Proceedings of the

3rd International Workshop on

Evaluation of Ontology based Tools

(EON 2004)

Organization Committee

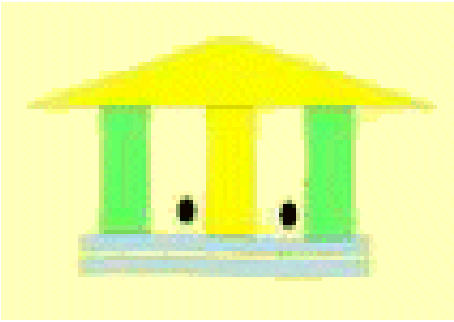
York Sure (Contact Person), Institute AIFB at University of Karlsruhe (DE)
Oscar Corcho, Intelligent Software Components, S.A. (ES)
Jérôme Euzenat, INRIA (FR)
Todd Hughes, Lockheed Martin (US)

Program Committee

Dean Allemang, TopQuadrant, Inc. (US), dallemang_@_topquadrant.com
Bill Andersen, Ontology Works, Inc. (US), andersen_@_ontologyworks.com
Sean Bechhofer, University of Manchester (UK), seanb_@_cs.man.ac.uk
Richard Benjamins, Intelligent Software Components, S.A. (ES), rbenjamins_@_isoco.com
John Davies, BT (UK), john.nj.davies_@_bt.com
AnHai Doan, University of Illinois (US), anhai_@_cs.uiuc.edu
Christian Fillies, SemTalk (DE), cfillies_@_semtalk.com
Fausto Giunchiglia, University of Trento (IT), fausto_@_dit.unitn.it
Asunción Gómez-Pérez, Universidad Politécnica de Madrid (ES), asun_@_fi.upm.es
Kouji Kozaki, Osaka University (JP), kozaki_@_ei.sanken.osaka-u.ac.jp
Natasha F. Noy, Stanford University (US), noy_@_smi.stanford.edu
Henrik Oppermann, Ontoprise (DE), oppermann_@_ontoprise.de
Norman Sadeh, Carnegie Mellon University (US), sadeh_@_cs.cmu.edu
Heiner Stuckenschmidt, Vrije Universiteit Amsterdam (NL), heiner_@_cs.vu.nl
Rudi Studer, University of Karlsruhe (DE), studer_@_aifb.uni-karlsruhe.de
Raphael Troncy, INA (France), for Institute National of Audiovisual,
Raphael.Troncy_@_ina.fr
Mike Uschold, Boeing (US), michael.f.uschold_@_boeing.com
Takahira Yamaguchi, Keio University (JP), yamaguti_@_ae.keio.ac.jp

Sponsorship

This workshop is sponsored by the EU thematic network of excellence Knowledge Web (http://knowledgeweb.semanticweb.org) and the EU integrated project SEKT (http://sekt.semanticweb.org).



Introduction

In the series of “Evaluation of Ontology-based Tools” workshops we intend to bring together researchers and practitioners from the fastly developing research areas “ontologies” and “Semantic Web”. Currently the semantic web attracts researchers from all around the world. Numerous tools and applications of semantic web technologies are already available and the number is growing fast. However, deploying large scale ontology solutions typically involves several separate tasks and requires applying multiple tools. Therefore pragmatic issues such as interoperability are key if industry is to be encouraged to take up ontology technology rapidly.

The main aim of this workshop is therefore to encourage and stimulate discussions about the evaluation of ontology-based tools. For the future this effort might lead to future benchmarks and certifications.

This workshop follows the previously held EON2002 (held at the EKAW 2002) and EON2003 (held at the ISWC 2003).

The EON workshops are associated with experiments. Initially they corresponded to the OntoWeb SIG on tools discussions, now this evolved into the Knowledge Web, in particular the workpackage on Heterogeneity for the experiment for this workshop. The general question is how to evaluate ontology related technologies. To break down this rather complex task into a pragmatic one, the group decided to focus first on ontology engineering environments (OEE) as a starting point in EON2002 and EON2003. These tools are rather common and widely used by the Semantic Web Community and some of the participating members were even tool provider themselves. During the first experiment, design issues and limitations of state-of-the-art OEEs were being evaluated. This time the focus was on ontology alignment, all participants were challenged to provide a set of mappings between given ontological fragment. The aim was to compare the alignments given by all participants to give an overview on state-of-the-art ontology alignment tools and their abilities.

