirements for an effective electronic drug information system in a hospital. I*nt J Clin Pharmacol Ther.* 42 (2004) 615.
8. I. Sim, P. Gorman, R. A. Greenes, R. B. Haynes et al. Clinical Decision Support Systems for the Practice of Evidence-based Medicine. J Am Med Inform Assoc. 2001 Nov–Dec; 8(6): 527–534.
9. Helbig, H. *Knowledge Representation and the Semantics of Natural Language*, Springer, Berlin, Heidelberg, New York. (2006)
10. Ciccarese, P., Caffi, E., Baiocchi, L., Quaglini, S., Stefanelli, M. A guideline management system. *Medino* (2004) 28-32.
11. Boxala, A.A., Peleg, M., Tu, S. et al. GLIF3: a representation format for sharable computer-interpretable clinical practice guidelines. *J. Biomed Inform.*, 37(3) (2004) 147-161.
12. Stevens, R., Goble, C.A., Bechhofer, S. Ontology-based Knowledge Representation for Bioinformatics. *Briefings in Bioinformatics*, 1(4) (2000) 398–416.
13. Bayegan, E. Nytro, O. Grimsmo, A. Ontologies for knowledge representation in a computer-based patient record. *14th IEEE Int. Conf. on Tools with Artificial ICTAI* (2002).
14. www.snomed.org/ SNOMED
15. http://www.nlm.nih.gov/research/umls/ UMLS
16. http://www.nlm.nih.gov/mesh/ MeSH
17. S. Dasmahapatra, D Dupplaw, B Hu, H Lewis, P Lewis, M Poissonnier, N Shadbolt. Ontology-Based Decision Support for Multidisciplinary Management of Breast Cancer. In Proc. of International Workshop on Digital Mammography, (2004)
18. S. Hussain, S. Raza Abidi, S. S. Raza Abidi. Semantic Web Framework for Knowledge-Centric Clinical Decision Support Systems. In Artificial Intelligence in Medicine, LNCS Springer, Vol. 4594 (2007) 451-455
19. http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(1) W3C Stack
20. http://www.w3.org/TR/owl-features/ OWL Web Ontology Language
21. http://www.w3.org/TR/rdf-sparql-query/ SPARQL Query Language for RDF
22. http://www.w3.org/TR/2006/WD-rif-ucr-20060710/ RIF Use Cases and Requirements
23. Bishop, C.M., *Pattern Recognition and Machine Learning*, Springer, Berlin, Heidelberg, New York (2006).
24. http://www.w3.org/Submission/SWRL/ SWRL: A Semantic Web Rule Language
25. http://www.ruleml.org/ The Rule Markup Initiative
26. McBride, B. Jena: Implementing the RDF Model and Syntax Specification. In *WWW2001, Semantic Web Workshop* (2001).
27. http://bossam.wordpress.com/ Bossam Rule/OWL Reasoner
28. http://pellet.owldl.com/ OWL DL reasoner

Applying Semantic Web Services

Stefania Galizia, Alessio Gugliotta, Carlos Pedrinaci, John Domingue
Knowledge Media Institute, The Open University,
Walton Hall, Milton Keynes, MK7 6AA
United Kingdom
{S.Galizia, A.Gugliotta, C.Pedrinaci, J.B.Domingue}@open.ac.uk

Abstract. The use of Semantic Web Services (SWS) for increasing agility and adaptability in process execution is currently investigated in many settings. The common underlying idea is the dynamic selection, composition and mediation - on the basis of available SWS descriptions – of the most adequate Web resource (services and data) to accomplish a specific process activity. In this paper we describe IRS-III, a framework for creating and executing semantic Web services, which takes a semantic broker based approach to mediating between service requesters and service providers. We describe the overall approach of IRS-III from an ontological perspective. We then illustrate our approach through three different applications to domains of Business Process Management, e-Learning and e-Science.

Keywords: Semantic Web Services, SWS Applications, Ontologies.

1 Introduction

The continuous diffusion of Web services increases the sharing of resources – services and data – on the Web. The specific nature of Web services - self-contained and platform-independent computational elements – gives them high availability and facilitates their reusability and interoperability across several application domains. One of the advantages of Web service technology is indeed the fairly simple aggregation of complex services out of repositories of simpler or even atomic services. However, Web service standards – [10], [12] and [14] - do not completely describe the capability of a service and cannot be understood by software programs. A human developer is thus required to interpret the meaning of inputs, outputs and applicable constraints, as well as the context in which services can be used. Therefore, the automatic discovery and selection at runtime of the most adequate resource for a given activity is limited, as well as the automatic solution of possible mismatches at the level of data format, message protocol and underlying organizational processes.

Semantic Web Services (SWS) research aims to automate the development of Web service based applications through the semantic Web technologies. By providing formal descriptions with well defined semantics, SWS facilitate the machine interpretation of Web service – functional and not functional - properties. The research agenda for SWS identifies a number of key areas of concern, namely:

- *Discovery:* finding the Web service which can fulfil a task. Discovery usually involves matching a formal task description against semantic descriptions of Web services.
- *Mediation*: we can not assume that the software components which we find are compatible. Mediation aims to overcome all incompatibilities involved. Typically this means mismatches at the level of data format, message protocol and underlying business processes.
- *Composition*: often no single service will be available to satisfy a request. In this case we need to be able to create a new service by composing existing components. Artificial Intelligence (AI) planning engines are typically used to compose Web service descriptions from high-level goals.

Significant results are already available, in terms of reference ontologies, e.g. OWL-S [9] and WSMO [5], comprehensive frameworks (e.g. DIP project[1] results), and more recently standards, e.g., SAWSDL[2]. Therefore, further research efforts are now investigating how SWS can be effectively applied – and in case improved - to solve other (Web-) service oriented computing problems.

In this paper we describe IRS-III (Internet Reasoning Service), a framework for creating and executing semantic Web services, which takes a semantic broker based approach to mediating between service requesters and service providers [2], [3]. More specifically, we have extended the core epistemological framework of our previous IRS-II framework [8] and incorporated the Web Services Modelling Ontology [5] conceptual model into the IRS-III framework.

In Section 2 we describe IRS-III specifically from an ontological point of view. In Section 3 we outline how SWS based systems can be successfully developed and deployed using IRS-III and we illustrate our approach through three different application domains: Business Process Management, e-Learning and e-Science. Section 4 concludes the paper.

2 IRS-III: A broker-based approach for SWS

IRS-III [2], [3] is a platform and broker for developing and executing SWS. A core design principle for IRS-III is to support capability-based invocation. A client sends a request which captures a desired outcome or goal and, using a set of semantic Web service descriptions, IRS-III will: a) discover potentially relevant Web services; b) select the set of Web services which best fit the incoming request; c) mediate any mismatches at the conceptual level; and d) invoke the selected Web services whilst adhering to any data, control flow and Web service invocation constraints.

To achieve this, IRS-III adopts a semantic Web based approach and is thus founded on ontological descriptions. At the heart of IRS-III there is the SWS Library, where semantic descriptions of various aspects of Web services, reference Domain Ontologies and Knowledge bases (instances) are stored using OCML representation

[1] http://dip.semanticweb.org/
[2] http://www.w3.org/2002/ws/sawsdl/

language [7]. Specific IRS-III components interpret such descriptions to discover and select the appropriate Web service, choreograph and ground to the Web service operations [4], orchestrate multiple Web services, and mediate semantic descriptions by running mediation rules or invoking mediation services [1]. Note that IRS-III supports grounding to standard Web services with a WSDL description, as well as stand-alone Java and Lisp code. Similarly, Web applications accessible as HTTP GET requests are handled internally by IRS-III.

2.1 IRS-III Service Ontology

The IRS-III service ontology forms the epistemological basis for IRS-III and provides semantic links between the knowledge level components describing SWS and the conditions related to its use. These descriptions are interpreted by the OCML reasoner. We describe the commonalities and differences between the service ontology and WSMO and then how the service ontology is used within IRS-III.

The IRS-III service ontology contains the following main items, which are also part of the Web Services Modelling Ontology [5]:

- Non-functional properties – these properties are associated with every main component model and can range from information about the provider such as the organisation's legal address, to information about the service such as category, cost and quality of service, to execution requirements such as scalability, security or robustness.
- Goal-related information – a goal represents the user perspective of the required functional capabilities. It includes a description of the requested Web service capability.
- Web service functional capabilities – represent the provider perspective of what the service does in terms of inputs, output, pre-conditions and post-conditions. Pre-conditions and post-conditions are expressed by logical expressions that constrain the state or the type of inputs and outputs.
- Choreography – specifies how to communicate with a Web service.
- Grounding – associated with the Web service choreography, a grounding describes how the semantic declarations are associated with a syntactic specification such as WSDL.
- Orchestration – the orchestration of a Web service specifies the decomposition of its capability in terms of the functionality of other Web services.
- Mediators – a mediator specifies which top elements are connected and which type of mismatches can be resolved between them.

The differences between our ontology and WSMO are described below:

- Meta-classes for the top-level SWS concepts – meta-class definitions for goal, mediator and Web service have been defined. These classes have a 'meta-' extension (e.g. meta-goal) and enable the IRS-III components to reason over the top-level concepts within the service ontology as first class entities.
- SWS user definitions as classes – following from the previous item, we enable users to define the required goals, mediators and Web services as subclasses of the corresponding WSMO concepts rather than as instances. In our view a class better captures the concept of a reusable service description and taxonomic

structures can be used to capture the constitution of a particular domain. For example, goals for booking flights may have sub-goals for booking European flights and for booking long-haul flights. A proposal for extending WSMO with goal templates, similar to our goal classes, has been suggested recently [11].

- SWS invocation contexts as instances – we reserve instances for invocation. When IRS-III receives a client request, instances of relevant goals, mediators and Web services are created to capture the current invocation context.
- Explicit input and output role declaration – in the interests of simplifying the definition of goals and Web services, our ontology incorporates explicit input and output role declarations. The declared input and output types are imported from domain ontologies. This feature enables SWS developers to view goals and Web services as 'one-shot' thus minimizing the need to consider complex interaction when appropriate.
- Orchestration and choreography language – the representation of our orchestration and choreography are defined within the service ontology.

Using SWS descriptions for implementing internal components, we implement several IRS-III internal components using the service ontology and OCML. Our assumption is that IRS-III components described through goals, mediators, and Web services and through ontological concepts and relations are easier to understand and maintain than if they were implemented purely in a programming language.

2.2 Using the Service Ontology

Before we describe the IRS-III server and its components we first highlight the main ways in which the service ontology is used to implement the core functionalities.

- Web services are linked to goals via mediators - if a wg-mediator associated with a Web service has a goal as a source, then this Web service is considered to solve that goal. An assumption expression can be introduced for further refining the applicability of the Web service.
- GG-mediators provide data-flow between sub-goals – in IRS-III, gg-mediators are used to link sub-goals within an orchestration and so they also provide dataflow between the sub-goals.
- Web services can inherit from goals - Web services which are linked to goals 'inherit' the goal's input and output roles. This means that input role declarations within a Web service are not mandatory and can be used to either add extra input roles or to change an input role type.

Client choreography – the provider of a Web service must describe the choreography from the viewpoint of the client. Within WSMO the choreography expresses a number of constraints which should not be violated when a deployed service is invoked. Within the IRS-III we evaluate the client choreography in order to interact with the deployed Web service.

Mediation services are goals – a mediator declares a goal as the mediation service which can simply be invoked. The required data transformation is performed by the associated Web service.

3 Creating Semantic Web Service Based Applications

Our generic application architecture is depicted in **Fig. 1**. As can be seen, the architecture is composed of four layers and enables collaboration between one or more stakeholders in a distributed fashion.

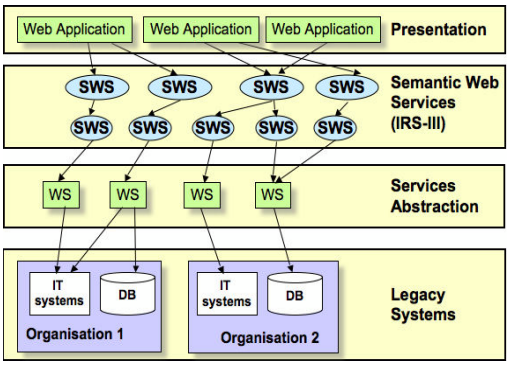


Fig. 1. The generic architecture used when creating IRS-III based applications

In particular, our approach enables the functionality provided by existing legacy systems from the involved business partners to be exposed as Web services, which are then semantically annotated and published using the IRS-III infrastructure. From the bottom up the four application layers are:

- Legacy system layer - consists of the existing data sources and IT systems available from each of the parties involved in the integrated application.
- Service abstraction layer - exposes the (micro-)functionality of the legacy systems as Web services, abstracting from the hardware and software platforms. In general existing Enterprise Application Integration (EAI) software will facilitate the creation of the required Web services. Note that for standard databases the necessary functionality of the Web services can simply be implemented as SQL query functions.
- Semantic Web services layer – given a goal request, this layer, implemented in IRS-III, will: a) discover a candidate set of services; b) select the most appropriate; c) resolve any mismatches at the data, ontological or process level; and d) invoke the relevant set of Web services satisfying any data, control flow and invocation requirements.
- Presentation layer – a Web application accessible through a standard Web browser which is built upon the semantic Web services layer. The goals defined within the semantic Web services layer are reflected in the structure of the interface and can be invoked either through the IRS-III API or as an HTTP GET request. We should emphasize that the presentation layer may be comprised of a set of Web applications to support different user communities. In this case each community would be represented by a set of goals supported by community related ontologies.

In order to successfully create applications from semantic Web services as depicted in **Fig. 1** above four key activities need to be carried out as follows:

1. Requirements capture – during this step the requirements for the overall application are captured. Although there is no prescribe methodology, the resulting documents should describe the stakeholders, the main users and roles, any potential providers for Web services, and any requirements on the deployed infrastructure and interfaces.
2. Goal description – using the requirements documents above, relevant goals are identified and described in IRS-III. During this process any required supporting domain ontologies will be created.
3. Web service description – descriptions of relevant Web services are created within IRS-III. Again, any domain ontologies required to support the Web service descriptions are defined.
4. Mediator description – mismatches between the ontologies used, and mismatches within and between the formal goal and Web service descriptions are identified and appropriate mediators created.

All of the above steps are carried out by the SWS application developer. The first two steps are user/client centric and therefore involve discussions with the relevant client stakeholders, whereas Step 3 will require dialogue with the Web service providers. Steps 2 and 3 are mostly independent and in the future we expect libraries of goals and Web services to become generally available to support re-use.

3.1 Business Process Management

Business Process Management (BPM) intends to support "business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information" [13]. BPM aims to support the complete life-cycle of business processes, however, by doing so BPM has made more evident the existing difficulties for obtaining automated solutions from high-level business models, and for analyzing the execution of processes from both a technical and a business perspective. The fundamental problem is that moving between the Business Level and the IT Level is hardly automated.

Semantic Business Process Management (SBPM) that is, the combination of Semantic Web and Semantic Web Services technologies with BPM, has been proposed as a solution for overcoming these problems [6]. SBPM aims at accessing the process space of an enterprise at the knowledge level so as to support reasoning about business processes, process composition, process execution, etc. Major efforts are currently devoted to pursuing the SBPM vision in the context of the European project SUPER[3]. The project follows a multi-layered approach where a number of standard languages and notations have been mapped to a stack of ontologies supported by a suite of semantically enhanced tools.

Within this project IRS-III is playing several key roles centered on its capabilities as a Semantic Web Services platform and its strong ontology reasoning support. The

[3] http://kmi.open.ac.uk/projects/super/

research in IRS-III carried in the context of SBPM is focused in several issues such as:

- Supporting the design of business process models – Adding formal semantics to business process models enabling business analysts to:
    - Find relevant existing process models for solving a business task, which match a given business context (e.g. business domain regulations or organizational policies),
    - Create new processes through the composition of processes exposed as Semantic Web Services,
    - Mediate between incompatible processes which are required to be to be connected.
- Generating an executable process model – Using ontological descriptions to move from an informal (usually diagram-based) business-level process model to a model which can be executed within an engine.
- Monitoring the progress of a running process – providing semantically rich information on the status of currently running processes, within a corporate ICT infrastructure, in a fashion which is understandable to the business analyst.

3.2 E-Learning

E-Learning aims to support students to achieve a predefined learning outcome. Current approaches consider the usage of software systems – e.g. Learning Content Management Systems (LCMS) – that provide the learner with composite learning contents: the so called Learning Objects (LOs). Based on either proprietary or standard metadata, a LO usually defines the learning process - i.e. the sequence of activities the learner has to follow to achieve his/her learning objective – as well as the set of learning resources – data or services - associated to each activity of the process. Since metadata standards mainly rely on syntactic descriptions, human developers are needed to understand the intended meaning of the metadata and carry out manually the activities related to learning process composition and resource allocation. Therefore, current approaches limit the reusability of existing learning resources available on the Web and restrict the ability of a learning application to adapt automatically to specific learning requirements and learning contexts.

In the context of the European project LUISA[4], we propose to move from the existing data and metadata based paradigm to a highly dynamic service-oriented approach based on semantic Web service technologies. To enable this vision, we adopt a semantic approach which abstract from data, services and existing process metadata standards. By making use of ontologies, we represent (i) a process in terms of sequences of learning goals to achieve and (ii) the learning context – i.e. domain requirements or learner profile and preferences - where the process is performed. At runtime, given a learning goal and the reference process context, IRS-III can identify and deliver the most appropriate resources that allow the learner to accomplish such a goal.

[4] http://kmi.open.ac.uk/projects/luisa/

As a result, we enhance the current state of the art by enabling context adaptive e-Learning applications based on distributed, flexible and open infrastructures. Starting from our semantic representation of processes, we can ground to multiple existing metadata standards and thus reuse the respective runtime environments. Instead of grounding the metadata standard activities to static learning resources, we link them to our learning goals. When the standard-compliant application processes an activity, the associated learning goal is invoked. Several services on top of repositories from different organizations can be linked to and thus provide resources for a specific learning goal. Their selection is based on axioms that declare the assumed learning context for a service. If a specific goal is not achievable by existing services, an opportune SWS orchestration can create on-the-fly integrated services. Note that for each goal new services can be easily integrated by simply introducing the respective semantic descriptions, without affecting the existing structure. Finally, each service can provide resource following different standards (or not following any standard at all), since appropriate mediation services can be used to address existing data heterogeneities.

3.3 E-Science

A number of large research initiatives[5] aim to develop computer models of human physiology that span multiple dimensional and temporal scales. EuroPhysiome[6] is an initiative to promote the development of the Virtual Physiological Human (VPH), a methodological and technological framework that will enable medical investigators to consider the human body as a single (though still hugely complex) system.

While the VPH will have a sizeable impact on all branches of biomedical research and clinical medicine, a primary target domain is the Musculoskeletal System, which we address in the Living Human Digital Library (LHDL[7]) project.

In LHDL, we serve a virtual community comprising students and professionals engaged in researching the musculoskeletal system. They are interested in accessing and managing complex biomedical data. Using Web services, LHDL researchers can share data, algorithms, and community services. In a large and complex domain like the biomedical one, understanding and coordinating these services is a major task. By adding formal semantic descriptions of the Web services, we can recruit computers to perform much of this reasoning for us.

IRS-III uses these semantic annotations to alleviate many of problems that usually impede full and easy interoperability between the kinds of heterogeneous resources deployed in huge context such as VPH. IRS-III assists in the technical tasks of finding, composing, and resolving mismatches between Web service components, as well as reason about high-level policy issues such as patient privacy, data security and provenance, and computational resourcing. In more detail, Web services represent the services that each VPH community will expose. A VPH user performs a request by a VPH portal; the portal sends the request which captures a desired outcome or goal

[5] http://www.mygrid.org.uk/, http://www.esnw.ac.uk/
[6] http://www.europhysiome.org/
[7] http://kmi.open.ac.uk/projects/lhdl/

and, using a set of semantic Web service descriptions, IRS-III will: a) discover potentially relevant Web services in any VPH sub-community; b) select the set of Web services which best satisfy the user request; c) mediate any mismatches; and d) invoke the selected VPH Web services according with any Web service invocation constraints.

The execution sequence of a complex semantic Web service is not hard-coded, but it is dynamically created using goal-based discovery and invocation: several Web services may be associated with a goal, and only the most applicable will be discovered and invoked at runtime (late binding). If a new service is available within one VPH community, the developers simply need to describe and then link it to an existing goal; if a service is altered, only the specific semantic description will be affected, and not the whole business process.

4 Conclusions

Semantic Web services research has the overall vision of bringing the Web to its full potential by enabling applications to be created automatically from available Web services in order to satisfy user goals. Fulfilling this vision will radically change the character of all online interaction including the nature of e-Commerce, e-Science, e-Learning, and e-Government. Key to achieving this vision is the provision of SWS platforms able to support the development and use of online libraries of reusable software components indexed through generic and domain specific ontologies. In this paper we have presented our SWS platform IRS-III, which contains a suite of tools to enable the development and management of semantic descriptions. Using the semantic Web service descriptions, IRS-III, through orchestration, mediation and choreography components, can broker between incoming goal requests and applicable Web services.

Over the past few years we have used IRS-III to create a number of SWS based applications. Within a number of new EU funded projects we are currently creating applications in the areas of: business process modelling, linking IRS-III to a BPEL engine[3]; e-learning, integrating IRS-III with learning resource repositories[4]; and, bio-informatics, describing Grid services related to the human musculo-skeletal system[7]. The diversity of the domains in which we are able to deploy IRS-III is evidence of the utility and robustness of our approach, and, we fully expect to gain further valuable insights into the overall requirements for semantic Web services during the deployment process. In this respect we welcome external parties to use our platform - the IRS-III API and browser for can be downloaded from the IRS-III Web site at http://kmi.open.ac.uk/projects/irs/.

Acknowledgments. This work was supported by the SUPER (Semantics Utilized for Process management within and between Enterprises) project, (FP6 – 026850); LUISA (Learning Content Management System Using Innovative Semantic Web Services Architecture) project (FP6 – 027149); LHDL (Living Human Digital Library) project (FP6 – 026932).

References

1. Cabral, L., Domingue, J.: Mediation of Semantic Web Services in IRS-III. In Proceeding of the Workshop on Mediation in Semantic Web Services in conjunction with the 3rd International Conference on Service Oriented Computing, Amsterdam, The Netherlands, (2005).
2. Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Norton, B., Tanasescu, V., Pedrinaci, C.: IRS-III: A Broker for Semantic Web Services based Applications. In Proceedings of the 5th International Semantic Web Conference, Athens, USA, November, (2006).
3. Domingue, J., Cabral, L., Galizia, S., and Motta, E.: A Comprehensive Approach to Creating and Using Semantic Web Services, In Proceedings of the W3C Workshop on Frameworks for Semantics in Web Service, Innsbruck, Austria, June 9-10, (2005).
4. Domingue, J., Galizia, S., and Cabral, L.: Choreography in IRS-III- Coping with Heterogeneous Interaction Patterns in Web Services. In Proceedings of 4th International Semantic Web Conference, Galway, Ireland, (2005).
5. Fensel, D., Lausen, H., Polleres, A., De Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: Web Service Modeling Ontology. Springer, (2006).
6. Hepp, M., Leymann, F., Domingue, J., Wahler, A. and Fensel, D.: Semantic business process management: A vision towards using semantic web services for business process management. In ICEBE, pages 535–540, (2005).
7. Motta, E.: An Overview of the OCML Modelling Language, In Proceedings of the 8th Workshop on Knowledge Engineering Methods and Languages (KEML '98). (1998).
8. Motta, E., Domingue, J., Cabral, L., and Gaspari, M.: IRS-II: A Framework and Infrastructure for Semantic Web Services. In Proceedings of the 2nd International Semantic Web Conference (ISWC2003), 20-23 October 2003, Sanibel Island, Florida, USA.
9. OWL-S Working Group: OWL-S 1.2 Pre-Release, (http://www.ai.sri.com/daml/services/owl-s/1.2/). (2006).
10. SOAP: SOAP Version 1.2 Part 0: Primer, (http://www.w3.org/TR/soap12-part0/. (2003).
11. Stollberg, M. and Norton, B.: A Refined Goal Model for Semantic Web Services. In Proc. of the 2nd International Conference on Internet and Web Applications and Services (ICIW 2007), Mauritius, (2007).
12. UDDI: UDDI Spec Technical Committee Specification v. 3.0, http://uddi.org/pubs/uddi-v3.0.1-20031014.htm. (2003).
13. Van der Aalst, W., Ter Hofstede, A., and Weske, M.: Business process management: A survey. In Business Process Management, pages 1–12, (2003).
14. WSDL: Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315. (2001).

Improving Responsiveness of Ontology-Based Query Formulation

Ivan Zorzi, Sergio Tessaris, and Paolo Dongilli

Free University of Bozen-Bolzano, Italy
<lastname@inf.unibz.it>

Abstract. Recent research showed the benefits of adopting formal ontologies as a means for accessing heterogeneous data sources. The use of an ontology not only provides a uniform and flexible approach for integrating and describing these sources, but it can be used to improve the usability of an integrated system by guiding the final user to formulate his/her information needs. More precisely, the task of formulating queries can be supported by an intelligent use of the ontology describing the information sources. In fact, it has been proved that an appropriate use of automated reasoning techniques can support a user in formulating a precise query —which best captures her/his information needs— even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. Previous work has been carried out on intelligent interfaces for query formulation and this paper describes how to improve usability of such systems by reducing the calls to reasoning services.

1 Introduction

In this paper we introduce the reader to the features of an intelligent query interface. We simply call it Query Tool and it was devised to enable users to access heterogeneous data sources by means of an integrated ontology. The Query Tool supports the users in formulating a precise conjunctive query, where the intelligence of the interface is driven by reasoning services running over a given logic-based ontology.

The ontology, which describes a given domain, defines a vocabulary which is richer than the logical schema of the underlying data, and it is meant to be closer to the user's wide vocabulary. The user can exploit the ontology's entities to formulate the query, and she is guided by such a richer vocabulary in order to understand how to express her information needs more precisely, given the knowledge of the system. This latter task —called intensional navigation— is the most innovative functional aspect of our interface. Intensional navigation can help a less skilled user during the initial step of query formulation, thus overcoming problems related to the lack of schema comprehension and enabling her to easily formulate meaningful queries. Queries can be specified through an iterative refinement process supported by the ontology via intensional navigation. The user may specify her request using generic terms, refine some terms

of the query or introduce new terms, and iterate the process. All details are thoroughly described in the coming sections.

Furthermore we draw the attention of the reader towards the optimisation techniques we are applying to the Query Tool in order to improve the usability of the system. Improvements are made working on three fronts: reducing as much as possible the calls to the reasoner, storing the taxonomy, and caching query information.

The paper is organised as follows. First of all we describe the technologies and techniques underlying the system, then we present the actual system (Query Tool) from the user perspective, with a complete exposition of the functionalities of the interface. Afterwards we illustrate the interaction with the reasoning services followed by a section on the optimisations of such a system. Finally, in the discussion section, we show how we are also leveraging natural language generation technologies to enhance user interaction with the interface.

2 Background

2.1 Ontology mediated access to data sources

The purpose of the presented Query Tool is to support query formulation in the context of information access mediated by ontologies. More specifically, the scenario in which we consider the deployment of the tool consists of one or more data sources providing their own query language (e.g. they can be relational sources). The information provided by the sources is described by means of a global ontology together with mappings relating the ontology vocabulary to the vocabulary of the data sources. We do not impose any constraint on the kind of mappings and/or architecture underlying the integration system.

The Query Tool relies on the availability of an ontology providing the vocabulary for the queries and a query engine capable to retrieve the data. These minimal requirements enable the Query Tool to be used in simple cases in which data are retrieved from a knowledge base (see [10]) as well as more complex architectures in which query answering requires complex processing (e.g. using rewriting [2]).

The ontology language adopted by the tool is OWL-DL (see [6]), therefore the conceptual model exposed to the user centres around the concept of classes and properties. While the user is guided to the construction of queries structured in terms and properties which can be refined (see the next sections for details), the system generates conjunctive queries composed by unary (classes) and binary (attribute and relation) predicates.

2.2 Queries

The Query Tool represents queries to the user as trees, in which nodes are labelled by classes and edges by properties. Each node of the tree correspond to a different variable and properties (edges) constitute the joins between a node

and the rest of the query. In this way the conjunctive queries generated by the system are acyclic.[1]

Users interact with the system to refine the query by a set of operations which can be performed on nodes of the query tree. Once selected, a node becomes the focus for the operations which can be divided into substitution (when a class is substituted by more general or specific one) and incremental refinement by addition of compatible classes or properties. Additionally, the system allows the deletion of part of the query.

For each focus the tool suggests the terms and/or properties which can be used to refine the query. This step requires the interaction with an OWL-DL reasoner in order to establish which properties or classes are "compatible" with the current query. This must be done in real time when the user interacts with the tool, since both the query and the focus affect the responses from the reasoner.

For more details on the query language and the user perspective over the tool, the reader is referred to [4]; in this paper we concentrate on showing how we increased the responsiveness of the system by optimising the use of the OWL reasoner.

2.3 Reasoning services

An OWL reasoner is employed to derive the information required to drive the interface. These information range from the taxonomical position of an OWL expression w.r.t. the terms of the ontology, to the satisfiability of an expression.

To allow for the maximum flexibility, the tool communicates with the reasoner by means of the DIG API (see [1]). To one side this enables the possibility of using any compliant reasoner; but on the other side the use of HTTP as underlying transport introduces additional overhead in terms of network connections.

For this reason, one of the first goal we wanted to achieve is to minimise the number of calls to the reasoner (see Section 5).

2.4 An Example

Now we want to present an example that will be referred throughout the paper to better understand the operations involving the reasoner. To do so, we employ an excerpt of the Wine Ontology which is shown in Fig. 1. We adopted the UML notation to represent the *is-a* relationships among terms and we introduced constraints of disjointness where needed.

We have *Wine_Descriptor* as root concept and *Wine_Taste* and *Wine_Colour* as specialisations of Wine_Descriptor.The concept *Wine* has a property *has_Colour* towards concept Wine_Colour; the inverse of this property is *colour_Of* when seen from concept Wine_Colour. Wine_Colour specialises in *Red* and *White* while Wine in *Red_Wine*, *White_Wine*, and *Table_Wine* respectively. Because of lack of space, we are not going to present the axioms of this sample ontology that can anyway be represented in OWL-DL.

[1] From the technical point of view cyclic queries wouldn't pose any problem; however, usability tests conducted in the context of a previous project suggested that the users don't find co-references intuitive enough.

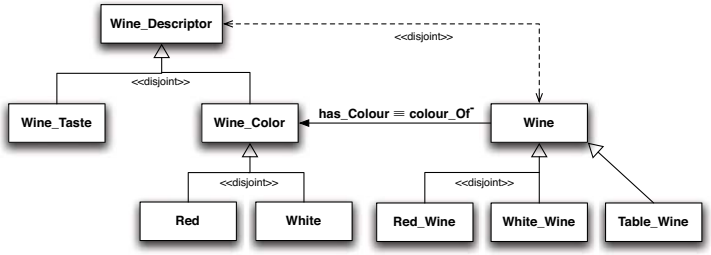


Fig. 1. An excerpt of the Wine Ontology.

3 Query Tool

In this section we present a brief description of the end user system. It is a Java-based application adopting the Standard Widget Toolkit (SWT) [11] for creating the graphical user interface. The system requires at least JRE 1.4 and a DL reasoner providing a DIG 1.x interface.

The query interface is provided with four Tabs:

- Admin: administrative interface used to load the ontology and connect to the reasoner.
- Compose: main query composition interface.
- Query: displays the actual query, mainly for debugging purposes.
- Results: displays the results of the query evaluation.

Initially the user is presented with the *Admin* Tab (see Figure 2). Here, some preliminary operations necessary for the query formulation have to be executed:

1. **Connection setup**: one of the operations the user has to carry out consists in testing the reasoner connection. A reasoner with reference to the ontology is used by the system to drive the query interface: in particular, it is used to discover the terms and properties which are proposed to the user to manipulate the query.
2. **Loading and managing ontology files**: all the operations the system provides cannot be accomplished without loading an ontology. The interface allows the user to specify an ontology in DIG 1.x format to be loaded into the system. Once the ontology is loaded into the system, the user has also the possibility to adjust the content of that ontology, depending on her needs; if the user wants that the modifications take a permanent effect, she can save them back to the file. As a matter of fact, users might frequently have the necessity to extend an ontology in order to obtain different results or to correct it as a consequence of unexpected behaviour.
3. **Loading a metadata file**: the interface gives the possibility to the user to specify a metadata file to be loaded into the system. Metadata files contain valuable information about the terms in the ontology; that information concerns essentially the lexicalisations of those terms. Actually, as the terms

contained in the ontology could be expressed by a sort of shorthand, their lexicalisations are provided so that the user can deal with clearly understandable terms.

4. **Customising lexicalisations**: given the metadata file, the interface offers to the user the opportunity to apply desired variations to the lexical information of the terms. Those variations can be saved back to a metadata file or just saved temporarily in the system. The query to be generated should be as unambiguous as possible: if the user can assign to the terms the lexicalisations which best give significant importance to her, the query formulation will be transparent and therefore the really intended result will be retrieved.

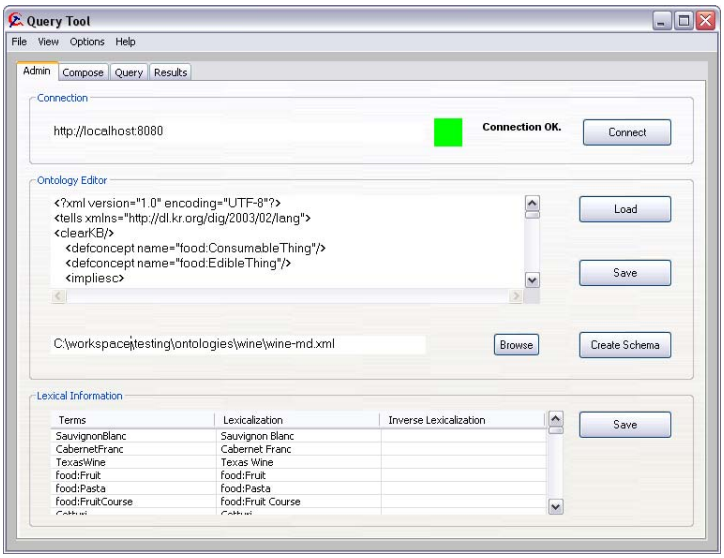


Fig. 2. Administrative interface of the Query Tool.

As you can see in Figure 2, the reasoner connection has been tested by means of the "Connect" button. An ontology has been loaded ("Load" button) and also a metadata file ("Browse" button). Subsequently, the "Create Schema" button has been clicked and all the lexicalisations of the ontology terms are presented in the "Lexical Information" table. Here the user can change the lexicalisations by clicking on the cell corresponding to the lexicalisation she wants to modify.

In the *Compose* Tab (see Figure 3) the user can formulate the query by means of pop-up menus presenting the possible operations. Initially the user is presented with a choice of different starting terms (all the concepts in the ontology or a subset defined by means of the metadata file): she selects the first term to be added in the query. Subsequently, the interface gives the possibility to perform the following operations:

– **Add compatible terms**: other terms specified in the ontology can be added to the query. The compatible terms are automatically suggested to the user by means of appropriate reasoning tasks on the ontology describing the data sources. Indeed, the system suggests only the operations which are compatible with the current query expression.
– **Substitute terms**: the system gives the opportunity of substituting the selected term of the query with a more specific or more general term. It can also be the case that in the ontology there are terms which are equivalent to the selected one: in this case the user is offered to replace the selection with an equivalent term.
– **Delete terms**: as the query is specified through an iterative refinement process, it could be the case that the user needs to delete some terms from the query.
– **Add or delete properties**: analogously, the user can add properties to the query. A property can be a relation or an attribute. The interface suggests a list with the possible alternatives. The user can specify some restriction values to attributes.

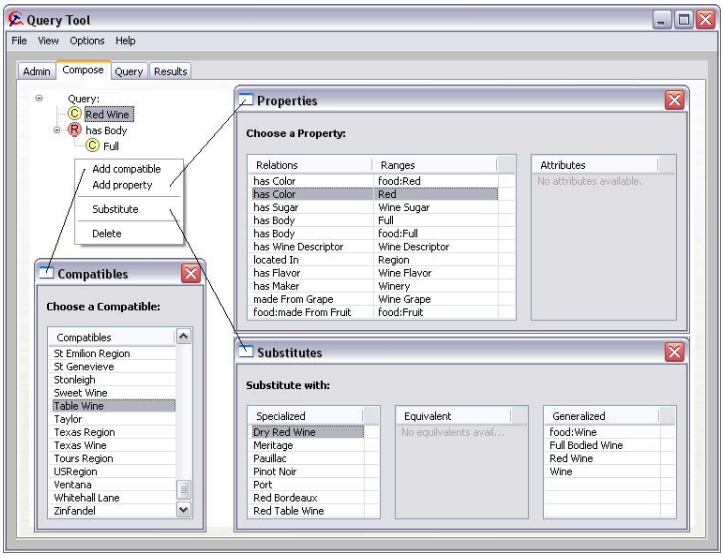


Fig. 3. Query composition interface.

The first operation to compose a query consists in selecting the starting term. By clicking on a pop-up menu ("Choose starting term") the user is presented with a windows showing all the terms that can be used as starting term.

Once the user has selected the starting term, it is possible to refine the query using again the pop-up menu. The operations allowed are listed in the pop-up

menu; the user can add a compatible term, add a property (relation or attribute), substitute the term or delete it.

If the user selects an attribute, it is possible to set it as distinguished variable or to add a restriction to the attribute. The user can also delete properties or terms from the query and select new ones.

Once the user has formulated the query, the *Query* tab shows the query in XML and DIG formats (the menu bar "Options" allows also to view the query in the corresponding SQL code). Finally, in the *Results* tab the user can retrieve the results (if any) corresponding with the formulated query.

In the menu bar, by clicking on "View" menu, the user can have a look to the log file ("View" log) of the application and also a concise description of the schema with all the taxonomy ("View" schema).

4 Reasoner interaction

In this section we describe all the operations (w.r.t. the reasoner) that users can perform during the query formulation process. Refinement of the query expression can be done by the following operations:

- addition of a compatible term;
- addition of a property;
- substitution of a term with an equivalent, more specific or more general term.

In primis we present the approach which enables the system to interact with the reasoner and then the formalisation of the above operations.

4.1 Query rolling-up

Before discussing the actual operations in terms of their use of logical deductions, we need to introduce a fundamental manipulation of the queries which enables us to exploit the reasoning services provided by a DIG reasoner. This operation is the so called *rolling up* of an acyclic conjunctive query (see [8]).

Roughly speaking, the rolling up transforms an arbitrary query without cycles into an equivalent DL expression. The key idea behind is the fact that a (sub)query of the form $P(x, y), R(y)$ is equivalent to the DL expression $(\exists P.R)(x)$. Any variable can be selected as the root of the tree (since we consider acyclic queries) and the rest of the query can be "rolled-up" starting from the leaves.[2]

To analyse the properties of a given query focused on a specific variable, say q^x, we roll-up the query using the focus as the root (with variable x associated with concept F) and then we interrogate the reasoner using the resulting complex concept $Q^{F(x)}$. In the following sections we describe in detail the procedure we employ.

[2] Inverse roles provide the possibility of collapsing queries of the form $P(y, x), R(y)$ as well. If the ontology doesn't include transitive roles and nominals, cyclic queries can be handled in the same way (see [7]). We're considering techniques to allow more expressive languages.

4.2 Addition of a compatible term

This operation requires the list of terms "compatible" with the given query. In terms of conjunctive queries, it corresponds to add a new term to the query. The term is compatible and can be added to the query if the resulting query is satisfiable. Let us formally define a compatible term w.r.t. a query. Given an ontology Σ and a focused query $Q^{F(x)}$ we want to find all the terms $Y \in \mathbb{C}$ (where $\mathbb{C}$ are all the unary atomic terms) such that:

$\Sigma \not\models Q^{F(x)} \sqcap Y \sqsubseteq \bot$	Y is not disjoint with $Q^{F(x)}$
$\Sigma \not\models Q^{F(x)} \sqsubseteq Y$	Y is not among ancestors of $Q^{F(x)}$
$\Sigma \not\models Y \sqsubseteq Q^{F(x)}$	Y is not among descendants of $Q^{F(x)}$

The reasoning service makes use of satisfiability to check which predicates in the ontology are compatible with the current focused query. This check corresponds simply to the addition of the term Y to the focused query $Q^{F(x)}$, and verify that the resulting query is satisfiable. Actually, this operation is very expensive because the number of reasoner calls matches the number of unary predicates.

Going back to the example of Sec. 2.4, if we have a query with concept Red_Wine and we want to find all the concepts which are compatible with it, it will turn out that concept Table_Wine is compatible with the query while concept White_Wine is incompatible since it is disjoint with Red_Wine.

4.3 Addition of a property

The addition of a property requires the discovery of both a binary term and its restriction (or range). The system should check all the different binary predicates from the ontology for their compatibility. Formally, a property P is compatible with a focused query $Q^{F(x)}$ if

$$\Sigma \not\models Q^{F(x)} \sqcap \exists P.\top \sqsubseteq \bot,$$

where $\top$ represents any possible concept of the domain.

This is practically performed by verifying the satisfiability of the query $Q^{F(x)} \sqcap \exists P.\top$, for all atomic binary predicates P in the signature. Once a binary predicate is found to be compatible with $Q^{F(x)}$, repeated satisfiability is used to select the least generic unary predicate $Y \in \mathbb{C}$ such that the query $Q^{F(x)} \sqcap \exists P.Y$ is satisfiable. In other terms, the operation would consist in determining which are the compatible properties first, and then establishing which are the restrictions applicable to P. To discover all compatible properties, we need a number of reasoner calls equal to the number of properties in Σ. In addition, for each property found, to determine its restriction, we need as many reasoner calls as the number of unary predicates.

Again, returning to the example (see Sec. 2.4), this time we want to discover the properties which are compatible with the query Wine_Descriptor. As compatible properties propagate upwards in the hierarchy, property colour_of would be among the compatible properties of Wine_Descriptor. If the user instead composes a query with the concept Wine_Taste, the property colour_of would

be incompatible because the concept Wine_Taste is disjoint with the concept Wine_Colour.

4.4 Substitution of a term

Here we want to substitute a focused term of the query with an equivalent, more specific or more general term. Let us examine the substitution with a more specific term. In this case we need to perform a containment test of two conjunctive queries. Given a query focussed on concept F ($Q^{F(x)}$), we are interested in the unary terms Y subsumed by $Q^{F(x)}$, where Y must be the most general concept among the terms found (i.e. there is no other concept Y subsumed by $Q^{F(x)}$ and containing Y). Formally, given an ontology Σ and a query $Q^{F(x)}$, we want to find all the terms $Y \in \mathbb{C}$ such that:

$$\Sigma \models Y \sqsubseteq F, \neg\exists Z \in \mathbb{C} \,|\, (Z \sqsubseteq F, Y \sqsubseteq Z, Z \neq Y).$$
$$\Sigma \not\models F \sqcap Q^{F(x)} \sqcap Y \sqsubseteq \bot.$$

From Figure 1 it is possible to see that if the query is composed by concept Wine and we want to substitute it with a more specific term, we would get Red_Wine, White_Wine, and Table_Wine as candidates for the substitution since they are direct children of concept Wine.

The cases of substitution with more general and equivalent terms are analogous.

For the sake of clarity we report the sequence of operations needed to retrieve the substituting terms:

- query rolling-up;
- retrieval of incompatible classes: the descendants of negation of the query;
- retrieval of parents and children of the substituting term;
- filtering using incompatible terms.

We will see in Section 5.1 that a similar procedure is adopted to reduce the calls to the reasoner when looking for compatible terms.