Further information about the workshop and the experiment can be found at: `http://km.aifb.uni-karlsruhe.de/ws/eon2004/`

We thank all members of the program committee, additional reviewers, authors, experiment participants and local organizers for their efforts.

We are looking forward to having fruitful discussions at the workshop!

York Sure, Oscar Corcho, Jérôme Euzenat, Todd Hughes

Table of Contents

A Benchmark Suite for Evaluating the Performance of the WebODE Ontology Engineering Platform

Raúl García-Castro, Asunción Gómez-Pérez

Ontology Engineering Group, Laboratorio de Inteligencia Artificial.
Facultad de Informática, Universidad Politécnica de Madrid, Spain
{rgarcia,asun}@fi.upm.es

Abstract. Ontology tools play a key role in the development and maintenance of the Semantic Web. Hence, we need in one hand to objectively evaluate these tools, in order to analyse whether they can deal with actual and future requirements, and in the other hand to develop benchmark suites for performing these evaluations. In this paper, we describe the method we have followed to design and implement a benchmark suite for evaluating the performance of the WebODE ontology engineering workbench, along with the conclusions obtained after using this benchmark suite for evaluating WebODE.

1. Introduction

In order for the Semantic Web to consolidate steadily, it needs the support of technology that allows to create and to maintain it. The continuous development of ontology editors and ontology tools for managing ontologies is an indication of this fact. These tools implement different knowledge models with different underlying knowledge representation paradigms, manage large upper level and general ontologies, and range from standalone to web-based applications. To be able to decide what ontology tools are needed for fulfilling actual and future requirements, we need to objectively evaluate them with regard to their quality attributes.

Evaluation studies and benchmark suites for ontology technology are still a bit scarce. For this reason, we think that there is a need to construct benchmark suites for ontology tools, in order to be able to objectively assess their quality and to allow for a better integration of this technology into other information systems.

The volume of ontologies and the amount of users that work concurrently with ontology tools increases continuously. Therefore, the performance of these tools emerges as one of the quality attributes to take care of.

As the development activity is one of the main ontology life cycle activities [1], we will first deal with the appraisal of ontology development tools. In this work, we focus on the WebODE ontology engineering workbench [2], evaluating its performance in terms of execution efficiency [3]. WebODE's global performance is inferred from the performance of the methods of its ontology management API, which allow managing the ontology components defined in the WebODE knowledge model (concepts, relations, instances, axioms, constants, bibliographic references, and imported terms).

The contents of this paper are the following:

Section 2 presents the state of the art in evaluating ontology development tools.

Sections 3 to 7 describe the method that we have followed to design and implement a benchmark suite for evaluating WebODE's performance and the analysis performed after executing it.

Section 8 presents the conclusions obtained and the related future work.

2. Related Work

Ontology technology has improved enormously since the creation of the first environments in the mid-1990s. In general, ontology technology hasn't been the object of evaluation studies but, as the use of this technology spreads, in the last few years many studies involving ontology tools evaluation have been performed. Most of these studies deal with the evaluation of ontology development tools.

Some authors have proposed a general framework for the evaluation of ontology development tools. To this group belongs the work presented by: Duineveld and colleagues [4], the deliverable 1.3 of the OntoWeb project [5], the experiments performed in the First International Workshop on the Evaluation of Ontology-based Tools (EON2002) [6], and Lambrix and colleagues [7].

Other authors have focused in specific criteria regarding ontology development tools: Stojanovic and Motik [8] analyzed the ontology evolution requirements fulfilled by the tools, Sofia Pinto and colleagues [9] evaluated the support provided by the tools in ontology reuse processes, the experiments of the Second International Workshop on Evaluation of Ontology-based Tools (EON2003) [10, 11, 12, 13, 14] involved the interoperability of the tools, and Gómez-Pérez and Suárez-Figueroa [15] evaluated the ability of the tools to detect taxonomic anomalies.

As a general comment, evaluation studies concerning ontology development tools have been of qualitative nature. To be able to objectively assess and compare these tools, evaluation criteria must be defined and benchmark suites must be developed in order to perform formal experiments that deal with quantitative data.

3. Evaluating WebODE

WebODE is a scalable workbench for ontological engineering that provides services for editing and browsing ontologies, importing and exporting ontologies to classical and semantic web languages, evaluating ontologies, mapping ontologies, etc. [2].

Because of the lack of work that deals with evaluating the performance of ontology development tools, the motivations for carrying out this study have been:

- To define a **method** for evaluating WebODE's performance.
- To obtain a **benchmark suite** for evaluating WebODE.
- To **evaluate** WebODE using the benchmark suite.

All these motivations converge in a single long-term goal: to achieve a **continuous improvement** in the platform's quality.

The method we have used to evaluate the performance of WebODE is composed of the following steps:

- To identify the evaluation goals, elements, and metrics.
- To design and implement the benchmark suite.
- To run the benchmark suite.

- To analyze the results obtained after running the benchmark suite.

4. Identification of the Evaluation Goals, Elements, and Metrics

In order to identify the elements and metrics that will be considered in the evaluation we have chosen the Goal/Question/Metric (GQM) paradigm[1] [16]. The idea beyond the GQM paradigm is that any software measurement activity should be preceded by the identification of a software engineering goal, which leads to questions, which in turn lead to actual metrics.

The WebODE ontology management API provides methods to insert, update, remove, and query the components of the WebODE knowledge model[2]. As the services provided by WebODE use these methods for accessing WebODE ontologies, the performance of these services strongly depends of the performance of the API methods. Therefore, our goal is to **evaluate the performance of the methods provided by the WebODE ontology management API**. Table 1 presents the questions and the metrics derived from this goal according to the GQM paradigm. The analysis of the results of executing the benchmark suite will provide answers to these questions.

Table 1. Questions and metrics obtained through the GQM approach

Question	**Metric**
Q1: Which is the actual performance of the WebODE API methods?	Execution time of each method
Q2: Is the performance of the methods stable?	Variance of the execution times of each method
Q3: Are there any anomalies in the performance of the methods?	Percentage of execution times out of range in each method's sample
Q4: Do changes in a method's input parameters affect its performance?	Execution time with parameter A = Execution time with parameter B
Q5: Does WebODE's load affect the performance of the methods?	WebODE's load versus execution time relationship

In summary, the elements to evaluate are the **72 methods** of the WebODE ontology management API, and the metric to use is the **execution time** of the methods over incremental load states.

5. Design and Implementation of the Benchmark Suite

Several authors have enumerated the desirable properties of a benchmark suite [17, 18, 19]: generality, representativeness, transparency, interpretability, robustness, scalability, portability, accessibility, and repeatability. These properties have been the basis of the requirements definition for the benchmark suite.

[1] Other approaches are Quality Function Deployment [20] and Software Quality Metrics [21].
[2] http://kw.dia.fi.upm.es/wpbs/WebODE_API_methods.html

5.1. Requirements for the Benchmark Suite

In order to have a **generic** and **representative** benchmark suite, the benchmarks that compose it use every WebODE API method, performing current operations over WebODE ontologies.

To be able to compare each method's execution time, the methods must be executed under the same conditions. Therefore, we have defined the execution environment and the load state of WebODE, having fixed both before running each benchmark.

The benchmarks and their results must be **transparent** and **interpretable**. Each benchmark executes just one method and stores the wall clock time elapsed while running the method. The only other operation performed by a benchmark is to restore the load state of WebODE in case it changed during the benchmark execution.

Furthermore, as each method has its input parameters, one or more benchmarks have been defined for each method according to variations in these input parameters. So, from the 72 API methods we got **128 benchmarks**[3].

As the benchmarks must be **robust** and **scalable**, they have been parameterized according to two parameters:

- **Load factor (X).** The load factor of WebODE's load state when executing a benchmark.
- **Number of iterations (N).** The number of consecutive executions of a method in a single benchmark. This parameter defines the number of sample measurements obtained after executing a benchmark.