5 Optimisation

As discussed in Section 2.3, the system relies on a DL reasoner to drive the query interface. If on one hand reasoning services with satisfiability and classification allow only to formulate consistent queries, on the other hand they introduce performance issues. Reasoner calls are expensive and should be therefore minimised as much as possible. The expensiveness of reasoner calls depends both on complexity and the fact that DL reasoners exploit the HTTP protocol to communicate.

In the following we present some optimisation techniques which can improve the usability of the system via a more responsive interface. Aim of the optimisation is to reduce the transitions between the query interface and the reasoner. Some techniques have already been exploited to reduce the number of reasoner

calls especially in the retrieval of compatible terms to be added to the query. Another important improvement comes from the storage of the taxonomy. Finally, information concerning the query can be cached during the query formulation process in order to extract some deductions to reduce reasoner calls.

5.1 Reducing reasoner calls

Concerning the refinement of the query by compatible terms, the basic policy to retrieve the compatible terms is to use the satisfiability reasoning service to check which unary predicates in the ontology are compatible with the current query. This check corresponds simply to the addition of the term to the current query, and to verify that the resulting query is satisfiable. Actually, this kind of operation is very expensive because the number of reasoner calls corresponds to the number of unary predicates in the ontology.

We adopted a different implementation in the current system. We classify the query and retrieve the its equivalents, ancestors, and descendants; then we classify the negation of the query and retrieve the descendants which are incompatible. The remaining unary predicates are the compatibles (see Section 4.2).

In reference to the addition of a property, as we discussed in Section 4.3, this operation requires the discovery of both a binary term and its restriction. One of the advantages of OWL-DL is the possibility of expressing the inverse of a role which is extremely useful for determining compatibility of binary terms. Hence, to discover the restriction of a property we use classification instead of repeated (and expensive) satisfiability. The idea is to classify the inverse of the property restricted to the query.
For example, to discover the restriction of property has_Colour applied to the query expression

$$\{x_1 \mid \text{Red\_Wine}(x_1), \text{Table\_Wine}(x_1)\},$$

we classify the expression $\exists \text{has\_Colour}^{-}(\text{Red\_Wine} \sqcap \text{Table\_Wine})$.
The reasoner returns the list of concept names more general and equivalent as range candidates of the relation has_Colour, when restricted to the domain (Red_Wine⊓Table_Wine). This method, not only lets us discover the least general predicate(s) which can be applied to the property in the given context, but also allows us to discard those properties which are incompatible with the query, i.e. bottom ($\bot$) is returned as range whenever a given property is incompatible with the query. Summarizing, we are able to both check the compatibility of a property with the query and find out the property's range by means of one single reasoner call.

5.2 Taxonomy storage

The taxonomy of the ontology provides static information concerning primitive concepts. If we store the taxonomy before starting to compose a query, initial operations like substituting a concept, would not involve the reasoner, thus improving efficiency.

The taxonomy is actually a partial order '$<$' from *Top* ($\top$), the whole domain, to *Bottom* ($\bot$), the empty set, where the partial order relation is subsumption. The partial order can be represented by a *directed acyclic graph (DAG)*, i.e. a directed graph that contains no cycles. An edge is drawn from a to b whenever $a < b$. A partial order satisfies the following properties:

- transitivity, $a < b$ and $b < c$ implies $a < c$;
- non-reflexive, not$(a < a)$.

These condition prevent cycles because $a < b < ... < z < a$ would imply that $a < a$, which is false. The only exception where the property $a < a$ holds is for the equivalent concepts.

The idea is to save not only the taxonomy but also other information pertaining each concept such as e.g. its incompatible classes and the list of incompatible properties.

5.3 Caching query information

The *focus* plays an important role during the query formulation process; in particular the system proposes the available operations on the query w.r.t. the current focus (i.e. the variable which is currently selected). The focus is crucial also for caching dynamic information concerning the query and the idea is to cache both the query and its actual classification at the focus level. In other words, we want to associate to each single variable which gets the focus the overall status of the query. Of course, cache at the single node level would be invalidated as soon as the user further refines the query.

An intuitive approach to exploit the cache would consist in modifying the system in a way that the user can only remove terms by following the exact inverse order of the one which has been used to formulate the query. This means that only the last operation can be undone. In this way we could reduce or even avoid reasoner calls because the information we need has already been cached at the node.

We know that refinement of the query is monotonic and therefore whenever the user adds new terms to the query, the domain is going to be restricted. This property can also lead us to some conclusions for reducing reasoner calls in further refinements of the query. This property does not hold when the user deletes a term from the query; in this case all the cache has to be removed.

6 Discussion

Optimising the communication and the quantity of exchanged messages behind the scenes between the Query Tool and the reasoner is only the first action we are taking to make the use of the interface more comfortable for the user.

The interaction time we are able to save with these enhancements is *partially* re-invested in the demands of a new and more complex interface we are building, based on state-of-the-art natural language generation (NLG) technologies.

The main challenge is that the query (now with partial verbalisation of single concepts and roles) is to be presented to the user in natural language with full verbalisation, and stepwise refinements of the query composed by the user are presented as natural language refinements that maintain the grammaticality of the sentences representing the query. Our solution adopts the paradigm called WYSIWYM ('What You See Is What You Meant'), a user-interface technique which uses (NLG) technology to provide feedback for user interaction [9]. The differences between our approach and that used by available systems employing WYSIWYM are explained in [5], while [3] reports our experiments in terms of discourse planning strategies of a complex concept description (query).

References

1. Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG Description Logic Interface. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*, 2003.
2. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. Ontology-based database access. In *Proc. of the 15th Italian Conf. on Database Systems (SEBD 2007)*, 2007.
3. Paolo Dongilli. Discourse planning strategies for complex concept descriptions. In *Proceedings of the 7th International Symposium on Natural Language Processing (SNLP-2007)*, Pattaya, Chonburi, Thailand, December 2007.
4. Paolo Dongilli, Enrico Franconi, and Sergio Tessaris. Semantics driven support for query formulation. In *Description Logics*, 2004.
5. Paolo Dongilli, Sergio Tessaris, and John Bateman. Leveraging Systemic-Functional Linguistics to Enhance Intelligent Database Querying. In *Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications*, Jinan, China, October 2006.
6. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
7. Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. How to decide query containment under constraints using a description logic. In *Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000.
8. Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. In Ian Horrocks and James Hendler, editors, *Proc. of the 2002 International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science. Springer-Verlag, 2002.
9. Richard Power and Donia Scott. Multilingual Authoring Using Feedback Texts. In *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 98)*, pages 1053–1059, Morristown, NJ, USA, 1998. Association for Computational Linguistics.
10. Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *3rd OWL Experiences and Directions Workshop (OWLED-2007)*, 2007.
11. SWT. The Standard Widget Toolkit. `http://www.eclipse.org/swt`, 2007.

Using WordNet to turn a folksonomy into a hierarchy of concepts

David Laniado, Davide Eynard, and Marco Colombetti

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133 Milano, Italy
{david.laniado,eynard,colombet}@elet.polimi.it

Abstract. As the volume of information in the *read-write Web* increases rapidly, folksonomies are becoming a widely used tool to organize and categorize resources in a bottom up, flat and inclusive way. However, due to their very structure, they show some drawbacks; in particular the lack of hierarchy bears some limitations in the possibilities of searching and browsing. In this paper we investigate a new approach, based on the idea of integrating an ontology in the navigation interface of a folksonomy, and we describe an application that filters del.icio.us keywords through the WordNet hierarchy of concepts, to enrich the possibilities of navigation.

1 Introduction

As the amount of information available in the Web grows every day faster, the task of classification is getting harder, the traditional top down approach is getting inadequate [1], and the new collaborative approach of *folksonomies* is emerging [2].

In folksonomies users can associate freely chosen tags to resources and in this way they produce knowledge for the entire community. Beside their dynamism and low cost, folksonomies present many disadvantages: in particular, their lack of hierarchy limits the possibility of searching and browsing related information.

Our purpose is to enrich the possibilities of navigation in a folksonomy by adding some explicit semantics, provided by a static hierarchy of concepts, to help users orient themselves among keywords. We chose to start with del.icio.us[1], one of the most popular folksonomies for social bookmarking, and to develop an alternative tool for the suggestion of related tags, based on the WordNet hierarchy of concepts.

In this paper, after a brief description of the current related work (Section 2), we describe both the design and the implementation of our project (Section 3). In Section 4 we show some results of our tests and an evaluation of the application, then in Section 5 we conclude with a summary and a discussion of future work.

[1] `http://del.icio.us`

2 Current Work

Joshua Schachter, founder of del.icio.us, defined it as *"a way to remember in public"*; in folksonomies each user can generally explore two spaces, the one of his bookmarks and the one of everyone's bookmarks; tags can be used to filter items.

As the work of categorization is performed by users, folksonomies are democratic, scalable, current, inclusive and have a very low cost. On the other hand, the absence of an authority and of a unique coherent point of view on the domain bears several limitations: the lack of hierarchy, the absence of synonym control, the lack of both precision and recall, the possibility of *gaming* [3] [4]. While the traditional classification schemes, based on taxonomies, favor searching and browsing, folksonomies encourage another paradigm of navigation, based on *finding* and *serendipity* [5].

Despite their strong limitations, folksonomies are rapidly gaining momentum: according to Clay Shirky, *"The mass amateurization of publishing means the mass amateurization of cataloging is a forced move"*[2].

As tags are just text strings, with no explicit semantics associated, it is not trivial to organize them for presentation to the user. The most common way to show a set of tags are *tag clouds*, visual representations where each tag is displayed with a font size which is proportional to its popularity. Tag clouds, however, don't keep into account relationships among tags or their meaning.

To allow the discovery of interesting and related items many applications have introduced links to *related tags*, where relatedness is generally measured with metrics based on co-occurence data. For example in del.icio.us, when a user visits the page containing all the bookmarks tagged with a certain tag, a list of tags related to that one is shown inside a sidebar.

Flickr[3], a popular folksonomy for photo sharing, introduced *clustering* as an interesting feature to help navigation in the space of a tag. The system is able to find clusters of related keywords, so items corresponding to different contexts for that tag are grouped together.

These features are very useful but often insufficient, for different reasons. First of all, they leave the lack of hierarchy problem unsolved: they build flat spaces of tags, so there is no criterion to organize them and only a small set of items can be displayed. Furthermore, there is no explicit connection with the meaning of keywords or semantic relationships among them, that might help users to orient themselves in the tag space.

An interesting study to integrate a *top down* classification paradigm with folksonomies is presented in [6]. Some investigations about the challenge to derive ontologies from folksonomies are presented in [7] and [8].

[2] http://many.corante.com/archives/2005/01/22/folksonomies_are_a_forced_move_a_response_to_liz.php

[3] http://flickr.com/

3 Our Project

The goal of our work is to investigate the possibility of integrating an ontology in the navigation interface of a folksonomy, filtering tags through a predefined semantic hierarchy to improve the possibilities of searching and browsing. In particular we chose to improve the *related tags* panel in del.icio.us; filtering a set of related tags through WordNet noun hierarchy it is possible to display a much higher number of them, organized according to a semantic criterion. As WordNet is a semantic lexicon of English, developed to reflect the semantics of natural language and the way in which humans classify objects, the relations and categories that it contains are likely to be immediately understood by most people [9].

The first problem when trying to map tags to WordNet is the one of tags that are not recognizable as words in the lexicon, even after a stemming process, and therefore cannot be mapped. To evaluate the relevance of the excluded data we have collected a large dataset, relative to about 30,000 del.icio.us users and containing about 480,000 different tags. Studying these data we found that only about 8% of the different tags used are contained in the lexicon, but we also observed that the most popular tags have a much higher probability of belonging to WordNet. This distribution in particular follows a power-law curve, very common in the field of collaborative systems, as showed in Figure 1. Of the 20 million total tagging relations present in our dataset, about 68.1% involve words contained in WordNet. We think this data might be much increased by using local wordnets in other languages and domain ontologies to cover more specific terms.

There is then the problem of words that are recognized as belonging to the lexicon, but not as nouns: these tags too can't be mapped, as the hierarchy of WordNet is only defined on nouns. According to the distinction formulated in [10] among *factual*, *subjective* and *personal* tags, we can argue that factual tags tend to correspond to nouns, as nouns fit better to describe factual knowledge, while adjectives tend to correspond to subjective tags. Further studies about this issue can be found in [11]. From a quantitative point of view, our dataset confirms the intuition that most of the tags, and especially most of the most popular tags, are nouns. Indeed the 85% of the different tags recognized by WordNet are nouns, while of the over 20 million total tagging relations, about 64.9% involve WordNet nouns, and just about 3% involve words belonging to the lexicon without being nouns; in other words this data tells that, in our dataset, about 95% of the times that a tag belonging to WordNet is used it has almost one meaning as a noun: the power law distribution is accentuated for nouns.

The application we have developed is based on a client-server paradigm, where all the tasks relative to the processing and storing of information are left to the server and the client has only to manage the visualization of results. The system architecture is shown in Figure 2.

The server is composed of a *scraper*, that extracts the data from del.icio.us HTML pages and stores them on a database, a module for *tag disambiguation* and a core module that builds the *semantic tree* of tags related to a given one,

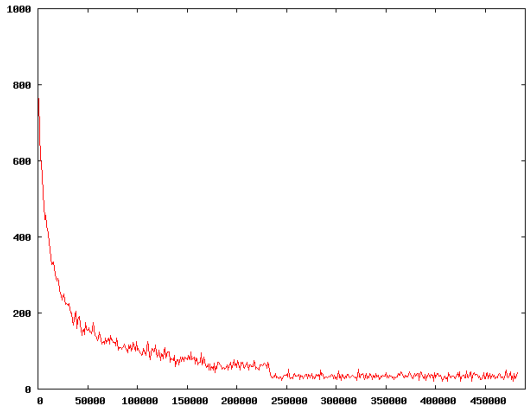


Fig. 1. The image shows the probability that a tag belongs to WordNet, in (inverse) function of its popularity. Along the X axis are represented tags from our dataset, grouped by 1000 and ordered by decreasing popularity; the Y axis shows the number of tags belonging to WordNet for each group of tags. The most popular tags are much more likely to belong to WordNet, following a power law distribution.

based on the hierarchy of concepts of WordNet. On the client side, according to the principle of *active navigation*, a JavaScript script executed inside the browser dynamically modifies the pages visualized by the user, integrating the additional information provided by the server.

3.1 Tag disambiguation

One problem when trying to map tags on an ontology is polisemy: as no explicit semantics is associated to tags by the users, the same tag can have different meanings according to different acceptation of the word, and consequently different positions in the ontology. For example the word "turkey" may refer to the country or to the animal, and in the second meaning you could want to distinguish between biological and gastronomic meaning, according to the context. In WordNet semantic relationships are not defined among words, but among *synsets*, groups of synonyms that represent units of meaning; each word can belong to different synsets according to its different acceptations. The word "turkey", for example, belongs to five synsets, where the first one is "turkey, Meleagris gallopavo" and the second is "Turkey, Republic of Turkey" .

To properly map a tag to the corresponding position in the ontology you need first to disambiguate it, in relation with the context in which it has been used. A fair solution naturally offered by a folksonomy is to use the other tags associated by some users to the same resource as the context for disambiguation.

Our algorithm for tag disambiguation acts for each tagged resource in the following way: the C most used tags for the resource are compared among them,

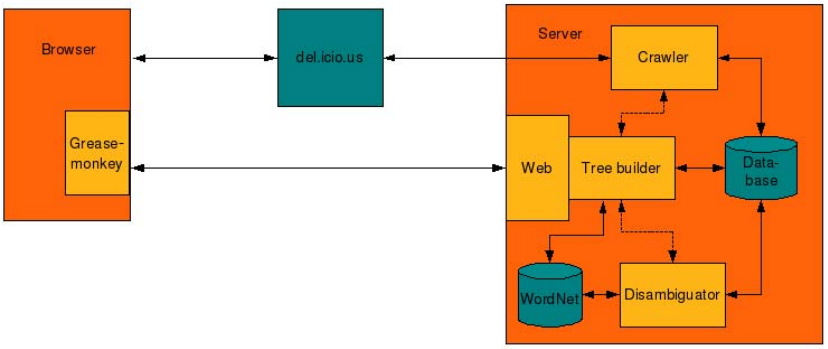


Fig. 2. The system architecture

and for each of them the meaning that is more strictly related to the other tags is selected; semantic relatedness among tags is calculated according to a choice of metrics based on WordNet [12] (adapted lesk, Hirst and St. Onge) and disambiguation is performed using the Perl library SenseRelate [13]. In the same way the remaining tags are disambiguated using the first C as a context. This solution is effective, as it reduces the sensitivity to less used tags, and efficient, as it avoids the exponential growth of the algorithm complexity with the number of different tags associated with a resource.

3.2 Building the tag semantic tree

The core module, for the construction of the tree of related tags, acts in four steps: *tree building*, *compression*, *branch sorting* and result output. All the algorithms developed have linear complexity with the number of input tags.

The set of tags to be considered is selected by collecting, for each of the latest N sites associated with the given tag, the M most frequent tags for that site; M and N are parameters that can be specified in the HTTP request. The construction of the tree is performed by an iterative algorithm; for each different tag present in the set of interest in a particular acceptation, the chain of the hypernyms is created as a path till the unique root of the noun hierarchy of WordNet and then merged with the existing tree. At the end of this process the tree is a subpart of WordNet noun hierarchy, chosen to contain all the tags of the set of interest.

As WordNet is very fine-grained, it can take more than 10 steps to descend from the root to a word; the tree has to be compressed to be useful for navigation, eliminating the useless nodes. The compression algorithm performs a breadth-first visit of the tree, in which all nodes considered unnecessary are deleted and

replaced by their children. On one hand, all the nodes corresponding to high level categories in WordNet, contained in a black list, are deleted; the information content of these nodes is generally too low to be useful for navigation. On the other hand all the nodes that do not correspond to any tag and have a branching factor lower than K or have no siblings are replaced by their children. The default value for K is 2; in this way the structure of the hierarchy is preserved and at the same time the most specific terms can ascend in the tree.

The branches are ordered by weight, where the weight of a node is calculated as the number of resources in the set of interest that have been tagged with the corresponding word in that acceptation. This guarantees that the branches of the hierarchy that are most strictly related to the given tag are shown first to the user. As a last step, the tree is output by the server in HTML or XML format.

Fig. 3. A screenshot from the del.icio.us page for tag "pasta", where the inner sidebar shows an expandable hierarchy of related tags, provided by our application.

3.3 User interface

The system rests on Firefox Browser and Greasemonkey extension to execute some JavaScript code inside the browser . When the user is visiting the del.icio.us

page for a certain tag, the script connects to our server to get the semantic tree of related keywords for that tag; as soon as the information is ready, a new sidebar is dynamically integrated in the page, showing an expandable tree. For each node of the hierarchy there are two links, one directed to the del.icio.us page for that tag and one to the page of the resources tagged both with that tag and with the given one; the size of each tag's intersection with the current keyword is shown in parenthesis and represents an indicative measure of relatedness for the users. Tooltips guide users showing WordNet definitions of the concepts corresponding to each node and indicating the destinations of links.

Figure 3 shows the result obtained for tag "pasta", where all the tags associated to the latest 300 sites tagged with "pasta" are displayed; in the picture you can see the first branches (i.e. the most related ones, in this case those about "food"), that have been expanded.

4 Tests and evaluation

We tested the system with different kinds of tags, according to different dimensions. The first dimension is the specificity of the tag from which the exploration starts; it's very different to display the space of a keyword situated in a specific domain or in a generic one. In the first case the resulting tree tends to be compact and to allow easier navigation, while in the second case it tends to have a high branching factor and a high number of first level nodes; anyway, as the branches are always ordered by weight, the most interesting concepts in relation to the given one are reachable exploring the first branches, also in case of very general keywords. The second dimension is given by the popularity of a tag, while the third one is given by the semantic field; each semantic field has its specificity and some of them rest on more conventional and ordered sets of words, such as the *food* context, visible in Figure 3, while some others are more prone to slang and neologisms, such as the one of *software*.

Figure 4 shows the result obtained for tag "blog"; as "blog" often refers to a kind of site more than to the content it can be considered a particular case, and a very general tag as there are blogs almost about everything. "Blog" is also one of the most popular tags in del.icio.us, so it is an extreme case also according to the second dimension. We obtained this result considering the latest 2000 del.icio.us bookmarks tagged "blog", and only the 15 more used tags for each of them, to cut the *long tail* of less used tags. In the picture you can see the hierarchy of scientific disciplines expanded.

According to this and other tests, the main problem for scalability seems to be the high number of nodes in the first level of the tree; some improvements could be obtained by making the tree compression algorithm more dynamic.

Comparing the related tags suggested by del.icio.us with the results we obtained, we observed that they are always somewhere in the first branches in the new sidebar. An exception must obviously be done for the words that don't belong to WordNet, that are absent in the new sidebar. Experimenting, for example, with the "Greasemonkey" tag (the experiment is possible even though

Fig. 4. A screenshot from the del.icio.us page for tag "blog", where the inner sidebar shows an expandable hierarchy of related tags, provided by our application.

the word itself is not contained in the lexicon) we found that many important related tags, like "JavaScript", are not recognized, while other important words, such as "extension", are interpreted in a wrong way as WordNet doesn't contain the acceptation related to software; all the tags for which there is in WordNet an acceptation related to software have instead been correctly interpreted by the system. These limitations could be addressed by resting on some domain ontologies to integrate WordNet and on Wikipedia for reconducting slang forms to more conventional ones (for example, Wikipedia recognizes "nyc" as an alternative form for "New York City", while WordNet does not).

In many cases synonyms or just different ways of spelling a word happen to be close to each other and easily recognizable in the tree provided by the new sidebar: the semantic hierarchy helps to face the problem of the synonym control to which a folksonomy is naturally prone.

As a last consideration we want to mention the problem of gaming. It's not unusual in del.icio.us to see the related tags sidebar entirely mucked up by spam, as we found in some of our examples. Gamers can trick del.icio.us to gain a good position for the tags they want to show and, as there are just a dozen tags suggested, the whole sidebar can easily be compromised. In the new sidebar the problem is embanked: as a much higher number of tags is shown, the presence of some spam tags doesn't make the whole suggestion system unuseful;however, the order of branches could be gamed .