For example, from the method *insertTerm(String ontology, String term, String description)*, that inserts concepts in an ontology, we defined two benchmarks regarding the different input values of the method. These benchmarks were parameterized according to the load factor (X) and the number of executions (N):

- benchmark1_1_08. It inserts X concepts in an ontology. This is repeated N times.
- benchmark1_1_09. It inserts 1 concept in X ontologies. This is repeated N times.

In order to have a **portable** benchmark suite, the benchmarks have been implemented in Java, using only standard libraries and with no graphical components.

The benchmark suite must also be **accessible** and **repeatable**, anyone should be able to replicate the experiments and achieve the same conclusions. Therefore, the benchmark suite source code and the results obtained in this evaluation are published in a public web page[4].

5.2. Definition of the Execution Environment

As the workload used in the evaluation must be characterized accurately [22], we have defined the execution environment with the variables that influence the execution time of a method: hardware configuration, software configuration, computer's load, and WebODE's load.

The WebODE's load variable has been the only one whose values have been altered. As we are not interested in the other three variables' effect in the execution times, these variables have taken fixed values during the execution of the benchmarks. Furthermore, in order to avoid other non-controlled variables that may

[3] http://kw.dia.fi.upm.es/wpbs/WPBS_benchmark_list.html
[4] http://kw.dia.fi.upm.es/wpbs/

affect the results, the computer used for executing the methods has been isolated: without network connection nor user interaction. Next, we define these variables and the values that they took when running the benchmarks.

- **Hardware configuration.** It is the hardware configuration of the computer where WebODE is running. The computer was a Pentium 4 2.4 Ghz monoprocessor with 256 Mb. of memory.
- **Software configuration.** It is the configuration of the operating system and of the software needed to execute WebODE. It was the following, using each system's default configuration: Windows 2000 Professional Service Pack 4; SUN Java 1.4.2_03 (the benchmarks were compiled with the default options); Oracle version 8.1.7.0.0 (the Oracle instance's memory configuration was changed to: Shared pool 30 Mb., Buffer cache 80 Mb., Large pool 600 Kb., and Java pool 32 Kb.); Minerva version 1 build 4; and WebODE version 2 build 8.
- **Computer's load.** It is the load of the computer where WebODE is running. This load was the corresponding to the computer just powered on, with just the programs and services needed to run the benchmarks.
- **WebODE's load.** It is the underlying database's load where WebODE ontologies are stored. The generation of this load is explained in the next section.

5.3. Workload Generation

We mentioned before that WebODE's load state must be the same for every benchmark execution. This common load state must also allow to execute the benchmarks with different load factors (X) and with no errors. Therefore, WebODE's initial load state has been worked out from each benchmark's execution needs.

Each **benchmark's minimum load state** has been defined as the minimum ontology components that must exist in WebODE in order to execute the benchmark with no errors. For example, considering the four benchmarks whose methods insert and remove concepts in an ontology, Table 2 shows each of these benchmark's minimum load state, being X the load factor, and Table 3 shows the minimum load state of the four benchmarks.

Table 2. Minimum load states of the benchmarks whose methods insert and remove concepts

Benchmark	Operation	Minimum load state
benchmark1_1_08	Inserts X concepts in an ontology	1 ontology
benchmark1_1_09	Inserts a concept in X ontologies	X ontologies
benchmark1_3_20	Removes X concepts from an ontology	1 ontology with X concepts
benchmark1_3_21	Removes a concept from X ontologies	X ontologies with one concept

Table 3. The minimum load state of the benchmarks shown in Table 2

Benchmarks	Minimum load state
benchmark1_1_08, benchmark1_1_09, benchmark1_3_20, and benchmark1_1_21	1 ontology with X concepts and X ontologies with1 concept

Therefore, the **benchmark suite initial load state**[5], used when executing all the benchmarks, has been defined as the union of all the benchmarks' minimum load states, and is composed of all the ontology components that must exist in WebODE in order to execute every benchmark with no errors.

[5] http://kw.dia.fi.upm.es/wpbs/WPBS_workload_generation.html

6. Execution of the Benchmark Suite

The 128 benchmarks that compose the benchmark suite have been run ten times with increasing load factors (X=500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, and 5000) and with a number of iterations (N) of 400.

With the aim of checking that 400 iterations is a valid sample size, we have run several benchmarks with higher and lower number of iterations and we have confirmed that the results obtained are virtually equivalent. We haven't used a higher sample because the slight precision improvement would mean a much higher duration of the benchmark suite execution.

As with a load factor (X) of 5000 we obtained enough data to be able to determine the methods' performance, the benchmarks haven't been executed with higher load factors.

The results of running a benchmark are N (in this case 400) measurements of the execution time of the method used in a benchmark. These results are stored in a text file in a hierarchical measurement data library, so as to access them easily.

7. Analysis of the Results

The conclusions obtained when analyzing the results of a benchmark suite execution are usually temporarily limited [23]. As the API methods will undergo changes, the results just inform about WebODE's current performance, not its future one.

In order to obtain information that can be used to make decisions, we must apply statistical analysis techniques to the execution results [24]. From the N measurements of the execution time of a method in a benchmark we can obtain:

- **Graphs** that show the behavior of the execution time.
- **Statistical values** worked out from the sample of N measurements.

Observing the graphs of the execution times measured in a benchmark, we saw that execution times are usually constant. One example of this can be seen in Figure 1, where the execution times of benchmark1_3_20 when running the method *removeTerm* 400 times with a load factor of 5000 are shown.

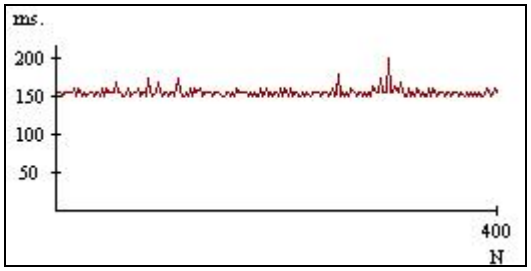


Figure 1. Execution times of benchmark1_3_20 with X=5000 and N=400

The next issue was to find which statistical values could be used to describe the execution times' samples. First of all, we ran normality tests over the measurements obtained after executing the benchmarks. As the distributions of the measurements were non-normal, we could not rely on usual values like mean and standard deviation for describing these measurements. Therefore, we used robust statistical values like

the **median**, the **upper and lower quartiles**, and the **interquartile range** (upper minus lower quartile).

To obtain the number of measurements out of range, we calculated the **percentage of outliers** in each sample. The traditional method is to consider as potential outlier values the measurements beyond the upper and lower quartiles adding and subtracting respectively 1.5 times the interquartile range [25]. As the Java method used for measuring time (*java.lang.System.currentTimeMillis()*) in the Windows platform has a precision of tens of milliseconds, in the results we frequently encountered interquartile ranges of zero milliseconds. This caused to be considered as outliers every determination that differed from the median. With the objective of fixing this precision fault, we have augmented the interquartile range when calculating the outliers to include half the minimal granularity (5 milliseconds) in both boundaries.

An example of the statistical values obtained can be seen in Table 4. This table shows the values obtained for the four benchmarks whose methods insert and remove concepts in an ontology. All the statistical values and graphs can be looked up and downloaded from the benchmark suite web page.

Table 4. Statistical values of the benchmarks whose methods insert and remove concepts

	Load	N	UQ	LQ	IQR	Median	% Outliers	Function
benchmark1_1_08	5000	400	60	60	0	60	1,25	y=62,0-0,0090x
benchmark1_1_09	5000	400	912	901	11	911	1,75	y=910,25-0,0030x
benchmark1_3_20	5000	400	160	150	10	150	1,25	y=155,25-0,0030x
benchmark1_3_21	5000	400	160	150	10	151	0,25	y=154,96-0,0010x

Next, we show the conclusions drawn after analyzing the data, answering the questions previously proposed in Table 1.