5 Conclusions and Future Work

We have proposed a new approach to integrate the navigation interface of a folksonomy adding explicit semantics provided by an ontology; we have developed a tool that uses WordNet to build a semantic hierarchy that helps users navigate and find related resources in del.icio.us.

We have shown that in this way it is possible to combine some advantages of the traditional top down approach to classification with the ones of the collaborative paradigm that is emerging on the Web, providing richer possibilities of searching and browsing, and dealing with some of the limitations to which folksonomies are prone, such as lack of recall, synonym control and gaming.

Our application is actually just a prototype and can be improved in several directions. The algorithm for the tree compression is one of the most delicate issues and could be improved by making it dynamic also for higher levels of the hierarchy, instead of just eliminating words contained in a black list.

Many improvements might be reached in tag recognition by using local wordnets in different languages and domain ontologies for specific terms.

As future work, it would be also interesting to use the results of tag disambiguation, performed by our application, to filter resources and not only tags; in this way it might be possible, for example, to show, among the del.icio.us bookmarks tagged as "turkey", only the ones that have been individuated as related to the geographical acceptation.

References

1. Clay Shirky. Shirky: Ontology is overrated – categories, links, and tags, 2005. http://shirky.com/writings/ontology_overrated.html.
2. Emanuele Quintarelli. Folksonomies: power to the people. June 2005. http://www-dimat.unipv.it/biblio/isko/doc/folksonomies.htm.
3. Ellyssa Kroski. The hive mind: Folksonomies and user-based tagging, December 2005. http://infotangle.blogsome.com/2005/12/07/the-hive-mind-folksonomies-and-user-based-tagging/.
4. Harry Halpin, Valentin Robu, and Hana Shepard. The dynamics and semantics of collaborative tagging. In *Proceedings of the 1st Semantic Authoring and Annotation Workshop (SAAW'06)*, 2006.
5. Adam Mathes. Folksonomies – cooperative classification and communication through shared metadata, December 2004. http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html.
6. E. Quintarelli, L. Rosati, and A. Resmini. Facetag: Integrating bottom-up and top-down classification in a social tagging system. In *IA Summit 2007*, 2007.
7. Christoph Schmitz, Andreas Hotho, Robert Jschke, and Gerd Stumme. Mining association rules in folksonomies. In V. Batagelj, H.-H. Bock, A. Ferligoj, and A. iberna, editors, *Data Science and Classification. Proceedings of the 10th IFCS Conf.*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 261–270, Heidelberg, July 2006. Springer.
8. Celine Van Damme, Martin Hepp, and Katharina Siorpaes. Folksontology: An integrated approach for turning folksonomies into ontologies. In *Bridging the Gap between Semantic Web and Web 2.0 (SemNet 2007)*, pages 57–70, 2007.
9. C. Fellbaum. *WordNet – An Electronic Lexical Database*. MIT Press, 1998.
10. Scott Golder and Bernardo A. Huberman. The structure of collaborative tagging systems, Aug 2005. http://arxiv.org/abs/cs.DL/0508082.
11. Hend S. Al-Khalifa and Hugh C. Davis. Towards better understanding of folksonomic patterns. In *HT '07: Proceedings of the 18th conference on Hypertext and hypermedia*, pages 163–166, New York, NY, USA, 2007. ACM Press.
12. Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. Wordnet: : Similarity - measuring the relatedness of concepts. In *AAAI*, pages 1024–1025, 2004.
13. S. Patwardhan, T. Pedersen, and S. Banerjee. SenseRelate::TargetWord - A Generalized Framework for Word Sense Disambiguation. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 73–76, Ann Arbor, MI, June 2005.

Reasoning with Instances of Heterogeneous Ontologies*

Luciano Serafini and Andrei Tamilin

Data & Knowledge Management Group
Foundation Bruno Kessler - IRST
Via Sommarive 18, 38100 Povo di Trento, Italy

Abstract. We address the problem of reasoning with instances of heterogeneously formalized ontologies. Given a set of semantic mappings, reconciling conceptual and instance level heterogeneity between the input ontologies, we build our approach upon the capability of mappings to enforce a propagation of concept membership assertions between ontologies. The approach is formally grounded on a distributed description logic framework, which formally encodes ontologies as description logic knowledge bases and mappings as bridge rules and individual correspondences. We first give a logical characterization to the propagation of concept membership assertions along bridge rules and individual correspondences between the input $\mathcal{SHIQ}$-ontologies, and further define a sound and complete tableau algorithm for capturing such a propagation.

1 Motivation and Approach

Ontology *heterogeneity* is one of the crucial problems to be solved on the semantic web. To sustain this claim it is enough to give a glance on the actual situation on the web – different ontologies although representing the same or largely overlapping domains do it in different, heterogeneous ways.

The state of the art approaches to the problem of reconciling heterogeneity between ontologies are built upon the utilization of *semantic mappings*. Roughly, a mapping comprises relations between semantically related elements of different ontologies. For example, a mapping can express the fact that the concept Automobile in one ontology is semantically equivalent to the concept Car in another ontology, or that the instance ferrary_enzo in one ontology semantically corresponds to the instance f60 in the other ontology. To discover mappings a number of (semi-)automated techniques and tools can be applied; we refer the reader to the comprehensive overview of the state of the art by Euzenat and Shvaiko in [5].

Once mappings are stated, it is necessary to provide a method for *reasoning* with them. Formally, this amounts to evaluating logical consequences of mappings on the mapped ontologies. Mappings form a source to a target ontology can be used to transfer knowledge between the two ontologies. Due to the formal correspondences of ontologies to DL knowledge bases, there are two types of knowledge that can be transferred: terminological knowledge (i.e., mappings can force new concept subsumption axioms

* This is a revised version of the paper "Instance Migration over Heterogeneous Ontology Environments" accepted for presentation at ISWC2007

in the target ontology) and assertional knowledge (i.e., mappings can force new instance assertions to concepts in the target ontology).

The main objective of the paper is to provide a logical characterization of the assertional information enforced by a set of mappings and on the base of this characterization enable reasoning with individuals of mapped ontologies. Our approach relies on the logical framework of distributed description logics (DDL) introduced by Borgida and Serafini in [3]. In such a framework a *distributed knowledge base* consists of a family of standard DL knowledge bases, corresponding to each given ontology, a set of *bridge rules*, corresponding to mapping between pairs of terminologies (T-boxes), and *individual correspondences*, corresponding to mapping between pairs of elements in instance storages (A-boxes).

This work presents: (1) the logical characterization of the capability of bridge rules and individual correspondences to propagate concept membership assertions across mapped ontologies and its affection on reasoning with instances of the ontologies; (2) the overview of a sound and complete tableau algorithm for reasoning with instances of $\mathcal{SHIQ}$-ontologies, built as an extension to the classical $\mathcal{SHIQ}$-A-box tableau [10] with the backward chaining strategy for computation of propagated concept membership assertions; (3) the outline of the practical implementation of the A-box reasoning algorithm in a distributed DDL Reasoner DRAGO.

The paper is organized as follows. In Section 2 we recall a definition of DDL's distributed knowledge base with bridge rules and individual mappings. In Section 3 we investigate reasoning with instances in distributed knowledge bases; we start with analysis of knowledge propagation along bridge rules and individual correspondences and further introduce in Section 4 a tableau reasoning algorithm capturing it. We end up with an overview of related work and concluding remarks.

2 Distributed Knowledge Bases

Given a setting of multiple ontologies interconnected by directed semantic mappings, the distributed description logics allows to formally encode it in terms of a distributed knowledge base. Following the original definitions of Borgida and Serafini in [3], in this section we recall the basics of distributed knowledge bases and reasoning tasks available for them.

2.1 Syntax and Semantics

The first component of a distributed knowledge base is a family of knowledge bases $\mathcal{K} = \{\mathcal{K}_i\}_{i \in I}$. According to a standard DL definitions, each $\mathcal{K}_i$ consists of a terminological component $\mathcal{T}_i$ (T-box) and an assertional component $\mathcal{A}_i$ (A-box). Since the very same symbol can be used in two knowledge bases with different meaning, to unambiguously refer to elements of $\mathcal{K}_i$, they are prefixed with the index i of the knowledge base. The notations $i{:}\, a$ $i{:}\, C$, $i{:}\, C \sqsubseteq D$, $i{:}\, C(a)$ and $i{:}\, R(a, b)$, stand for an individual a, concept C, subsumption $C \sqsubseteq D$, assertions $C(a)$ and $R(a, b)$, respectively in the knowledge base $\mathcal{K}_i$.

Mappings from $\mathcal{K}_i$ to $\mathcal{K}_j$ ($i \neq j$) are encoded as sets of bridge rules

– $i{:}C \xrightarrow{\sqsubseteq} j{:}D$ (into-bridge rule)

– $i{:}C \xrightarrow{\sqsupseteq} j{:}D$ (onto-bridge rule)

and individual correspondences

– $i{:}a \longmapsto j{:}b$ (individual correspondence)

where C and D are concept names of $\mathcal{T}_i$ and $\mathcal{T}_j$, and a and b are individuals of $\mathcal{A}_i$ and $\mathcal{A}_j$ respectively[1].

Both bridge rules and individual correspondences from $\mathcal{K}_i$ to $\mathcal{K}_j$ express a subjective possibility of $\mathcal{K}_j$ to translate some of the concepts and individuals of $\mathcal{K}_i$ into its local concepts and individuals. For example, the following mapping between two ontologies describing the domain of cars

$$i{:}\mathsf{Transmission} \xrightarrow{\sqsubseteq} j{:}\mathsf{Gearbox} \tag{1}$$

$$i{:}\mathsf{Motor} \xrightarrow{\sqsupseteq} j{:}\mathsf{V\_Engine} \tag{2}$$

$$i{:}\mathsf{sequential\_manual\_transmission} \longmapsto j{:}\mathsf{f1\_gearbox} \tag{3}$$

can be given with the following intuitive reading: from $\mathcal{K}_j$'s point of view, i's concept Transmission is more specific than its local concept Gearbox, i's concept Motor is conversely more general than its local concept V_Engine, and finally that i's individual sequential_manual_transmission can be translated into its local individual f1_gearbox. Note that in the general case, DDL admits that an individual can have more than one translation.

A *distributed T-box* $\mathfrak{T}$ consists of T-boxes $\mathcal{T}_i$ and a collection $\mathfrak{B}$ of bridge rules between them. A *distributed A-box* $\mathfrak{A}$ consists of A-boxes $\mathcal{A}_i$ and a collection of individual correspondences $\mathfrak{C}$. A *distributed knowledge base* $\mathfrak{K}$ is then a tuple $\langle \mathfrak{T}, \mathfrak{A} \rangle$.

The semantics of DDL is defined with a fundamental assumption that each knowledge base $\mathcal{K}_i$ in the family is *locally interpreted* on its *local interpretation domain*. To support directionality, (i.e., mappings from i to j only propagate in the i-to-j-direction), we admit the hole interpretation $\mathcal{I}_\epsilon$ with empty domain (see more details in [12])[2]. By definition, we impose that $\mathcal{I}_\epsilon$ satisfies any knowledge base.

A *distributed interpretation* $\mathfrak{I}$ of a distributed knowledge base $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle$ consists of a family of local interpretations $\mathcal{I}_i$ on local interpretation domains $\Delta^{\mathcal{I}_i}$ and a family of domain relations $r_{ij} \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$ between pairs of local domains. Domain relation r_{ij} is defined to denote $\{d' \in \Delta^{\mathcal{I}_j} \mid \langle d, d' \rangle \in r_{ij}\}$.

A distributed interpretation $\mathfrak{I}$ *satisfies* a distributed knowledge base $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle$, is called a model of $\mathfrak{K}$, if all its' components are satisfied according to the following rules:

– $\mathcal{I}_i$ satisfies $\mathcal{K}_i$

[1] In this work we concentrate only on individual correspondences, and don't consider complete correspondences as introduced in [3].

[2] Classically, DL interpretation maps every individual into an *element* of the domain, while the hole maps everything into the empty *set*. To allow homogeneous treatment of standard DL interpretations and holes, we require that any individual x is standardly interpreted into a singleton set, rather than into an element of the domain. Hence, $\mathcal{I}_i \models C(a) \Longleftrightarrow a^{\mathcal{I}_i} \subseteq C^{\mathcal{I}_i}$, rather than $a^{\mathcal{I}_i} \in C^{\mathcal{I}_i}$.

– $r_{ij}(C^{\mathcal{I}_i}) \supseteq D^{\mathcal{I}_j}$ for all $i{:}C \xrightarrow{\sqsupseteq} j{:}D$
– $r_{ij}(C^{\mathcal{I}_i}) \subseteq D^{\mathcal{I}_j}$ for all $i{:}C \xrightarrow{\sqsubseteq} j{:}D$
– $b^{\mathcal{I}_j} \subseteq r_{ij}(a^{\mathcal{I}_i})$ for all $i{:}a \longmapsto j{:}b$

2.2 Distributed inference services

Although both in DL and DDL the fundamental reasoning services include checking concept subsumption and instance checking within a certain ontology, in DDL, besides the ontology itself, the other related by mappings ontologies should be taken into account. Given a distributed knowledge base $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle$, DDL defines the following *distributed inference services*:

Subsumption: A concept C is *subsumed* by a concept D in i with respect to $\mathfrak{K}$ if for every distributed interpretation $\mathfrak{I}$ of $\mathfrak{K}$ we have that $C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i}$. In this case we will write $\mathfrak{K} \models i{:}C \sqsubseteq D$.

Instantiation: An individual a is an *instance of* a concept C in i with respect to $\mathfrak{K}$ if for every distributed interpretation $\mathfrak{I}$ of $\mathfrak{K}$ we have that $a^{\mathcal{I}_i} \subseteq C^{\mathcal{I}_i}$. In this case we will write $\mathfrak{K} \models i{:}C(a)$.

Subsumption service is typically called a terminological reasoning service, while instantiation service is called assertional reasoning service. So far, the task of terminological reasoning has been undisclosed in [12]. It has been shown that certain combinations of into- and onto-bridge rules can lead to the propagation of knowledge in form of subsumption axioms across ontologies participating in DDL. Moreover, in case of DDL with $\mathcal{SHIQ}$ components without instances adding these additional propagation rules to existing DL tableaux algorithms leads to a correct and complete reasoning in DDL. In the following sections we close the gap by addressing the question of assertional reasoning in DDL.

3 Characterization of Reasoning with Instances

For the sake of clarity, we start considering the case of DDL with two component knowledge bases and unidirectional sets of bridge rules and individual correspondences. The general results and proofs can be found in the technical report [13].

3.1 Inference patterns

In the following we characterize the knowledge propagated from a knowledge base i (the source) to j (the target) by a set of *propagation rules* of the form:

$$\frac{(1)\text{ facts in } i, \quad (2)\text{ bridge rules from } i \text{ to } j, \quad (3)\text{ individual mappings from } i \text{ to } j}{(4)\text{ fact in } j}$$

which must be read as: if the facts in (1) are true in $\mathcal{K}_i$, the bridge rules in (2) are contained in $\mathfrak{B}_{ij}$, the individual correspondences in (3) are contained in $\mathfrak{C}_{ij}$, then the fact in (4) must be true in $\mathcal{K}_j$.

Following the semantics of mappings in DDL outlined in the previous section, it can be observed that the individual correspondences can interact with into-bridge rules with the effect of propagating concept membership assertions:

$$\frac{i{:}\,C(a), \quad i{:}\,C \xrightarrow{\sqsubseteq} j{:}\,D, \quad i{:}\,a \longmapsto j{:}\,b}{j{:}\,D(b)} \tag{4}$$

Because $b^{\mathcal{I}_j} \subseteq r_{ij}(a^{\mathcal{I}_i}) \subseteq r_{ij}(C^{\mathcal{I}_i}) \subseteq D^{\mathcal{I}_j}$, we indeed have that $\mathfrak{I} \models j{:}\,D(b)$.

In languages that support disjunction, the above propagation can be generalized to the propagation of concept membership assertions over a disjunction of $n \geqslant 0$ concepts:

$$\frac{i{:}\,(C_1 \sqcup \ldots \sqcup C_n)(a), \quad i{:}\,C_k \xrightarrow{\sqsubseteq} j{:}\,D_k \ (1 \leqslant k \leqslant n), \quad i{:}\,a \longmapsto j{:}\,b}{j{:}\,(D_1 \sqcup \ldots \sqcup D_n)(b)} \tag{5}$$

Rule (5) appears to be the *most general* form of assertion propagation in DDL when individual correspondences are restricted to be *functional*. A set of individual correspondences $\mathfrak{C}_{ij}$ is functional if for every individual a of $\mathcal{A}_i$ the set $\mathfrak{C}_{ij}$ contains at most one individual correspondence $i{:}\,a \longmapsto j{:}\,b$. For the sake of presentation, in this paper we restrict ourself to functional individual correspondences, leaving the most general case to the technical report [13])[3].

It is also important to note, that when $n = 0$, the inference pattern in (5) becomes the following inference rule:

$$\frac{i{:}\,\bot(a), \quad i{:}\,a \longmapsto j{:}\,b}{j{:}\,\bot(b)} \tag{6}$$

which states that to propagate the inconsistency of $\mathcal{K}_i$ to $\mathcal{K}_j$ it's enough to have one single individual correspondence. From the representational point of view this inference rule is very fragile. We currently do not see an easy solution to fix this sensitivity to inconsistency propagation. This topic will be subject for further studies.

3.2 Soundness and completeness

To demonstrate the correctness and completeness of the inference pattern presented in Section 3.1, we follow the approach similar to the one taken in [12]. The main idea consists in construction of an operator which essentially applies the generalized inference pattern (5) to extend knowledge bases with new assertions induced by mappings.

Given a set of bridge rules $\mathfrak{B}_{12}$ and set of individual correspondences $\mathfrak{C}_{12}$ from $\mathcal{K}_1$ to $\mathcal{K}_2$, the *individual correspondence operator* $\mathfrak{C}_{12}(\cdot)$, taking as input a knowledge base

[3] To give an intuition of the effect of non functional individual mappings, consider the case in which there are two into-bridge rules $i : C_1 \xrightarrow{\sqsubseteq} j : D_1$ and $i : C_2 \xrightarrow{\sqsubseteq} j : D_2$ and, the non functional set of individual mappings $\{i : a \longmapsto j : b, \ \ i : a \longmapsto j : c\}$. Then the fact that $\mathcal{K}_i \models C_1 \sqcup C_2(a)$ entails the disjunctive assertion $(D_1(b) \wedge D_1(c)) \vee (D_2(b) \wedge D_2(c))$. This implies that the general case requires technicalities for disjunctive A-boxes.

$\mathcal{K}_1$ and producing an A-box of $\mathcal{K}_2$, is defined as follows:

$$\mathfrak{C}_{12}(\mathcal{K}_1) = \left\{ (D_1 \sqcup \ldots \sqcup D_n)(b) \;\middle|\; \begin{array}{l} \mathcal{K}_1 \models (C_1 \sqcup \ldots \sqcup C_n)(a) \\ 1{:}\, C_k \xrightarrow{\sqsubseteq} 2{:}\, D_k \in \mathfrak{B}_{12}\ (1 \leqslant k \leqslant n) \\ 1{:}\, a \;\longmapsto 2{:}\, b \;\in \mathfrak{C}_{12} \end{array} \right\}$$

It is remarkable that *onto*-bridge rules do not affect instance propagation. The reason is that onto-bridge rules impose only existence of preimages of objects that already exists in the target ontology. Into-bridge rules, instead, constraint the individual mappings to be defined whithin a certain range. The individual correspondence operator formalizes the assertional knowledge that is propagated across ontologies.

The characterization of the propagation of the terminological knowledge is characterized by an analogous operator, called *bridge operator*, introduced in [12] and defined as follows: $\mathfrak{B}_{12}(\cdot)$, taking as input a knowledge base $\mathcal{K}_1$ and producing a T-box of $\mathcal{K}_2$:

$$\mathfrak{B}_{12}(\mathcal{K}_1) = \left\{ B \sqsubseteq D_1 \sqcup \ldots \sqcup D_n \;\middle|\; \begin{array}{l} \mathcal{T}_1 \models A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n \\ 1{:}\, C_k \xrightarrow{\sqsubseteq} 2{:}\, D_k \in \mathfrak{B}_{12}\ (1 \leqslant k \leqslant n) \\ 1{:}\, A \;\xrightarrow{\sqsupseteq} 2{:}\, B \;\in \mathfrak{B}_{12} \end{array} \right\}$$

With the remarkable exception of inconsistency propagation—by rule (6)—the individual correspondences do not affect the propagation of terminological knowledge. The inferences formalized by the two operators described above *completely* describe the possible propagations that are forced by a set of bridge rules and individual correspondences. This is formally stated in the following theorem.

Theorem 1 (Soundness and completeness). *Let $\mathfrak{K}_{12}$ be a distributed knowledge base consisting of $\mathcal{K}_1$, $\mathcal{K}_2$ $\mathcal{SHIQ}$ knowledge bases, and $\mathfrak{B}_{12}$, $\mathfrak{C}_{12}$ mappings between them. For any statement ϕ (of the form $C \sqsubseteq D$ or $C(a)$) in the language of $\mathcal{K}_2$*

$$\mathfrak{K}_{12} \models 2 : \phi \quad \Longleftrightarrow \quad \langle \mathcal{T}_2 \cup \mathfrak{B}_{12}(\mathcal{K}_1),\ \mathcal{A}_2 \cup \mathfrak{C}_{12}(\mathcal{K}_1) \rangle \models \phi$$

The proof of the generalization of the Theorem 1 is fully described in the technical report. Some remarks are necessary.

Independence between terminological and assertional propagation From the characterization above one can see that propagation of terminological and assertional knowledge are orthogonal. The two effects can be computed independently in parallel. What is more important, however, is that the change of the A-box does not affect the propagation of the terminological knowledge. This means that if the source T-box does not change the terminological propagation is computed once for all.

Local propagation of assertional knowledge Assertional propagation operator ensures, if a change of the source A-box involves only the set of individuals $\{a_1, \ldots, a_n\}$, then assertional propagation must be computed only for the portion of the target A-box $\mathcal{A}_2$ concerning the set of individuals $\{b \mid 1 : a_i \longmapsto 2 : b \in \mathfrak{C}_{12}\}$.

Upper bound and complexity If the mapping from 1 to 2 is finite and contains m into-bridge rules, n onto-bridge rules, and o individual correspondences, then the

bridge operator $\mathfrak{B}_{12}$ generates at most $n * 2^m$ subsumption statements, and the individual operator $\mathfrak{C}_{12}$ generates at most $o * 2^m$ instance membership statements. Since the propagation of statements needs checking subsumption and instantiation in the source knowledge base, which is EXPTIME complete, we have that computing subsumption and instantiation in a distributed setting is EXPTIME complete in the dimension of the source knowledge base plus mappings.

Vanilla implementation The above theorem supports a vanilla implementation of *forward chaining* inference engine for DDL. The implementation consists of three steps: computation of propagation operators $\mathfrak{B}_{12}(\mathcal{K}_1)$ and $\mathfrak{C}_{12}(\mathcal{K}_1)$, construction of extended version of knowledge base $\mathcal{K}_2$ as $\langle \mathcal{T}_2 \cup \mathfrak{B}_{12}(\mathcal{K}_1),\ \mathcal{A}_2 \cup \mathfrak{C}_{12}(\mathcal{K}_1)\rangle$, and finally applying to this knowledge base one of existing DL reasoners, such as FaCT++ [15], Racer [7], or Pellet [14].

The vanilla approach to reasoning has a strong advantage of reuse of existing highly optimized DL reasoners, however it can be very costly for situations when semantic mappings are changing dynamically or when the number of reasoning questions to be verified is relatively small. In the next section, we propose an alternative, *backward chaining* approach to reasoning, which does "lazy" computation of propagated axioms and hence better fits to instable and short-living distributed environments.

4 Distributed $\mathcal{SHIQ}$-A-box Tableaux Algorithm

In this section we present a distributed tableaux algorithm for reasoning with instances in DDL. Our design idea consists in constructing a network of standard DL tableaux, one for each ontology, which communicate via mappings in a backward fashion.

Since we restricted the expressivity of ontologies participating in DDL to $\mathcal{SHIQ}$ DL, we will consider in the following that ontologies $\mathcal{K}_1$ and $\mathcal{K}_2$ from a distributed knowledge base $\mathfrak{K}_{12} = \langle \mathfrak{T}_{12}, \mathfrak{A}_{12}\rangle$ are attached with $\mathcal{SHIQ}$-tableau reasoning procedures $\mathbf{Tab}_1$ and $\mathbf{Tab}_2$ [10]. Due to the reduction of reasoning with concepts to reasoning with instances [2], we suppose that each procedure $\mathbf{Tab}_i(\alpha)$ can check the satisfiability of any statement α of form $i{:}\, C \sqsubseteq D$, $i{:}\, C(a)$.

As described in [10], the $\mathcal{SHIQ}$-tableau works on a so called "completion forest", a collection of trees whose root nodes correspond to instances in A-box. Given a knowledge base, the algorithm initializes a completion forest $\mathcal{F}$ with a set of root nodes $\mathbf{x}_0 = \{x_0^k\}$ corresponding to a set of instances b_k in A-box, labels each x_0^k with a set $\mathcal{L}(x_0^k)$ of concepts C for each concept assertion $C(b_k)$ in A-box, and finally draws an edge between x_0^k and x_0^m for each role assertion $R(h_k, h_m)$ in A-box. After that, the set of $\mathcal{SHIQ}$ completion rules expanding the forest $\mathcal{F}$ is applied. The fully expanded forest then represents a model of the knowledge base. To test entailment of arbitrary assertion $X(a)$, $\neg X(a)$ is added to A-box and further the tableau is expanded to see whether a model of such knowledge base can be constructed or not.

To accommodate the knowledge propagation from $\mathcal{K}_1$ to $\mathcal{K}_2$ in $\mathfrak{K}_{12}$, we intervene in the completion process of $\mathbf{Tab}_2$ in order to capture new facts induced by bridge rules and individual correspondences. Hence, we get a *distributed tableaux procedure* $\mathbf{DTab}_2$ which extends $\mathbf{Tab}_2$ with two additional expansion rules:

$\mathfrak{C}_{12}$-rule:

if 1. $x \in \mathbf{x}_0$, such that $x = b^{\mathcal{I}_2}$ and $1{:}\, a \longmapsto 2{:}\, b$,
$\mathbf{H} \subseteq \{H_k \mid 1{:}\, B_k \xrightarrow{\sqsubseteq} 2{:}\, H_k \in \mathfrak{B}_{12}\}$,
$\mathbf{B} = \{B_k \mid H_k \in \mathbf{H}, 1{:}\, B_k \xrightarrow{\sqsubseteq} 2{:}\, H_k \in \mathfrak{B}_{12}\}$,
2. $\mathbf{Tab}_1((\bigsqcup \mathbf{B})(a)) = \mathit{true}$ for $\bigsqcup \mathbf{H} \notin \mathcal{L}(x)$,
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\bigsqcup \mathbf{H}\}$

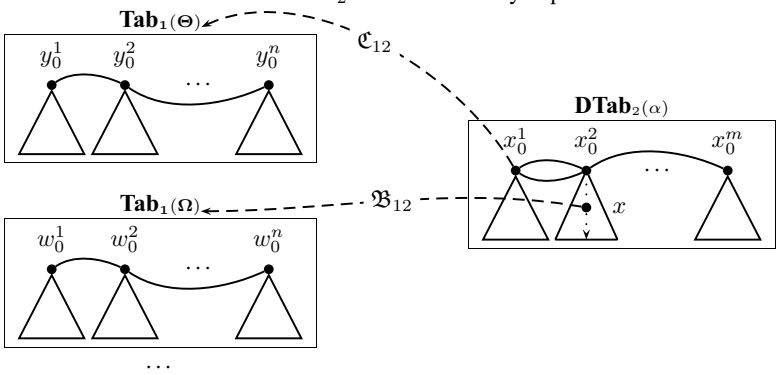
$\mathfrak{B}_{12}$-rule:

if 1. $G \in \mathcal{L}(x)$, such that $1{:}\, A \xrightarrow{\sqsupseteq} 2{:}\, G \in \mathfrak{B}_{12}$,
$\mathbf{H} \subseteq \{H_k \mid 1{:}\, B_k \xrightarrow{\sqsubseteq} 2{:}\, H_k \in \mathfrak{B}_{12}\}$,
$\mathbf{B} = \{B_k \mid H_k \in \mathbf{H}, 1{:}\, B_k \xrightarrow{\sqsubseteq} 2{:}\, H_k \in \mathfrak{B}_{12}\}$,
2. $\mathbf{Tab}_1(A \sqsubseteq \bigsqcup \mathbf{B}) = \mathit{true}$ for $\bigsqcup \mathbf{H} \notin \mathcal{L}(x)$,
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\bigsqcup \mathbf{H}\}$

The principle idea of these additional expansion rules consists in implementing backward versions of bridge and individual correspondences operators introduced in Section 3.2. According to rule $\mathfrak{C}_{12}$, if **DTab**$_2$ encounters a root node x connected by an individual correspondence, then a disjunction of concepts $\bigsqcup \mathbf{H}$ should be added to the label $\mathcal{L}(x)$ if $\bigsqcup \mathbf{H}(x)$ is entailed by interaction of individual correspondence with into-rules. To determine this entailment, **DTab**$_2$ remotely requests foreign **Tab**$_1$ to check if it is the case that $\bigsqcup \mathbf{B}(b)$ in $\mathcal{K}_1$.

The role of $\mathfrak{B}_{12}$-rule is to analyse the nodes of completion forest and import consequences of subsumption propagations. If **DTab**$_2$ encounters a node x which contains a label G connected by an onto-bridge rule, then if $G \sqsubseteq \bigsqcup \mathbf{H}$ is entailed by the bridge rules, the label $\bigsqcup \mathbf{H}$ is added to x. While in order to determine the entailment, **DTab**$_2$ invokes the procedure **Tab**$_1$ with a question whether a subsumption $A \sqsubseteq \bigsqcup \mathbf{B}$ holds in $\mathcal{K}_1$.

The distributed execution of **DTab**$_2$ can be intuitively depicted as follows:



Theorem 2 (Termination, Soundness, Completeness). *Given $\mathcal{SHIQ}$ DL knowledge bases $\mathcal{K}_1$ and $\mathcal{K}_2$, let $\mathfrak{K}_{12} = \langle\langle\{\mathcal{T}_1, \mathcal{T}_2\}, \mathfrak{B}_{12}\rangle, \langle\{\mathcal{A}_1, \mathcal{A}_2\}, \mathfrak{C}_{12}\rangle\rangle$ be a distributed knowledge base. Then, given a $\mathcal{SHIQ}$ statement α*

1. *a distributed procedure* **DTab**$_2(\alpha)$ *terminates, and*

2. *α is satisfiable in $\mathcal{K}_2$ with respect to $\mathfrak{R}_{12}$ if and only if* $\mathbf{DTab}_2(\alpha)$ *yields a complete and clash-free completion forest.*

It can be shown that the proposed algorithm enjoys generalization to arbitrary number of $\mathcal{SHIQ}$ knowledge bases participating in DDL, and moreover can be extended to distributed knowledge bases containing cyclical pathes of bridge rules and individual correspondences. For the sake of clarity, we omit the discussion of these generalizations and refer the reader to the technical report [13] for details.

Note that due to the remark to Theorem 1 on independence of terminological and assertional propagation, the implementation of the tableaux introduced in this section can be constructed on top of existing implementation of DRAGO DDL Reasoner by reusing the implementation of bridge completion rule and adding additionally the individual completion rule as described in the present algorithm.

5 Related Work

The importance of resolving heterogeneity problem on the web pushes the big research efforts to devising frameworks capable of representing and reasoning with multiple ontologies interrelated by semantic mappings. While DL is already the standard for working with web ontologies, the question of formal representations and reasoning with mappings is still a subject to the standardization.

In *SomeWhere* [6], the authors target a question of decentralized approach to querying heterogeneous ontologies. Mappings in *SomeWhere* has a form of a subsumption statements and the reasoning is based on rewriting techniques for combining reasoning over heterogeneous ontologies. The big advantage of the presented approach is its scalability, while the disadvantage is its limitation to a "propositional" ontologies, containing only disjunction, conjunction and negation.

Another recent example of decentralized infrastructure for querying distributed ontologies is *KAONp2p* [8, 9]. The authors adopt the approach of [4] to express mappings as correspondences between conjunctive queries over ontologies. The querying further requires the terminologies and mapping to be merged into a single global ontology, while instance data is then retrieved from distributed instance storages.

The recent study of query answering in *distributed description logics* has been proposed in [1]. The main idea consist in constructing a closure ontology by forward propagating, via DDL mappings, relevant axioms contained in other mapped ontologies (in a vein of vanilla implementation of DDL reasoner discussed in the current study). Doing so, further enables reformulation of distributed query answering problem into local query answering. Although the approach of [1] is sound, the authors point out the incompleteness of their study.

Another important framework is *$\mathcal{E}$-connections* [11]. Original purpose of $\mathcal{E}$-connections is to aggregate ontologies that model different (non-overlapping) aspects of the world, rather then integrate those overlapping as in DDL. Nonetheless, it has been shown in [11] that mathematically DDL constructs can be simulated in $\mathcal{E}$-connections, however sacrificing the directionality of knowledge propagation. Another difference concerns with reasoning approach. In contrast to distributed coordinating tableaux in DDL, in $\mathcal{E}$-connections a global tableau, both theoretically and practically, needs to be constructed.

6 Conclusion

In the present study, we investigated a task of correct and complete reasoning with instances over heterogeneous ontologies. We formally grounded our approach on DDL framework. Theoretically, we formalized inferences with instances and defined the distributed tableaux algorithm for reasoning with multiple $\mathcal{SHIQ}$ DL ontologies. Practically, we extended terminological reasoning services available in the DRAGO DDL Reasoner with the support of assertional reasoning tasks.

References

1. F. Alkhateeb and A. Zimmermann. Query Answering in Distributed Description Logics. In *Proc. of the 1st Conference on New Technologies, Mobility and Security (NTMS)*, 2007.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. 2003.
3. A. Borgida and L. Serafini. Distributed Description Logics: Assimilating Information from Peer Sources. *Journal of Data Semantics*, 1:153–184, 2003.
4. D. Calvanese, G. De Giacomo, and M. Lenzerini. A Framework for Ontology Integration. In *Proc. of the Semantic Web Working Symposium (SWWS-2001)*, pages 303–316, 2001.
5. J. Euzenat and P. Shvaiko, editors. *Ontology Matching*. Springer Verlag, 2007.
6. F. Goasdoué and M-C. Rousset. Querying Distributed Data through Distributed Ontologies: a Simple but Scalable Approach. *IEEE Intelligent Systems*, 18(5):60–65, 2003.
7. V. Haarslev and R. Moller. RACER System Description. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR-2001)*, pages 701–706, 2001.
8. P. Haase and B. Motik. A Mapping System for the Integration of OWL-DL Ontologies. In *Proceedings of the First International Workshop on Interoperability of Heterogeneous Information Systems (IHIS 05)*, pages 9–16. ACM Press, 2005.
9. P. Haase and Y. Wang. A Decentralized Infrastructure for Query Answering over Distributed Ontologies. In *Proceedings of the 22nd Annual ACM Symposium on Applied Computing (SAC-2007)*, 2007.
10. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with Individuals for the Description Logic SHIQ. In *Proceedings of the 17th International Conference on Automated Deduction (CADE-2000)*, pages 482–496, 2000.
11. O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-Connections of Abstract Description Systems. *Artificial Intelligence*, 156(1):1–73, 2004.
12. L. Serafini, A. Borgida, and A. Tamilin. Aspects of Distributed and Modular Ontology Reasoning. In *Proc. of the 19th Joint Conference on Artificial Intelligence (IJCAI)*, 2005.
13. L. Serafini and A. Tamilin. Reasoning with Instances in Distributed Description Logics. Technical report, Fondazione Bruno Kessler - IRST, 2007. `http://sra.itc.it/people/tamilin/publications/2007/swap/tr.pdf`.
14. E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A Practical OWL-DL Reasoner. *Journal of Web Semantics*, 2006.
15. D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130, pages 292–297, 2006.

Some experiments on the usage of a deductive database for RDFS querying and reasoning

Giovambattista Ianni[1,2], Alessandra Martello[1],
Claudio Panetta[1], and Giorgio Terracina[1]

[1] Dipartimento di Matematica, Università della Calabria,
I-87036 Rende (CS), Italy,
[2] Institut für Informationssysteme 184/3, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{ianni,a.martello,panetta,terracina}@mat.unical.it

Abstract. Ontologies are pervading many areas of knowledge representation and management. To date, most research efforts have been spent on the development of sufficiently expressive languages for the representation and querying of ontologies; however, querying efficiency has received attention only recently, especially for ontologies referring to large amounts of data. In fact, it is still uncertain how reasoning tasks will scale when applied on massive amounts of data. This work is a first step toward this setting: based on a previous result showing that the SPARQL query language can be mapped to a Datalog, we show how efficient querying of big ontologies can be accomplished with a database oriented extension of the well known system DLV, recently developed. We report our initial results and we discuss about benefits of possible alternative data structures for representing RDF graphs in our architecture.

1 Introduction

The *Semantic Web* [4, 11] is an extension of the current Web by standards and technologies that help machines to understand the information on the Web so that they can support richer discovery, data integration, navigation, and automation of tasks. Roughly, the main ideas behind the Semantic Web aim to *(i)* add a machine-readable meaning to Web pages, *(ii)* use ontologies for a precise definition of shared terms in Web resources, *(iii)* make use of KR technology for automated reasoning on Web resources, and *(iv)* apply cooperative agent technology for processing the information of the Web. The development of the Semantic Web proceeds in layers of Web technologies and standards, where every layer stays on top of lower layers.

There is currently much research work on the three consecutive RDF(S), Ontology and Rule (listed from bottom to top) layers. The RDF(S) layer was initially conceived as a basic framework for defining resources available on the Web and their connections. In this vision, RDF(S) should have little or no semantics, and focuses only on the logical format of information, which is based on an encoding of data as a labeled graph (or equivalently, a ternary relation, commonly called RDF *triplestore* or RDF *graph*).

The Ontology layer should be built on top of RDF(S) and should provide the necessary infrastructure for describing knowledge about resources. An ontology can be written using one of the three official flavors of the OWL language [18], currently accepted as a W3C Standard Recommendation. An OWL knowledge base is written in RDF(S), where some of the keywords of the language are now given additional semantics.

OWL is based on decidable flavors of description logics and features rich expressiveness which, unfortunately, introduces high computational costs for many of the reasoning tasks commonly performed over an ontology. Nonetheless, a variety of Web applications require highly scalable processing of data. This puts the focus back to the lower RDF(S) data layer. In this context, RDF(S) should play the role of lightweight ontology language. Indeed, RDF(S) has few and simple descriptive capabilities (mainly, the possibility to describe and reason over monotonic taxonomies of objects). One can thus expect from RDF(S) query systems the ability of querying very large datasets with excellent performance, yet allowing limited reasoning capabilities on the same data.

As a candidate W3C recommendation [8], the SPARQL language is reaching consensus as query language of election for RDF(S) data. In this scenario, an RDF(S) triplestore plays the role of a database, but, as an important difference, a triplestore might contain information not explicitly stored, obtainable by logical inference. Allowed logical inference rules are given by the official RDF(S) semantics specification, whereas SPARQL plays the role of query language.

Although SPARQL-enabled triplestores are many [3, 20, 2, 21] their scalability or querying capabilities are still far from maturity, having one or more of the following drawbacks:

– RDF(S) semantics is implemented by materializing all the inferred data a priori. This latter option can not be adopted in practice if massive amount of data are involved in the inferencing process, since inferred information is usually much bigger in size than explicit information.

– The basic reasoning machinery of RDF(S) prescribes heavy usage of transitive closure (recursive) constructs. Roughly speaking, given a class taxonomy, an individual belonging to a leaf class must be inferred to be member of all the ancestor classes, up to the root class. This prevents a straightforward implementation of RDF(S) over RDBMSs, since RDBMSs usually feature very primitive, inefficient implementations of recursion in their native query languages.

But, interestingly, in Datalog, recursion is a first class citizen. Also, most of the SPARQL features can be mapped to a rule based language with stable model semantics [19]. Intuitively, a large fragment of the RDF(S) semantics can thus be implemented by means of a translation to an equivalent Datalog program.

Thus, one may think to adopt a Datalog based language language for implementing RDF(S). Value invention constructs, as those introduced in [6] (where it is defined a form of Answer Set Programming with external predicates and value invention), allow, in practice, the manipulation of infinite universes of individuals (as in the RDF(S) scenario) in a finite model setting.

Many important efforts in the Semantic Web community aim to integrate Ontologies with Rules under stable model semantics (e.g. [9, 16]), considering

both OWL and RDF(S). In this context, the possibility to exploit a Datalog-like language to express both the ontology and the query/rule language would provide important benefits.

However, it is well known in the research community that current (extended) Datalog based systems present important limitations when the amount of data to reason about is large; in fact: *(i)* reasoning is generally carried out in main-memory and, hence, the quantity of data that can be handled simultaneously is limited; *(ii)* the interaction with external (and independent) DBMSs is not trivial and, in several cases, not allowed at all, but in order to effectively share and elaborate large ontologies these must be handled with some database technology; *(iii)* the efficiency of present datalog evaluators is still not sufficient for their utilization in complex reasoning tasks involving large amounts of data.

In the following we refer to a recently proposed database-oriented extension of the well known Answer Set Programming system DLV, named DLV^{DB} [22], which presents the features of a Deductive Database System (DDS) and can do all the reasoning tasks directly in mass-memory; DLV^{DB} does not have, in principle, any practical limitation in the dimension of input data, is capable of exploiting optimization techniques both from the DBMS field (e.g., join ordering techniques [12]) and from the DDS theory (e.g., magic sets [17]), and can easily interact (via ODBC) with external DBMSs.

DLV^{DB} turned out to be particularly effective for reasoning about massive data sets (see benchmark results presented in [22]) and supports a rich query and reasoning language including stratified recursion, true negation, negation as failure, and all built-in and aggregate functions already introduced in DLV [10]. As a consequence, DLV^{DB} seems to be a good candidate also as an ontology querying engine.

To accomplish this goal, several building bricks are missing: *(i)* a mapping from RDF(S) semantics to Datalog; *(ii)* the translation of SPARQL queries in Datalog; *(iii)* the connection of massive RDF(S) data to a suitable system such as DLV^{DB}; *(iv)* the evaluation of queries directly on a given triplestore using DLV^{DB}.

The present paper concentrates on points (iii) and *(iv)*. About point *(i)*, *(ii)* and *(iii)* the reader may refer to [13],[5] and [19]. In particular, it aims to represent a first step toward the reconciliation of expressiveness with scalability for ontology querying, by means of deductive database technology.

The paper is organized as follows. In the next Section we briefly introduce the main peculiarities of the DLV^{DB} system. The Section 3 is devoted to present our experimental results, whereas in the section 4 we discuss about alternative data structure better suited to handling RDF data. Finally, in Section 5 we draw some conclusions.

2 DLV^{DB}

DLV^{DB} [22] is an extension of the well known ASP system DLV [14] designed both to handle input and output data distributed on several databases, and to allow the evaluation of logic programs directly on databases. It combines the expressive power of DLV with the efficient data management features of DBMSs [12].

The detailed description of DLV^{DB} is out of the scope of the present paper; here we briefly outline the main peculiarities which make it a suitable Datalog-based ontology querying engine. The interested reader can find a complete description of DLV^{DB} and its functionalities in [22].
The system, along with documentation and some examples, are available for download at `http://www.mat.unical.it/terracina/dlvdb`.