7.1. Finding out Methods' Performance

In order to be able to clearly differentiate the execution times, we have analyzed the data obtained from running the benchmarks with the maximum load factor used, X=5000, and with a number of iterations (N) of 400. We use the median of the execution times of a method in a benchmark as an indicator of its **performance**.

The medians of the execution times of all the API methods range from 0 to 1051 milliseconds. Figure 2 shows the histogram of these medians, where we clearly see a group of values higher than the rest. The execution times of this group belong to 12 benchmarks that execute 8 methods (as different benchmarks have been defined for each method). These 8 methods, with a median execution time higher than 800 ms., have been selected for the improvement recommendations. The rest of the methods have median execution times lower than 511 ms., being most of them around 100 ms.

Bearing in mind the kind of operation that the methods carry out (inserting, updating, removing, or selecting an ontology component), we did not find significant differences between the performances of each kind of method.

Taking into account what kind of element of the knowledge model a method manages (concepts, instances, class attributes, instance attributes, etc.), in the slowest methods' group are present almost every method that manages relations between concepts. Methods that manage instance attributes also have high execution times, and the rest of the methods behave similarly, only standing out the methods that manage imported terms and references as being the ones with lower execution times.

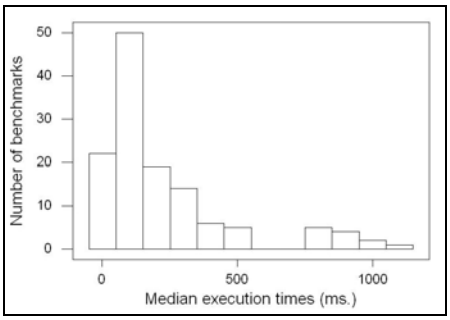


Figure 2. Histogram of the medians of the execution times

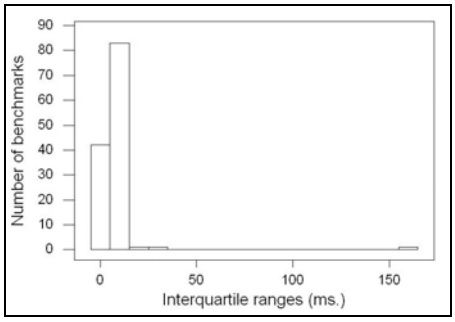


Figure 3. Histogram of the interquartile ranges of the execution times

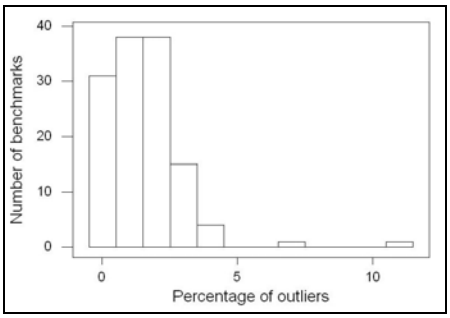


Figure 4. Histogram of the percentage of outliers of the execution times

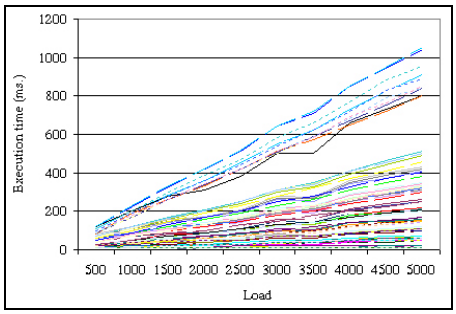


Figure 5. Evolution of the execution times when increasing WebODE's load

Regarding the **spread** of the execution times of the methods, we analyzed the interquartile range (IQR) of the execution times of the methods. Figure 3 shows the histogram of the IQRs of the execution times. Almost every method has an IQR from 0 to 11 ms. Having into account that the granularity of the measurements is of 10 milliseconds, we can state that the execution times have a low spread. The only exceptions are *removeTermRelation* (benchmark1_3_09) with an IQR of 19, *addValueToClassAttribute* (benchmark1_1_14) with an IQR of 30, and *getAvailableOntologies* (benchmark1_4_01) with an IQR of 160. This last method has been selected for the improvement recommendations due to its atypical IQR value.