Generally speaking, DLV^{DB} allows for two typologies of execution: *(i)* direct database execution, which evaluates logic programs directly on database, with a very limited usage of main-memory but with some limitations on the expressiveness of the queries, and *(ii)* main-memory execution, which loads input data from different (possibly distributed) databases and executes the logic program directly in main-memory. In both cases, interoperation with databases is provided by ODBC connections; these allow handling, in a quite simple way, data residing on various databases over the network.

For the purposes of this paper, it is particularly relevant the application of DLV^{DB} in the direct database execution modality for the querying of large ontologies. In fact, usually, the user has his data stored in (possibly distributed) triplestores and wants to carry out some reasoning on them; however the amount of such data can be such that the evaluation of the query can not be carried out in main-memory. Then, it must be evaluated directly in mass-memory.

Moreover, DLV^{DB} turned out to be particularly effective for reasoning about massive data sets (see benchmark results presented in [22]) and supports a sufficiently rich reasoning language for querying ontologies (see also Section 3).

Three main features characterize the DLV^{DB} system in the direct database execution modality: *(i)* its ability to evaluate logic programs directly and completely on databases with a very limited usage of main-memory resources, *(ii)* its capability to map program predicates to (possibly complex and distributed) database views, and *(iii)* the possibility to easily specify which data is to be considered as input or as output for the program. In the application context considered in this paper, these characteristics allow the user to have a wide flexibility in querying available ontologies.

In order to properly carry out the evaluation, the system needs to know the mappings between input/output data and program predicates, as well as whether the temporary relations possibly needed for the mass-memory evaluation should be maintained or deleted at the end of the execution. The user can specify this information by some auxiliary directives which must be fed to the system beside the logic program.

3 Experiments

In this section we present the results of our experiments aiming at comparing the performance of DLV^{DB} with several state-of-the-art triplestore. The main goal of our experiments was to evaluate both the scalability and the the query language expressiveness of the tested systems. All tests have been carried out on a Pentium 4 machine with a 3.00 GHz CPU and 1.5 Gbytes of RAM.

3.1 Compared Systems

In our tests we compared DLV^{DB} with three state-of-the-art triplestores, namely: Sesame, ARQ, and Mulgara. The first two systems allow both in-memory and RDBMS storage and, consequently, we tested them on both execution modalities. In the following we shall refer the in-memory version of Sesame (resp., ARQ) as Sesame-Mem (resp. ARQ-Mem) and the RDBMS version as Sesame-DB (resp. ARQ-DB). For each system we used the latest official available release. We next briefly describe them.

Sesame [20] is an open source Java framework with support for storage and querying of RDF(S) data. It offers to developers a flexible access API and several query languages; however, its native language (which is the one adopted in our tests) is SeRQL – Sesame RDF Query Language. In fact, the current stable release of Sesame does not support the SPARQL language yet. Some of the query language's most important features are: *(i)* expressive path expression syntax that match specific paths through an RDF graph, *(ii)* RDF Schema support, *(iii)* string matching. Furthermore, it allows simplified forms of reasoning on RDF and RDFS. In particular, inferences are performed by pre-computing the closure $R(G)$ of the input triplestore G. The latest official release currently available is the version 1.2.7.

ARQ [3] is a query engine implementing SPARQL under the Jena framework[3]. ARQ includes a rule-based inference engine and performs non materialized inference. As for Sesame, ARQ can be executed with data loaded both in-memory and on a RDBMS. We executed SPARQL queries from Java code using the Jena's API (version 2.5) in both execution modalities.

Mulgara [2] is a database system specifically conceived for the storage and retrieval of RDF(S). Mulgara is an Open Source active fork of the Kowari project[4]. The adopted query language is *iTQL* (Interactive Tucana Query Language), a simple SQL-like query language for querying and updating Mulgara databases. A compatibility support with SPARQL is declared, yet not implemented. The Mulgara Store offers native RDF(S) support, multiple databases (models) per server, and full text search functionality. The system has been tested using its internal storage data structures (XA Triplestore). The latest release available for Mulgara is mulgara-1.0.0.

3.2 Benchmark Data Set

We adopted as reference benchmark data the DBLP database [15]. DBLP contains a large number of bibliographic descriptions on major computer science journals and proceedings; the server indexes more than half a million articles and several thousand links to home pages of computer scientists. Recently, an OWL ontology has been developed for DBLP data and the corresponding RDF can be downloaded at the web address `http://sw.deri.org/~aharth/2004/07/dblp/`. The main classes represented in this ontology are *Author*, *Citation*, *Document*, and *Publisher*, where a *Document* can be one of: Article, Book, Collection, Inproceedings, Mastersthesis, Phdthesis, Proceedings, Series, WWW.

[3] `http://jena.sourceforge.net`
[4] `http://www.kowari.org/`

In order to test the scalability of the various systems we considered several subsets of the entire database, each containing an increasing number of statements and constructed in such a way that the greater sets strictly contain the smaller ones. Generated data sets contain from 50000 to 2000000 RDF statements[5].

3.3 Tested Queries

As previously pointed out, the expressiveness of the query language varies for each tested system. In order to compare both scalability and expressiveness, we designed for kind of queries of increasing complexity, ranging from simple selections to queries requiring different forms of inferences over the data.

In more detail, we selected the following for queries which will be referred to as Q_1, Q_2, Q_3 and Q_4, respectively.

– Q_1: Select the names of the Authors and the URI of the corresponding Articles they are author of;

– Q_2: Select the names of the Authors which published at least one Article in year 2000;

– Q_3: Select the names of the Authors which are creators of at least one document (i.e. either an Article, or a Book, or a Collection, etc.);

– Q_4: For each Author in the database, select the corresponding name and count the number of Articles he published.

Here, queries Q_1 and Q_2 are simple selections; Q_3 requires a simple form of inference; in fact articles, books, etc. must be abstracted into documents. Query Q_4 requires the capability to aggregate data, which is not provided by all query languages.

It is worth observing that queries Q_1, Q_2, and Q_3 can be executed by all the evaluated systems. As for Q_3, we exploited the Krule engine for Mulgara, the inferencing repository in Sesame and the inferencing Reasoner in ARQ. Note that Sesame-DB materializes the possible inferenced data just during the loading of the RDF dataset in the database; however, in our tests, we measured only the query answering time for it. Query Q_4 can not be evaluated neither by ARQ nor by Sesame because both SPARQL and SeRQL query languages do not support aggregate operators.

Due to space constraints, we can not show here the details of all the queries. Just to show an example, we next present the encodings used for Q_1 in the various systems. Syntax is self-intuitive.

DLV^{DB} encoding for Q_1

$$q_1(NAME, RES) :- \; triple(RES, \text{“}rdf{:}type\text{”}, \text{“}Article\text{”}),\; triple(RES, \text{“}dc{:}creator\text{”}, PERS),\; triple(PERS, \text{“}foaf{:}name\text{”}, NAME).$$

[5] An RDF statement is a small cluster of RDF triples usually not larger than 10 within our datasets.

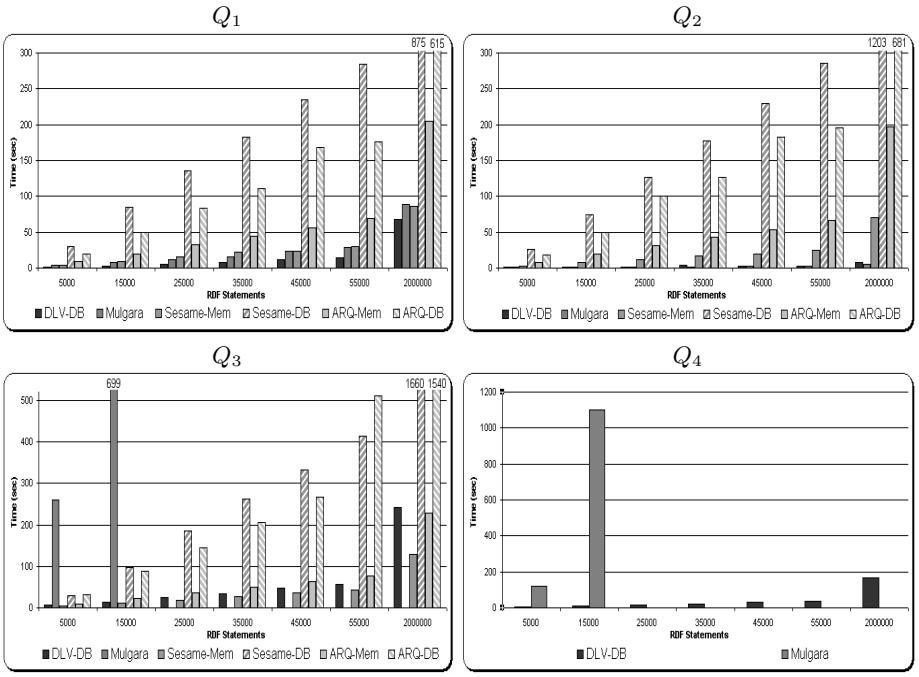


Fig. 1. Results for queries Q_1, Q_2, Q_3, Q_4

Sesame encoding for Q_1

select *name*, *res*
from {*res*} <*rdf:type*> {*type*},
{*res*} <*dc:creator*> {*pers*}, {*pers*} <*foaf:name*> {*name*},
where *type* =<*Article*>

ARQ encoding for Q_1 (SPARQL syntax)

select *?name ?res*
where {*?res* <*rdf:type*><*Article*> .
?res <*dc:creator*> *?pers*. *?pers* <*foaf:name*> *?name*}

Mulgara encoding for Q_1

select *$name*, *$res*
from <*rmi://localhost/server1#triple*>
where *$res* <*rdf:type*><*Article*> **and**
$res <*dc:creator*> *$pers* **and** *$pers* <*foaf:name*> *$name*;

3.4 Results and Discussion

Figure 1 shows the results we have obtained for the five queries described above. In the figure, the chart of a system is absent whenever it has not been able to solve the query due to some system's fault or if its response time was greater than 3600 seconds (1 hour). Moreover, if a system's query language was not sufficiently expressive to answer a certain query, it has not been included in the graph. From the analysis of the figure we can draw the following observations.

Mulgara has, after DLV^{DB}, the more expressive query language and, for the simple queries Q_1 and Q_2 the best performance along DLV^{DB} and, in some

cases, Sesame-Mem. However, when the queries involve the more advanced parts of the language, the efficiency of Mulgara quickly drops; in fact, both in query Q_3 and in query Q_4 its response time exceeded the limit set in our tests already after 15000 RDF statements.

Sesame-Mem turned out to be competitive in all considered data sets only for queries Q_1 and Q_3; in fact, it has not been able to solve query Q_4 due to lack of expressiveness in the query language; moreover, in query Q_2, its performance degraded when the input data sets increased. Sesame-DB always had significantly worse performance than Sesame-Mem.

ARQ always presented the worst performances (except in one case); moreover, as occurred for Sesame, also the database version of ARQ revealed worse performance than its in-memory version. The expressiveness of ARQ's query language prevented to encode queries Q_4.

Finally, DLV^{DB} revealed both the best performance (in almost all the data sets and queries) and the highest expressiveness of the query language, thus demonstrating its good potential to be exploited as ontology querying engine.

It is worth pointing out that both Sesame and ARQ performance are negatively influenced by the usage of a DB (see, in particular, results of queries Q_1 and Q_2); this can be probably motivated by the fact that they carry out (at least parts of) their computations in main memory anyway and, consequently, transferring data from disk to memory produces just overhead. On the contrary, DLV^{DB} and Mulgara exploit the database technology directly for their reasoning tasks and, consequently, are more effective.

4 Experiments with alternative data structures

In the context of real-world applications it becomes crucial the choice of a data schema for the relational database handling RDF data model, since this has a direct impact on the performance and scalability issues. The discussed solution assumes to store the RDF(s) graph at hand, using a straightforward representation, where a single 3-columns table contains one row for each statement of the form $\langle subject, predicate, object \rangle$. This representation, though flexible, is not efficient when several self-joins are required to sweep over this single large table. A first step in order to improve the performance of the database, maintaining this simple schema, is to reduce the execution time required by the join's operations. A solution largely adopted in similar applications and discussed in [1], is to avoid to store explicitly string values referring to URIs and literals in the main table, replacing them with an hash value. Indeed, integer matching is intuitively much faster than string matching. Each URIs/literal string is mapped to and integer: the main tables stores triples in form of integer values, while additional tables store the association from URI/Literals to integer, which is used for a post-normalization. Several experiments that we reproduced on this new configuration show the validity of this approach with respect to the first one.
Finally, we have considered other suggestions for alternative data structures better suited for handling RDF data, called property tables technique ([1],[23]). These aim at denormalizing RDF tables by storing them in a flattened representation, trying to encode triples according to the hidden "schema" of RDF data,

similarly to a traditional relational schemas. The idea is to define a set of property tables containing (cluster of) properties that tend to be defined together (and then storing the triples from the RDF dataset whose properties belong to the selected attributes), or to cluster similar sets of subjects together, grouping them in a property-class table. There is a variety of storage schemes and several variations of these which have also been implemented in existing RDF stores, using hybrid representation that combine features of both. The most important advantage of these choices is the possibility of accessing directly all the triples having the same property value. However, these configurations can be extremely sparse (by the presence of NULL values in the table) and not well suited for supporting multi-valued attributes. Thus, while such techniques usually improve performance of queries involving a single property table, it is required to properly cluster the property values occurring in the dataset.
Inspired by these considerations we extended our representation schema implementing a fully decomposed storage model in which the triples table is rewritten into n two column tables where n is the number of unique properties in the dataset. This approach (discussed in [1], [23]) support succinct representation of multi-valued attributes and heterogeneous records (subjects not defining a particular property). Moreover, this data scheme allows to access directly assertions related to the same property value. Unfortunately, for a query which quantifies over property values, several tables have to be merged. This overhead seems reasonable (as we verified testing performance on query ranging on variable predicates). Furthermore, insert and update operations can be slower, since for operation on statements related to the same subject, more tables need to be accessed.
We carried out several experiments on these new data structures to compare the execution time of the queries for the same dataset used in previous test. Clearly, this implies the translation of queries to queries over the new representations. The results obtained shows that the solution using triples table storing identifiers instead of strings performs better than the simple one, but the best choice (actually) is to use a fully decomposed storage schema. Especially, this seems to give more benefit as the number of triples grows. For example, the query Q_1 (run on the biggest dataset) takes 74 seconds running on the single table representation, 46 seconds running using hash representation of URIs/literals and 28 seconds running on the fully decomposed schema configuration with hash representation.

5 Conclusions

In this paper we presented a first step toward efficient and reliable ASP-based querying of ontologies. We experimentally proven that our solution, based on a database oriented implementation of ASP, improves both scalability and expressivity of several state-of-the-art systems. Although, currently, RDF data are stored in the standard triple format, the first experiments with alternative data structures are very promising. The representation of data in some more structured form (as already some of the tested systems do) could significantly improve performance. Another promising research line consists in using database integration techniques in the ontology context such as in [7].

References

1. D. J. Abadi, A.Marcus, S. Madden, and K. J. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, pages 411–422, 2007.
2. T. Adams, G. Noble, P. Gearon, and D. Wood. MULGARA homepage. http://www.mulgara.org/ , since 2006.
3. ARQ homepage. http://jena.sourceforge.net/ARQ/, since 2004.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
5. J. de Bruijn and S. Heymans. RDF and logic: Reasoning and extension. In *Proceedings of the 6th WebS, in conjunction with the 18th DEXA*, Regensburg, Germany, September 3–7 2007.
6. F. Calimeri, S. Cozza, and G. Ianni. External sources of knowledge and value invention in logic programming. *Ann. Math. Artif. Intell.*, 50(3-4):333–361, 2007.
7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati. Efficient integration of relational data through dl ontologies. CEUR Electronic Workshop Proceedings, 2007.
8. A. Seaborne E. Prud'hommeaux. Sparql query language for rdf. w3c candidate recommendation, 14 june 2007. `http://www.w3.org/tr/rdf-sparql-query/`.
9. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In *(IJCAI) 2005*, pages 90–96, Edinburgh, UK, August 2005.
10. W. Faber and G. Pfeifer. `DLV` homepage, since 1996. http://www.dlvsystem.com/.
11. D. Fensel, W. Wahlster, H. Lieberman, and J. Hendler, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2002.
12. H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation*. Prentice Hall, 2000.
13. G. Ianni, A. Martello, C. Panetta, and G. Terracina. Faithful and effective querying of RDF ontologies using DLVDB.
14. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, July 2006.
15. Michael Ley. Digital bibliography and library project http://dblp.uni-trier.de/.
16. B. Motik and R. Rosati. A faithful integration of description logics with logic programming. In *IJCAI*, pages 477–482, 2007.
17. I.S. Mumick, S.J. Finkelstein, H. Pirahesh, and R. Ramakrishnan. Magic conditions. *ACM Trans. Database Systems*, 21(1):107–155, 1996.
18. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. Owl web ontology language semantics and abstract syntax. w3c recommendation, 10 february 2004.
19. A. Polleres. From sparql to rules (and back). In *In Proceedings of the 16th World Wide Web Conference (WWW2007), Banff, Canada*, 2007. Extended technical report version available at `http://www.polleres.net/publications/GIA-TR-2006-11-28.pdf`.
20. SESAME homepage. http://www.openrdf.org/, since 2002.
21. Sparql implementations. `http://esw.w3.org/topic/sparqlimplementations`.
22. G. Terracina, N. Leone, V. Lio, and C. Panetta. Experimenting with recursive queries in database and logic programming systems. *Theory and Practice of Logic Programming (TPLP). Available on-line at http://arxiv.org/abs/0704.3157*, 2007. Forthcoming.
23. Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking database representations of rdf/s stores. In *International Semantic Web Conference*, pages 685–701, 2005.

Who the FOAF knows Alice? RDF Revocation in DBin 2.0*

Christian Morbidoni[2], Axel Polleres[1], and Giovanni Tummarello[1]

[1] DERI Galway, National University of Ireland, Galway
{firstname.lastname}@deri.org
[2] SeMedia Group, Universita' Politecnica delle Marche, Ancona, Italy
christian@deit.univpm.it

Abstract. In this paper we take a view from the bottom to RDF(S) reasoning. We discuss some issues and requirements on reasoning towards effectively building Semantic Web Pipes, aggregating and patching RDF data from various distributed sources. Even if we leave out complex description logics reasoning and restrict ourselves to the RDF world, it turns out that some problems, in particular how to deal with contradicting RDF statements and patching RDF graphs, do not yet find their proper solutions within the current Semantic Web Stack. Besides theoretical solutions which involve full DL reasoning, we believe that more practical and probably more scalable solutions are conceivable one of which we discuss in this paper. Namely, we provide means to express revocations in RDF and resolve such revocations by means of a specialized RDF merge procedure. We have implemented this conflict-resolving merge procedure in the DBin 2.0 system.

1 Introduction

Publishing RDF files on the Web is bound to become more and more a way to state facts that are asserted or believed to be true by the producer of the source itself. DBpedia [1], for example, publishes a large collection of such facts by extracting them from the collective works of the Wikipedia communities. FOAF [5] files are personal RDF models which are created by individuals to state facts about, typically, themselves. Nothing, however, prevents them in general to state facts about other entities and this is in fact a fundamental feature of the "Semantic Web", everyone is allowed to "state" about, virtually, anything. In some cases one might even be inclined to trust third-party information more than self-descriptions, for instance comments about an enterprise or a product one considers to buy. The sum of RDF statements, currently known to be HTTP retrievable, is now in the order of billions with millions of individual HTTP locations (sources) hosted on tens of thousands of web sites, rapidly increasing. Along with this increased take-up of RDF on the Web, upcoming query language standards

* A preliminary version of this paper has been presented at the ISWC 2007 Workshop on New forms of Reasoning for the Semantic Web. This work has been partially supported by the European FP6 project inContext (IST-034718), by Science Foundation Ireland under the Lion project (SFI/02/CE1/I131), and by the European project DISCOVERY(ECP-2005-CULT-038206).

like SPARQL [14], or RDF search engines like SWSE [7] or Sindice [15] shall finally enable structured querying over Web data. Unfortunately however, there is no clear and established model on how to use such amounts of information coming from many diverse sources. Using any available source directly, e.g. crawling/downloading and using it might not be advisable or sufficient. More information might be needed such as, for example, patches to the original data. Other cases include when a source is in general considered useful but is known to contain statements which need to be removed, e.g. outdated facts (a "negative" information patch is needed), or subjective assertions which can be accepted or not depending on who is reading the data. In general, getting information from the Web into one's own semantic client or system is very likely to require, or at least benefit, from a series of custom steps to be performed involving a number of external or internal sources before having a version which can be used directly. Also, facing the sheer amount of data to be expected, more complex tasks such as ontological inferences or complex query answering will profit from such preprocessing which only preserves relevant and useful information. In this paper, we focus on one facet of such preprocessing, namely allowing to retract unwanted RDF data, and present a practical solution for this problem.

Along these lines, the remainder of this paper discusses the following issues: In Section 2 we introduce the idea of "Semantic Web Pipes", i.e. how a new breed of applications composed of small building blocks to aggregate, filter and preprocess junks of RDF data could contribute to make the Semantic Web real. In such aggregations from arbitrary sources on the Web we will naturally have to deal with contradicting statements. We will have a look on how current Semantic Web languages could support the expression of such contradicting/negative statements and how the resolution of conflicts is being addressed in Section 3. Actually, we will come to the conclusion that current languages do *not* properly address this problem so far. Based on this observation and in an attempt to address the problem with a technique we already successfully applied in a related domain (for synchronising RDF resources), we propose to express revocations of RDF statements by means of so called RDF MSG hashes. We discuss this approach and its implications in Section 4. A prototype implemented on top of the DBin 2.0 system is briefly described in Section 5, before we conclude with an outlook to future work.

2 Towards Semantic Web Pipes

Yahoo Web Pipes[1] are a recent development which has certainly had already a big impact to the latest wave of web development by showing how customized services and information streams can be implemented by sequentially processing and interleaving existing feeds and services. With Pipes, resources, e.g. RSS feeds, can be merged with one another, filtered according to specific pipe rules, used as an input for an on-line restful API to get yet more results, etc. Most interestingly, this all happens without the original providers of informations and services had to change anything on their side or reach any form of agreement if not to use HTTP and possibly RSS. Current mashup

[1] `http://pipes.yahoo.com/`

models like Yahoo Pipes are however limited to "streams" of information (e.g. news feeds) or single, simple API invocations on a remote site (e.g. a search for a specific word, or, more general, one-shot Web service invocations).

In the same way as a Web Pipe enables an existing Web information stream to be customized, extended and reused for a specific purpose as decided by the pipe creator, we see a very clear interest in trying to use this model to address the issue we highlighted before: how to make use of web published RDF sources? We might for example want to use DBpedia knowledge about a topic, but yet sum it with the knowledge coming from certain specific sites and correcting it by eliminating some statements we believe to be false. The Web Pipe model teaches us that we do not really want to download the DBpedia RDF dump, and operate directly on a local version of it, e.g. by adding and subtracting triples in a complex SPARQL query (see also the following Section). By doing so once and in a static manner, we would create a customized knowledge base at the beginning but would miss any new information that any of the composing sources might later add. A much more dynamic and useful model would therefore be a "Semantic Web Pipes" model where an RDF piping engine can on the fly and on demand work out the customized composition and processing of a set of Web sources according to our specific needs. In case where information needs to be simply added, the RDF semantics [8] specifies how to merge two models: the piping engine has therefore to do not much more than downloading the files and putting them together in the same store, standardizing apart blank nodes. But what to do when information needs to be patched in a traditional sense, i.e. in part both removed and added?

As a use case, let us take the case where Bob is stating that Charles knows Alice in his FOAF [5] file. Alice has a questionable reputation, and Charles, clearly, has no control on Bob's FOAF file. Clearly, a minimal requirement on distributed metadata is the ability to counter such false statements, thus giving Charles a way to state in his FOAF file a simple and unambiguous statement: "I don't know Alice". We aim to provide a simple and minimalistic solution to this problem, thus avoiding unnecessarily complex reasoning.

3 Related Works: Expressing Negative RDF Statements

First, we note that neither RDF nor RDF Schema provide means to make negative statements such as "Charles doesn't `foaf:know` Alice", see last statement in Figure 1(b).

```
@prefix : <http://examp.org/ bob#>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
:me foaf:name ''Bob''.
:me foaf:knows <http://alice.exa.org/i> .
:me foaf:knows <http://ex.org/charles#me>.
<http://ex.org/ charles#me> foaf:knows
        <http://alice.exa.org/i>.
...
```

(a) Bob's FOAF file

```
@prefix : <http://ex.org/ charles#>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix rdf: <http://www...rdf-syntax-ns#>
:me rdf:type foaf:Person;
    foaf:name "Charles".
:me foaf:knows <http://examp.org/bob#me>.
:me foaf:knows <http://alice.exa.org/i>.
...
```

(b) Charles' FOAF file

Fig. 1. Personal information in FOAF

The semantics of RDF(S) is purely monotonic and described in terms of positive inference rules, so even if Charles added instead a new statement

```
:me  myfoaf:doesntknow  <http://alice.exa.org/i> .
```

he would not be able to state that statements with the property `myfoaf:doesntknow` should single out[2] `foaf:knows` statements.

N3

Tim Berners-Lee's Notation 3 (N3) [2] provides to some extent means to express what we are looking for by the ability to declare falsehood over reified statements which would be written as:

```
{ :me foaf:knows <http://alice.exa.org/i> } a n3:falsehood .
```

Nonetheless, this solution is somewhat unsatisfactory, due to the lack of formal semantics for N3; N3's operational semantics is mainly defined in terms of its implementation cwm[3] only.

OWL

The falsehood of Charles knowing Alice can be expressed in OWL, however in a pretty contrived way, as follows (for the sake of brevity we use DL notation here, the reader might translate this to OWL syntax straightforwardly):

$$\{charles\} \in \forall \texttt{foaf:knows}.\neg\{alice\}$$

Reasoning with such statements firstly involves OWL reasoning with nominals, which most DL reasoners are not particularly good at, and secondly does not buy us too much, as the simple merge of this DL statement with the information in Bob's FOAF file would just generate a contradiction, invalidating all, even the useful answers. Para-consistent reasoning on top of OWL, such as for instance proposed in [9] and related approaches, solve this problem of classical inference, but still requiring full OWL DL reasoning.

SPARQL

Finally, more along the Pipes idea, one could as a naive solution, deploy an off-the-shelf SPARQL engine and filter Bob's FOAF file by a query, leaving just the clean statements. Imagine that Charles stores his unwanted statements in the RDF Web source `<http://ex.org/~charles/badstatements.rdf>`, then such a query filtering the information from merging Bob's and Charles' FOAF files could look as follows:

```
CONSTRUCT { ?S ?P ?O }
FROM <http://ex.org/~charles/foaf.rdf>
FROM <http://ex.org/~bob/foaf.rdf>
FROM NAMED <http://ex.org/~charles/badstatements.rdf>
```

[2] In fact, we mean here overriding instead of simply contradicting in the pure logical sense.
[3] `http://www.w3.org/2000/10/swap/doc/cwm`

```
WHERE { ?S ?P ?O .
        OPTIONAL { GRAPH <http://ex.org/~charles/badstatements.rdf>
                         { ?S1 ?P1 ?O1 . }
                   FILTER (?S1 = ?S && ?P1 = ?P && ?O1 = ?O &&) }
        FILTER ( !Bound(?S1) )  }
```

However, simply putting the bad information in a separate file is not a proper solution for the scenario we outlined, as it is not clear how a Crawler stumbling over `<http://ex.org/~charles/badstatements.rdf>` should disambiguate this data from valid RDF information. Rather, we would need to reify the negative statements using for instance the N3 version outlined before, or the "native" RDF reification vocabulary[4] which would – besides blowing up metadata by unhandy reified statements – further complicate SPARQL querying of that Data[5] to filter out the "good" data.

In the following, we will sketch a more practical solution to the problem, exploiting previous work on Minimum Self Contained Graphs.

4 Implementing RDF revocations based on MSG hashes

Any RDF graph may be viewed as set of triples. Triple level processing of distributed RDF files, particularly identifying the same RDF graphs, is made very complex by the existence of blank nodes. For this reason, the RDFSync algorithm, which we presented in previous work, introduced the notion of Minimum Self Contained Graph (MSGs) [16].

Simply said, an MSG is constructed starting from a triple and collecting, for each blank node in it, all the other triples attached to these until no more blank nodes are involved. Such "closure" makes sure that a graph can be recomposed at a different location simply by merging all the MSGs by which it is composed, even if these are transferred one at a time.

As MSGs are stand-alone RDF graphs, they can be processed with algorithms such as canonical serialization.We use an implementation of the algorithm described in [4], which is part of the RDFContextTools Java library[6] to obtain a canonical string representing the MSG and then we hash it to an appropriate number of bits to reasonably avoid collisions. This hash acts as a unique identifier for the MSG with the fundamental property of being content based, which implies that two remote peers would derive the same hash-ID for the same MSGs in their Databases.

Each graph can be therefore treated as a set of digital hashes each one representing an MSGs. In the context of the problem addressed in the present work, we use such digital hashes to refer to the MSG itself, ie. the finest granularity at which we allow to revoke RDF statements is at the level of MSGs. The hash function we use for MSGs takes the form of a literal encoding the 16 bytes of the MD5 hash of the canonical graph serialization mentioned above.

[4] Using `rdf:Statement`,`rdf:subject`, `rdf:predicate`,`rdf:object`

[5] Note that, in the FILTER query, we exploit the admittedly awkward way to model set difference in SPARQL which as such might already not be considered intuitive unanimously.

[6] http://www.dbin.org/RDFContextTools.php

Stating that an MSG is false/revoked is therefore as easy as stating one triple where the subject is a blank node, the predicate is a designated one[7] and the object is a 16 bytes literal containing the MSG hash. So the negative statement could be made directly within Charles' FOAF file or in a separate file as follows:

```
@prefix pipes: <http://pipes.deri.org/2007/10/ns#> .
_:a  pipes:revokesMSGHash
          "HASH_OF_:ME_FOAF:KNOWS_ALICE"^^xsd:string .
```

Storing MSG hashes instead of reifying statements has (except saving storage space) some other interesting implications: This solution allows revoking sets of statements which involve blank nodes. This would not be possible using reification due to the arising ambiguity. Digital hashes over MSGs, which are agnostic about blank node IDs, avoid this problem. Some particular cases, however, require further discussion (see next Session);

A drawback of the solution to quasi "encode" the negative statements in MSG hashes which in fact possibly turns out to be a feature in certain use cases, is that the negated statements are not clearly "readable", e.g. by direct inspection of the RDF file. This can be considered a feature rather than a bug for instance when one cares that denied statements are not to be known by third-parties upfront.[8]

If, on the contrary, the denied statements should be made legible, one could think of adding auxiliary statements for this purposes (such as the above-mentioned reified N3 statements, or using agreed complementary predicate URIs modified, e.g. to adding "`not:`" in front as part of the URI or as a designated URI Scheme).

4.1 MSGs involving blank nodes: issues and considerations

Our approach do not allows to revoke single statements composing an MSGs, but only the whole MSG itself. In the case the MSG in question contains blank nodes this means that if we imagine Charles would be revoking the MSG hash for

```
MSG 1:
<http://ex.org/~charles/foaf.rdf#me>  foaf:knows _:a .
 _:a foaf:name ``Alice''.            .
```

that would not have any effect if Bob had stated for instance:

```
MSG 2:
<http://ex.org/~charles/foaf.rdf#me>  foaf:knows _:a .
 _:a foaf:name ``Alice''; foaf:homepage <http://alice.exa.org/> .
```

in his graph, as the two sets of statements are actually two distinct MSGs.

Let us consider again the *MSG 1* of the previous example. As, with respect to RDF Semantics, blank nodes should be given the meaning of existential quantified variables, denying *MSG 1* would mean to deny any instance of such MSG (that is isomorphic

[7] The prefix `http://pipes.deri.org/2007/10/ns\#` defines various other properties and classes to annotate and describe revocations, see [10] for details.

[8] Although, by some additional machinery particular negated statements could be revealed quite easily in our current approach.

MSGs with a URI in place of the blank node). If, for instance, a graph contains the following statements:

```
<http://ex.org/~charles/foaf.rdf#me>  foaf:knows
<http://alice.exa.org/i> ; foaf:name ``Alice'' .
```

one might expect the revocation to affect them. The MSG based implementation, however, would left them untouched. In real cases, where blank nodes are seldom used as existential quantified variables (but rather as individual without name, as it usually happens for FOAF persons), we claim that our approach still gives correct (with respect to user expectations) results.

4.2 Computational load

Decomposing a graph into MSGs and calculating MSG hashes might be computationally expensive if the graph is big, contains a large number of bnodes, and/or highly connected bnodes. As there is no way to retrieve an MSG starting from its hash, if not decomposing the graph into MSGs and computing the hashes to find a match, the operation of applying revocations might be time consuming. To deal whit this issue, we could add additional information to revocations, namely one extra statement pointing to one, randomly chosen URI involved in the original MSG. Such an extended revocation could look as follows:

```
_:a  pipes:revokesMSGHash
          "HASH_OF_:ME_FOAF:KNOWS_ALICE"^^xsd:string .
_:a  pipes:involvedResource
          <http://alice.exa.org/i> .
```

When applying such a revocation, we only need to calculate the hashes of those MSGs which – as a sufficient condition – contain at least one statement involving the chosen URIs for revoked MSGs (in this case `<http://alice.exa.org/i>`), thus avoiding a complete graph decomposition.

Another way to go might be to do a complete MSG decomposition once, when the graph is originally loaded, and to keep an index of MSG hashes to original triples. Such initial computational effort would however result in faster operations for repeated pipe calculation. Furthermore we notice that MSG decomposition might be needed anyway for other purposes, for example to perform remote RDF synchronization [16].

5 A Simple Semantic Web Pipe Execution Engine: Description and Implementation

Having explained the idea to encapsulate negative statements in MSG hashes and its possible benefits, we have implemented a first prototypical Semantic Web Pipe engine at the heart of the DBin 2.0 Semantic Web client and authoring tool, which we conceive to be the basis of an effective Semantic Web application middle-ware. While DBin 0.x [11] based on a P2P infrastructure where information “flows” across peers, DBin 2.0 simply provides the user with a more controlled way to define the order and the

location of the sources to import and then "executes" the pipe to generate a final RDF base which is then browsed and queried.

For our simple prototype, we exploit this order in evaluating RDF statements to be overridden: In the DBin piping engine, RDF sources can be either local or remote. These are ordered in a stack according to the priority selected by the user. At execution stage, a new empty triplestore is created which will contain the graph resulting from the pipe, let us call it 'T'. The sources are then processed one by one, from the one with the lowest priority to the one with the highest priority.[9] Naming the currently processed graph 'G', the "ordered merge" procedure is the following:

1. G is cleaned by any negative MSG that overwrites a positive MSG in G (this means that if G expresses "X" and "not X" we delete both the assertions);
2. The content of G is added to T;
3. Negative statements are "applied", i.e., if positive statements exist in T corresponding to statements revoked in G, the lower priority positive statements are removed (this step is the same of the first one except that it is applied to the resulting graph T);
4. Any remaining revocations are dropped, as they must not have effects on the higher priority graphs considered in next cycles.

Once this ordered, conflict-resolving "merge" procedure has been performed for all the RDF sources, T contains the final RDF model and DBin applies RDFS reasoning on it. We remark that the result in absence of negated statements tantamounts to exactly the common RDF merge.

Clearly, by handling conflict resolution at the RDF merge level, and applying RDFS reasoning only at the last step many issues are solved in a simple, intuitive and, at the same time, efficient manner. By removing at each step any remaining negative statement we opt for a "non symmetric" approach where positive statements are somehow considered more important and persistent than "negative" ones. Moreover, the remaining RDF set is clearly consistent (being simple RDF).

We note however, that there could also be possibly problematic corner cases. For instance, imagine that Bob sneaks in the unwanted statement about Alice as follows:

```
<http://ex.org/~charles#me>
       myfoaf:likes <http://alice.exa.org/i>.
myfoaf:likes rdfs:subPropertyOf foaf:knows.
```

In this disguise, even if Bob's FOAF data is given lower priority than Charles' FOAF file, the unwanted statement would survive the conflict resolution during our ordered merge, since we do not do RDFS inference in this process.

We are currently, investigating repairs to our approach which remedy this situation, e.g. by labeling inferred triples with the priority of the lowest statement contributing to their inference and, in a recursive process removing conflicting inferred triples in a post processing step. Unfortunately, we conjecture that finding this lowest statement is,

[9] In the current implementation, the priorities are implicitly given through a simple sequence of sources which is processed one by one and priority is thus totally ordered.

in the general intractable[10], but we hope that an approximative solution, which at least guarantees that only overall sound triples are inferred might be achievable.

Another drawback of the current approach is that the priority order among considered RDF sources has to be given upfront as user input to DBin, which might not be a problem for smaller scale pipe examples, but be undesirable as the number of known sources grow to large scale. Trust negotiation policies, see e.g. [3], encoded directly as RDFstatements within the sources could help to assess priorities among RDF sources as we require them directly from RDF data in those resources.

Finally we notice that, in some cases, the end user might want to have more options than simply putting the considered sources in a total order, i.e. the pipe being a strict sequence of sources. To allow more flexible handling of overriding statements, allowing to consider multiple sources at the same priority, we are working to add support for a partial rather than a total order of sources. There might be different ways to handle revocations within a set of sources that have equal priority. A "cautious" solution might be to allow each source to revoke both MSGs from any of the equally prioritized sources and MSGs which are stated by sources with lower priority. An other approach, that we call "brave ", might be to ignore revocations coming from equally prioritized sources and to apply only those that come from a higher prioritized source.

6 Conclusions and future works

We outlined in the present work a practical solution to add negative statements to RDF without generating overall logical inconsistency. Even leaving aside full OWL inference, we believe that being able to override RDF statements based on user priorities on which Web resources are more or less trustworthy, is a crucial feature in Semantic Web applications. In this paper we first analyzed how negative statements can at all be expressed in current Semantic Web languages and came to the conclusion these languages do not properly address this problem, not providing means to override statements in a user defined priority order among RDF sources on the Web. Based on this observation, we presented a practical solution to the problem which is implemented on top of the DBin 2.0 system.

Our general ideas are based on the assumption that we believe only partially in Web scale DL reasoning, i.e. handling complete OWL inferencing, to be feasible in the near future. Our approach is a more practical one dealing with the increasing number of RDF data out there in an effective and arguably feasible manner. Negative statements treated in this work, which is still in a preliminary stage, are a first example of practical necessities we plan to address when effectively and efficiently processing Semantic Web data for useful Semantic Web applications in the spirit of "Semantic Web Pipes". In this sense, this work is conceived to spark discussions for more practical solutions towards making the Semantic Web real, which might also raise controversy among "purists" in terms of what the term "Semantic Web Reasoning" comprises and what not. More examples of issues we want handle in practical implementations include linking RDF data by adding views (see also [6, Section 2.10]), possibly involving scoped negation [13, 12] and evaluate scalability of such extensions in practical scenarios.

[10] A concrete algorithm and complexity studies for such an algorithm are still on our agenda

References

1. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *6th Int'l Semantic Web Conference*, Busan, Korea, Nov. 2007.
2. T. Berners-Lee. Notation 3, since 1998. Available at `http://www.w3.org/DesignIssues/Notation3.html`.
3. P. A. Bonatti and D. Olmedilla. Rule-based policy representation and reasoning for the semantic web. In *Reasoning Web - Third International Summer School*, pages 240–268, Dresden, Germany, Sept. 2007.
4. J. J. Carroll. Signing rdf graphs. In *The Semantic Web - ISWC 2003, Second International Semantic Web Conference*, pages 369–384, Sanibel Island, FL, USA, Oct. 2003.
5. D. Brickley and L. Miller. Friend of a Friend (FOAF) Vocabulary Specification 0.9. Namespace Document, May 2007, available at `http://xmlns.com/foaf/spec/20070524.html`,.
6. A. Ginsberg, D. Hirtle, F. McCabe, and P. Patranjan (eds.). RIF Core Design. W3C Working Draft 10 July 2006, available at `http://www.w3.org/TR/2006/WD-rif-ucr-20060710/`.
7. A. Harth, J. Umbrich, and S. Decker. Multicrawler: A pipelined architecture for crawling and indexing semantic web data. In *5th International Semantic Web Conference*, Athens, GA, USA, Nov. 2006.
8. P. Hayes. RDF semantics. W3C Recommendation, February 2004, available at `http://www.w3.org/TR/rdf-mt/`.
9. Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, Aug. 2005.
10. C. Morbidoni, A. Polleres, G. Tummarello, and D. Le Phuoc Semantic Web Pipes. Technical Report DERI-TR-2007-11-07, available at `http://www.deri.ie/fileadmin/documents/DERI-TR-2007-11-07.pdf`, Nov. 2007.
11. M. Nucci, C. Morbidoni, and G. Tummarello. Enabling semantic web communities with dbin: an overview. In *ISWC2006 Semantic Web challenge*, Athens, GA, USA, 2006. Finalist.
12. A. Polleres, C. Feier, and A. Harth. Rules with contextually scoped negation. In *3rd European Semantic Web Conference (ESWC2006)*, volume 4011 of *Lecture Notes in Computer Science*, Budva, Montenegro, June 2006. Springer.
13. A. Polleres, F. Scharffe, and R. Schindlauer. SPARQL++ for mapping between RDF vocabularies. In *6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007)*, Vilamoura, Algarve, Portugal, Nov. 2007. To appear.
14. E. Prud'hommeaux and A. Seaborne (eds.). SPARQL Query Language for RDF. W3C Candidate Recommendation, June 2007, available at `http://www.w3.org/TR/2007/CR-rdf-sparql-query-20070614/`.
15. G. Tummarello, R. Delbru, and E. Oren. Sindice.com: Weaving the open linked data. In *Proceedings of the International Semantic Web Conference (ISWC)*, Nov. 2007. To appear.
16. G. Tummarello, C. Morbidoni, P. Puliti, and F. Piazza. Signing individual fragments of an RDF graph. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, Chiba, Japan, 2005.

Semantic-enhanced EPCglobal Radio-Frequency IDentification

Michele Ruta[1], Tommaso Di Noia[1], Floriano Scioscia[1], Eugenio Di Sciascio[1]

SisInfLab, Politecnico di Bari, Bari, Italy
{m.ruta,t.dinoia,f.scioscia,disciascio}@poliba.it

Abstract. We propose to enhance EPCglobal RFIDs enriching them with semantic capabilities. Memory organization of tags and the data exchange protocol are exploited and extended to the purpose. By design, the proposed enhancements do not alter the basic behavior of protocol and tag memory organization and are thus fully backward compatible. In order to store annotated descriptions, a compression algorithm for DIG syntax has also been devised. We report here results in comparison with other XML-based compression tools and simulations for enhanced tags reading and decompression.

1 Introduction

Radio-Frequency IDentification (RFID) is a promising infrastructure-less technology interconnecting via radio two main components: **(1)** a transponder carrying data (*tag*) located on the object to be identified; **(2)** an interrogator (*reader*) able to receive the transmitted data. Traditional RFID applications have been focused on supply chain management and asset tracking [12]. Nevertheless, at the state of the art, tags with higher memory capacity and on-board sensors disclose new scenarios and enable further applications. Currently, RFID technology is merely used as a link between physical objects and a "virtual counterpart" [9] in the digital world. Tags only store an identification code, which is used as a key to retrieve relevant properties of the object from an information server, through a networked infrastructure. Two main issues restrain an overall exploitation of the standard capabilities. First of all, the original identification mechanism only enables a rudimentary string matching, providing exclusively "yes/no" replies. Furthermore, RFID-based technology usually relies on stable support infrastructure and fixed database servers.

We propose an extension of EPCglobal RFID standard [11] supporting logic-based formalisms for knowledge representation and enabling advanced services. Semantic-based annotations are stored on RFID tags, exploiting machine understandable ontological languages originally conceived for the Semantic Web effort. Noteworthily, protocols to read/write tags are preserved in the proposed extension, maintaining the original code-based access, thus keeping a backward compatibility with basic applications practically without any modification.