In order to detect **anomalies**, we generated the histogram of the percentage of outliers in the execution times of the methods, shown in Figure 4. Most of the benchmarks have from 0 to 3.75% of outliers. These values confirm the lack of anomalies except the peaks in the execution times that can be seen in the graphs. The only methods to emphasize are *openOntology* (benchmark2_01), with 11.5% of outliers, and *addValueToClassAttribute* (benchmark1_1_15), with 7% of outliers. These two methods have been selected for the improvement recommendations.

Studying whether **changes in a method's parameters** affect its performance, we have observed that in 21 methods the performance varies when changing its input parameters. This variation is lower than 60 milliseconds except in five methods that show a difference in their execution times when changing parameters from 101 to 851 ms., and have been selected for the improvement recommendations.

7.2. Establishing Load-Performance Relationship

To study WebODE's load effect in performance, we analyzed the medians of the execution times of the methods from a minimum load state (X=500) to a maximum load state (X=5000), and with a number of iterations (N) of 400. We estimated the function that these medians define by **simple linear regression** and considered its **slope** in order to examine the relationship between the load and the execution time of the methods.

Figure 5 shows the plot of every benchmark's median execution time with the different load factors. As can be seen, the 8 methods whose execution times are higher than the rest are also the methods whose performance is more influenced by the load. To be precise, the slope of these methods' function is greater than 0.15, and the slope of the rest of the methods' function ranges from 0 to 0.1. As we stated before, these methods have been selected for the improvement recommendations.

7.3. Development of Improvement Recommendations

Once the data has been analyzed, the next step is the development of the improvement recommendations. These recommendations include those methods whose execution times:

- Have a median execution time higher than 800 ms.
- Have an interquartile range greater than 150 ms.
- Have more than a 5% of outlier values.
- Vary more than 100 ms. when modifying its input parameters.
- Increase when augmenting load with a slope greater than 0.15.

Table 5 shows the 12 of the 72 WebODE's API methods included in the improvement recommendations, and the reasons for their inclusion.

Table 5. Methods in the improvement recommendations

	Execution time > 800 ms.	Interquartile range > 150 ms.	Outlier values > 5%	Execution time variation > 100 ms.	Slope when increasing load > 0.15
removeTermRelation	X				X
getInheritedTermRelations	X				X
insertTerm	X			X	X
insertRelationInstance	X			X	X
openOntology	X		X		X
getAdHocTermRelations	X				X
getTermRelations	X				X
getAvailableOntologies	X	X			X
addValueToClassAttribute			X		
insertConstant				X	
updateSynonym				X	
getInstances				X	

8. Conclusions and Future Work

In this paper we have set out the method we followed to develop a benchmark suite for assessing the temporal performance of the WebODE ontology engineering

workbench. We have also stated how we executed this benchmark suite and the main conclusions obtained after analyzing the collected results.

The main achievement obtained after performing this study is that we have precisely determined WebODE's performance, identifying:

- The slowest methods.
- The methods with high spreaded execution times.
- The methods with anomalies in their execution times.
- The effect of changing a method's parameters in its performance.
- The link between WebODE's load and its methods' performance.

The analysis of the results showed that changes in a method's input parameters significantly affected its performance. This fact must be taken into account when defining benchmark suites, either for WebODE or for other systems.

Besides being able to **evaluate** WebODE's performance, the benchmark suite that we have developed will allow us to:

- **Monitor** WebODE, being able to observe the performance of critical elements and how changes in the platform affect its performance.
- **Diagnose** future problems in WebODE.

The benchmarks that compose the proposed suite just execute one method and store its execution time. One way of improving the study could be making these benchmarks execute different kinds of synthetic requests to WebODE, in order to study not just the stability of the methods but the stability of the whole platform.

Once we know the performance of WebODE regarding its ontology management API, we could work out the approximate performance of the services and applications that use this API. This could be done either by empirically obtaining the frequency of use of real services and applications or by defining use frequencies for each different kind of application.

In this work we have just evaluated the execution efficiency of the WebODE API methods. There are many other WebODE attributes that we would be interested in measuring like reliability, usability, or functionality (to cite just a few samples); and future studies will focus in them.