According to W3C recommendations for mobile applications [7], our approach copes with limited storage and computational capabilities of mobile and

embedded devices, and with reduced bandwidth provided by wireless links. Issues related to the verbosity of semantic annotation languages cannot be neglected. Compression techniques become essential to enable storage and transmission of semantically annotated information in mobile contexts. Hence, in order to make our approach sustainable in reality, we devised and exploited a novel efficient XML compression algorithm, specifically targeted for DIG 1.1 [1] document instances.

2 Motivation

The main idea of our approach is that a semantic-based extension of current RFID technology supporting formalisms for knowledge representation, allows semantically rich and unambiguous information to follow an object in each step of its life cycle. Products then auto-expose their description to whatever RFID-enabled computing environment they are dipped in. This favors decentralized approaches for context-aware applications in pervasive computing environments, based on less expensive and more manageable mobile ad-hoc networks. Product and process information can be queried, updated and integrated during manufacturing, quality control, packaging and supply chain management, thus allowing full traceability up to sales, and intelligent and de-localized querying of product data. Semantic-enhanced RFID object discovery can be leveraged also for sales and post-sale services, by assisting customers in using the products they purchased more effectively.

Beyond manufacturing and commerce, other application areas can benefit from adding accurate semantic-based object description to traditional RFID identification and tracking capabilities. For example, in tourism settings such as museums or archaeological sites, visitors could perform interactive knowledge discovery by approaching tagged items with an RFID-enabled mobile device and querying the system for further resources of interest. In the healthcare sector, relevant information can be embedded within RFID tags attached to patient accessory (*e.g.*, wristband) and to drug packages. Since no further infrastructure is needed, support can be provided for patient diagnosis and therapy at the hospital as well as for follow-up at home.

3 Proposed Enhancements

3.1 EPCglobal RFID standards

In our framework we refer to RFID transponders conforming to the EPC (Electronic Product Code) standard for class I - second generation UHF tags [11]. We assume the reader be familiar with basics of this technology.

The practical feasibility of a proposal for advanced usage of RFID technologies must take into account some important constraints. First of all the severe bandwidth and memory limitations of current RFID systems, in order to meet cost requirements for large-scale adoption. Due to technological advances and

Table 1. SELECT command able to detect only semantic enabled tags

PARAMETER	Target	Action	MemBank	Pointer	Length	Mask
VALUE	100_2	000_2	01_2	00010101_2	00000010_2	11_2
DESCRIPTION	SL flag	set (if match)	EPC bank	initial address	bit to be compared	bit mask

growing demand, passive RFID tags with greater memory amounts are expected to be available [2]. Nevertheless, XML-based ontological languages like OWL (http://www.w3.org/TR/owl-features/) and DIG (http://dl.kr.org/dig/) are far too verbose for a direct storage on RFID tags. A further goal is to preserve the original EPCglobal RFID technology standards as much as possible, in order to ensure compatibility and smooth coexistence of new semantic-based object discovery applications and legacy identification and tracking ones.

In order to enable the outlined enhancements, RFID tags and the air interface protocol must provide read/write capabilities for semantically annotated product descriptions w.r.t. a reference ontology, along with additional data-oriented attributes. Neither new commands nor modification to existing ones have been introduced. Moreover, a mechanism is clearly required to distinguish semantic enabled tags from standard ones, so that semantic based applications can exploit the new features without interfering with legacy applications. In order to accomplish that, we extend the memory organization of tags compliant with the above referenced standard. We exploit two bits in the EPC tag memory area currently reserved for future purposes. The first one –at 15_{hex} address– is used to indicate whether the tag has a user memory (bit set) or not (bit reset). The second one –at 16_{hex} address– is set to mark semantic enabled tags. In this way, a reader can easily distinguish semantic based tags by means of a SELECT command with parameter values as in Table 1. Values for the triple $\langle MemBank, Pointer, Length\rangle$ identify the two-bit memory area starting at 15_{hex} address in the EPC memory bank. The reader commands each tag in range to compare those two bits with bit mask 11_2. The match outcome will be positive for semantic enabled tags only. The *Target* and *Action* parameter values mean that in case of positive match the tag must set its *SL* flag and clear it otherwise. The following inventory step will skip tags having SL flag cleared, thus allowing a reader to identify only semantic enabled tags. Protocol commands belonging to the inventory step have not been described, because they are used in the standard fashion.

The EPC standard requires the content of TID memory up to $1F_{hex}$ bit is fixed. TID bank can be extended to store optional information, generally consisting of tag serial number or manufacturer data. Hence we use the TID memory area starting from 100000_2 address to store a 128-bit *Ontology Universally Unique Identifier* (OUUID) marking the ontology w.r.t. the description contained within the tag is expressed [10]. In order to retrieve the OUUID stored within a tag, a reader will exploit a READ command by adopting parameter values as in Table 2. *MemBank* parameter identifies the TID memory bank and the *WordPtr* value specifies that the reading must start from the third 16-bit memory word, *i.e.*, from 20_{hex} address. Finally, the *WordCount* parameter indicates that 128 bits (eight 16-bit words) have to be read.

Table 2. READ command able to extract the OUUID from the TID memory bank

PARAMETER	MemBank	WordPtr	WordCount
VALUE	10_2	00000010_2	00001000_2
DESCRIPTION	TID memory bank	starting address	read up to 8 words (128 bit)

Table 3. READ command able to extract the semantically annotated description from the User memory bank

PARAMETER	MemBank	WordPtr	WordCount
VALUE	11_2	00000000_2	00000000_2
DESCRIPTION	User memory bank	starting address	read up to the end

Contextual parameters (whose meaning may depend on the specific application) are stored within the *User memory bank* of the tag. There, we also store the semantically annotated description of the product the tag is clung to (compressed with the algorithm described later on). An RFID reader can perform extraction and storing of a description from/on a tag by means of one or more READ or WRITE commands, respectively. Both commands are obviously compliant with the RFID air interface protocol. Table 3 reports parameter values of the READ command for extracting the full contents of the User memory, comprising both contextual parameters and the compressed annotation.

The EPCglobal standard also provides a support infrastructure for RFID applications by means of the so called *Object Naming Service* (ONS) [3]. In our approach the ONS mechanism is considered as a supplementary system able to grant the *ontology support*. If a reader does not manage the ontology the description within the tag refers to, we may retrieve it exploiting the ONS service. The *EPCglobal Network Protocol Parameter Registry* maintains all the registered service suffixes (*ws* for a Web service, *epcis* for a EPCglobal Information Service (providing authoritative information about objects associated with an EPC code), *html* for a Web Page of the manufacturer). We hypothesize to register the new *dig* suffix to indicate a service able to retrieve ontologies with a specified OUUID value.

In case of EPC derived from the GS1 standard[1], we assume that the pair of fields used for ONS requests –and referred to the manufacturer and to the merchandise class of the good– will correspond to a specific ontology. In fact that pair identifies exactly the product category. Two goods with the same values for that field parameters will be surely homogeneous or even equal. Nevertheless the vice versa is not verified.

3.2 Compression algorithm

A compression algorithm specifically targeted to the packing of standard DIG 1.1 format has been devised in our framework. The general approach, however, is easily adaptable to any other ontological language based on XML, such as OWL. Each DIG document instance conforms to DIG XML Schema, which comprises

[1] GS1 (originally EAN.UCC) is the international organization that introduced the bar code identification of products and services.

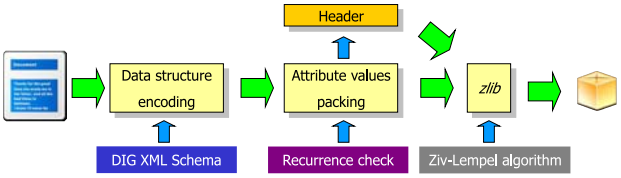


Fig. 1. Structure of the proposed DIG compression tool

at most 40 different tags. In a DIG document, no value is set inside any tag; only tag attributes can be specified, within a well defined finite set of types.

We propose a simple DIG compression solution particularly suitable for pervasive applications, whose structure is shown in Figure 1. Three fundamental phases can be identified: **(1)** *data structures packing*; **(2)** *attribute values packing*; **(3)** *zlib packing*. We exploit the peculiarity of the DIG format having few, well defined and limited tag elements.

(1) *Data-structures packing*. The proposed compression algorithm is based on two fundamental principles. First of all, pure data have to be divided from data structures; furthermore data and data structures have to be separately encoded in order to obtain a higher compression rate. Data structures are basically XML elements with possible related attributes, whereas data simply are attribute values. As noted above, data-structures in DIG syntax are fixed and well defined by DIG XML Schema, whereas data are different from document to document. XML elements are coded by associating an unambiguous 8-bit code to each structure in a static fashion. Consider that DIG files adopt either `ISO 8859-1` or `UTF-8` character encodings, which use 1 byte for each character (special characters requiring more than 1 byte in `UTF-8` do not belong to the DIG symbol set): so an early size saving is achieved. The association between XML structures and the corresponding code is fixed and invariable. This is a further advantage because, unlike general purpose XML compressors, it is unnecessary to include a header containing the decoding table within the compressed file.

(2) *Attribute-values packing*. Most recurrent words are identified in the previously distinguished data section. They will be coded with a 16-bit sequence. A header for the compressed file is thus built, containing correspondences between each text string and the related 16-bit code. It is dynamically created and exclusively belongs to a specific DIG document instance. The provided header will be exploited in the decompression phase.

Assigned codes differ by their second byte, because the first octet is adopted as padding in order to distinguish the attribute value coding from regular ASCII characters. This second compression stage allows to obtain a further size saving, particularly in ontologies with very frequent concepts and roles. On the other hand, the use of the header could compromise compression performances for short files, as the space consumption for the header itself reduces savings obtained with compression. Hence the encoding of all the string values of a DIG file without any a-priori distinction has to be definitely avoided.

A correct compression procedure should properly take into account both the length of an attribute string and its number of occurrences within the file. The minimum length of strings to encode can be trivially calculated by comparing the size consumption needed to store *string–code* correspondences and the saving obtained with the encoding: in the proposed approach only text attributes with a length of at least three characters will be encoded.

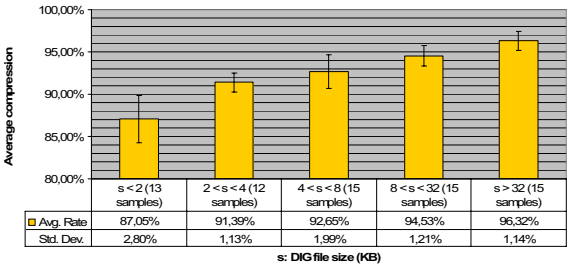
Furthermore we must evaluate the number of occurrences of each attribute i (from now on $nr\_occurences_i$). We set an optimal minimum value we call $nr\_occurences\_min$ and we will encode only i attribute values where results $nr\_occurences_i > nr\_occurences\_min$. We have performed statistical evaluations trying the compression of 72 sample DIG documents and evaluating obtained compression rates varying $nr\_occurences\_min$. Results show that the best compression rates are produced by $nr\_occurences\_min$ values within the range [2–8] with an average of 4.03 and a standard deviation in the range [0–0.3]. Thus we set $nr\_occurences\_min = 4$, so only attribute strings with at least three characters and recurring at least four times will be encoded.

(3) *zlib packing.* Finally *zlib* library based on the *Ziv-Lempel* compression algorithm [13] is exploited to apply an eventual third compression level, optimizing the overall result. Ziv-Lempel algorithm does not perform particularly well when compressing a partially coded input (it is difficult to find more occurrences of the same character sequence). The use of *zlib*, however, resulted useful in our approach specially for large files, where it produces the compression of words excluded by previous compression steps and of the file header.

4 Evaluation

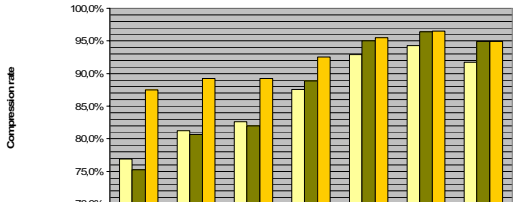
A generic evaluation of the proposed approach has been carried out taking into account two different aspects. First of all, performances of the compression/decompression algorithms have been investigated and furthermore reading and decompression times of software simulated semantic-enhanced RFID tags were evaluated, in order to provide an initial assessment of the impact that our approach may have on RFID systems performances.

Regarding the compression and decompression performances three fundamental parameters have been estimated: **(1)** *compression rate*, **(2)** *turnaround time*, **(3)** *memory exploitation.* Two tools were developed in C language implementing our compression and decompression algorithms. They were named *DIG Compressor* and *DIG Decompressor*, respectively. Currently, Windows and Linux platforms are supported, leveraging the freely available *zlib* compression library. Tests for compression rate and running time were performed using: (1) a PC equipped with an Intel Pentium 4 CPU (3.06 GHz clock frequency), 512 MB RAM at 266 MHz and Windows XP operating system; (2) a PC equipped with a Pentium M CPU (2.00 GHz clock frequency) and 1 GB RAM at 533 MHz, running Gentoo GNU/Linux with 2.6.19 kernel version and *Valgrind* [6] profiling toolkit.



	s < 2 (13 samples)	2 < s < 4 (12 samples)	4 < s < 8 (15 samples)	8 < s < 32 (15 samples)	s > 32 (15 samples)
Avg. Rate	87,05%	91,39%	92,65%	94,53%	96,32%
Std. Dev.	2,80%	1,13%	1,99%	1,21%	1,14%

Fig. 2. Obtained compression rates



	instance1	instance2	instance3	ontology1	ontology2	ontology3	ontology4
	2035	3445	4079	12801	66247	111384	190685
gzip	76,9%	81,2%	82,6%	87,6%	92,9%	94,3%	91,7%
XMill	75,2%	80,7%	82,0%	88,9%	95,0%	96,4%	94,9%
DIG Compressor	87,5%	89,2%	89,2%	92,5%	95,5%	96,5%	94,9%

Fig. 3. Performance comparison on a representative sample of DIG documents – **Compression rate**

Firstly, compression rates achieved by the proposed algorithm were considered. We carried out tests with 70 DIG documents of various size. Our aim was to evaluate compression rates for both smaller instance descriptions and larger ontologies. Figure 2 shows average compression rates and standard deviations for different size ranges of DIG input data. Overall average compression rate is $92.58 \pm 3.58\%$. As expected, higher compression rates were achieved for larger documents. Even for very short DIG files (less than 2 kB), however, average compression rate is $87.05 \pm 2.80\%$, which is surely satisfactory for our purposes.

A comparative evaluation was carried out using as benchmarks the general purpose XML compressor *XMill* [5] and *gzip* (http://www.gzip.org/) generic compressor. Testing the compression rate, the proposed system allowed to obtain smallest resulting files, as shown in Figure 3. For each DIG file, the original size in bytes is reported. It should be noticed our algorithm performed significantly better for small DIG documents. This result is very encouraging, since in our mobile scenarios we usually deal with small XML documents representing the annotations of available resources.

In order to evaluate the turnaround time, each test was run 10 times consecutively, and the average of the last 8 runs was taken. Results are presented in Figure 4. It can be noted that DIG Compressor has higher turnaround times

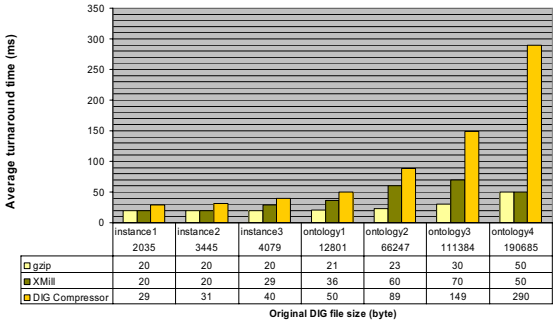


Fig. 4. Performance comparison on a representative sample of DIG documents – **Turnaround time**

than other tools, though absolute values are still quite acceptable. Such a result suggests we need further optimizations for execution speed.

Finally, memory usage analysis was performed using *Massif* tool of *Valgrind* debugging and profiling toolkit. Massif measures stack and heap memory profile throughout the life of a process. For our comparison, only the memory occupancy peak was considered. DIG Compressor memory usage is only slightly higher than the one of *gzip*, with high correlation ($r = 0.96$) between the two value sets. This result could be expected, since our algorithm relies on Ziv-Lempel compression in its last phase. On the contrary, *XMill* showed a more erratic behavior. Outcomes can be reputed as encouraging because memory-efficient implementations of *zlib* library are currently available for all major mobile platforms.

A thorough experimental evaluation of all aspects of framework performance requires its complete implementation into a testbed with real semantic-enabled RFID devices. That would only be possible through partnership agreements with device manufacturers/integrators, that we are currently pursuing. At this stage, a prototypical semantic-enhanced RFID infrastructure has been simulated by extending IBM WebSphere RFID Tracking Kit middleware solution for RFID applications. RFID simulations and tests have been performed on that testbed, which is deployed on a laptop PC equipped with Pentium M processor (2.00 GHz clock frequency), 1 GB RAM at 533 MHz and Microsoft Windows XP operating system. Compressed semantic annotations of 40 different products were used. Their average size is 266 ± 104 B (range 91-440 B). Simulated RFID data access from each tagged item was repeated 100 times, recording the sum of reading and decompression time. For each item the mean value was then considered.

Results are reported in Figure 5. Average access time is 2.02 ± 0.36 ms, corresponding to a theoretical tag read rate of approximately 500 tags/s. Since tests were run on a software-simulated RFID platform, exact numerical values are not significant as their order of magnitude. The latter can be sensibly compared to performance of RFID systems compliant with EPCglobal standards for Class 1 Generation 2 UHF RFID systems.

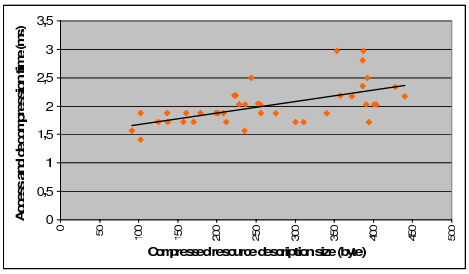


Fig. 5. Simulated RFID tag reading and decompression time for 40 resource descriptions. Regression line is plotted.

It is known that RFID system performance in the field highly depends on the particular application, environmental conditions (electromagnetic noise, RFID reader density) and local regulations affecting the available bandwidth. Early simulations and tests carried out by independent research laboratories estimated reading rates in a range of 7-100 tags/s in typical conditions [8, 4]. Our simulation results are quite above these with such data. Hence, a very preliminary evidence that adoption of compressed semantic annotations on RFID tags does not impair performances in the field w.r.t. traditional ones is so provided. The latter, in turn, will not suffer any direct performance degradation from the newly introduced features, as they will read the EPC only. Finally, the access time showed a moderate positive correlation ($r = 0.60$) with annotation size. This may suggest that structure of a DIG annotation (*i.e.*, exploited logic constructors and frequency of attribute names) has also an impact over the decompression.

5 Conclusions

Our approach can support a range of use cases, involving different stakeholders in each step of a product life cycle. During product manufacturing and distribution, a wide-area support network interconnecting commercial partners is not strictly needed. This is a significant innovation w.r.t. common RFID supply chain management solutions. By means of their semantic-enabled RFID tags, products are always accompanied by structured and rich description of their characteristics, endowed with unambiguous and machine-understandable semantics. Beyond improved traceability, a semantic-based approach may provide unique value-added capabilities. In particular, query flexibility and expressiveness are much greater than both keyword-based information retrieval and standard service/resource discovery protocols, which support code-based exact matches only. Non-exact match types are prevalent in real scenarios, involving a large number of resources by many different sources. Semantic-based techniques can support non-exact matches and ranking, further providing means for results explanation. This enables an effective query refinement process and can strengthen user trust in the discovery facility.

Bibliography

[1] S. Bechhofer, R. Möller, and P. Crowther. The DIG Description Logic Interface. In *Proceedings of the 16th International Workshop on Description Logics (DL'03)*, volume 81 of *CEUR Workshop Proceedings*, September 2003.

[2] R. Bridgelall. Enabling Mobile Commerce Through Pervasive Communications with Ubiquitous RF Tags. *IEEE Wireless Communications and Networking, (WCNC)*, 3:2041–2046, March 2003.

[3] EPCglobal Ratified Specification. Object Naming Service (ONS - ver. 1.0). http://www.epcglobalinc.org, October, 4, 2005.

[4] Y. Kawakita and J. Mistugi. Anti-collision performance of Gen2 air protocol in random error communication link. In *Proceedings of the International Symposium on Applications and the Internet Workshops - SAINT 2006*, pages 68–71, 2006.

[5] H. Liefke and D. Suciu. Xmill: an efficient compressor for xml data. *SIGMOD Rec.*, 29(2):153–164, 2000.

[6] N. Nethercote and J. Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. In *Conference on Programming Language Design and Implementation - PLDI 07*. ACM SIGPLAN, June 2007.

[7] J. Rabin and C. McCathieNevile. Mobile Web Best Practices 1.0. *W3C Proposed Recommendation*, 2006.

[8] K. Ramakrishnan and D. Deavours. Performance benchmarks for passive UHF RFID tags. In *Proceedings of the 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems*, 2006.

[9] K. Römer, T. Schoch, F. Mattern, and T. Dübendorfer. Smart Identification Frameworks for Ubiquitous Computing Applications. *Wireless Networks*, 10(6):689–700, November 2004.

[10] M. Ruta, T. Di Noia, E. Di Sciascio, and F. Donini. Semantic-Enhanced Bluetooth Discovery Protocol for M-Commerce Applications. *International Journal of Web and Grid Services*, 2(4):424–452, 2006.

[11] K. Traub, G. Allgair, H. Barthel, L. Bustein, J. Garrett, B. Hogan, B. Rodrigues, S. Sarma, J. Schmidt, C. Schramek, R. Stewart, and K. Suen. EPCglobal Architecture Framework. Technical report, EPCglobal, July 2005.

[12] R. Weinstein. Rfid: A technical overview and its application to the enterprise. *IT Professional*, 07(3):27–33, 2005.

[13] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

Author Index