Although this benchmark suite has been designed specifically for WebODE, we plan to extend it to other ontology engineering platforms (KAON, OntoEdit, Protégé-2000, etc.). This could be done either by finding commonalities between the ontology management APIs of the different platforms or by means of a common management API such as OKBC [26].

Acknowledgments

This work is partially supported by the IST project KnowledgeWeb (IST-2004-507482) and by the IST project Esperonto (IST-2001-34373).

References

1. [Fernández-López et al., 1997] M. Fernández-López, A. Gómez-Pérez, N. Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Spring Symposium on Ontological Engineering of AAAI. Stanford University, California, 1997, pp 33-40.

2. [Arpírez et al., 2003] J.C. Arpírez, O. Corcho, M. Fernández-López, A. Gómez-Pérez. WebODE in a nutshell. AI Magazine. 24(3), Fall 2003, pp. 37-47.
3. [IEEE, 1991] IEEE-STD-610 ANSI/IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. February 1991.
4. [Duineveld et al., 1999] A.J. Duineveld, R. Stoter, M.R. Weiden, B. Kenepa, and V.R. Benjamins. Wondertools? a comparative study of ontological engineering tools. In Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'99), Banff, Canada, 1999. Kluwer Academic Publishers.
5. [Ontoweb, 2002] Ontoweb deliverable 1.3: A survey on ontology tools. Technical report, IST OntoWeb Thematic Network, May 2002.
6. [Angele and Sure, 2002] J. Angele and Y. Sure (eds.). Evaluation of ontology-based tools. In Proceedings of the 1st International Workshop EON2002, Sigüenza, Spain, September 2002. CEUR-WS.
7. [Lambrix et al., 2003] P. Lambrix, M. Habbouche, and M. Pérez. Evaluation of ontology development tools for bioinformatics. Bioinformatics, 19(12):1564-1571, 2003.
8. [Stojanovic and Motik, 2002] L. Stojanovic and B. Motik. Ontology evolution within ontology editors. In Proceedings of the International Workshop on Evaluation of Ontology-based Tools (EON2002), Sigüenza, Spain, October 2002.
9. [Sofia Pinto et al., 2002] H. Sofia Pinto, Duarte Nuno Peralta, and Nuno J. Mamede. Using Protégé-2000 in reuse processes. In Proceedings of the International Workshop on Evaluation of Ontology-based Tools (EON2002), Sigüenza, Spain, October 2002.
10. [Corcho et al., 2003] O. Corcho, A. Gómez-Pérez, D.J. Guerrero-Rodríguez, D. Pérez-Rey, A. Ruiz-Cristina, T. Sastre-Toral, and M.C. Suárez-Figueroa. Evaluation experiment of ontology tools' interoperability with the WebODE ontology engineering workbench. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
11. [Isaac et al., 2003] A. Isaac, R. Troncy, and V. Malais. Using XSLT for interoperability: DOE and the travelling domain experiment. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
12. [Calvo and Gennari, 2003] F. Calvo and J.H. Gennari. Interoperability of Protégé 2.0 beta and OilEd 3.5 in the domain knowledge of osteoporosis. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
13. [Fillies, 2003] C. Fillies. Semtalk EON2003 semantic web export / import interface test. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
14. [Knublauch, 2003] H. Knublauch. Case study: Using Protégé to convert the travel ontology to UML and OWL. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
15. [Gómez-Pérez and Suárez-Figueroa, 2004] A. Gómez-Pérez and M.C. Suárez-Figueroa. Evaluation of RDF(S) and DAML+OIL import/export services within ontology platforms. In Proceedings of the Third Mexican International Conference on Artificial Intelligence, pages 109-118, Mexico City, Mexico, April 2004.
16. [Basili et al., 1994] V.R. Basili, G. Caldiera, D.H. Rombach. The Goal Question Metric Approach. Encyclopedia of Software Engineering, 2 Volume Set Willey, 1994, pp 528-532.
17. [Bull et al., 1999] J.M. Bull, L.A. Smith, M.D. Westhead, D.S. Henty, R.A. Davey. A Methodology for Benchmarking Java Grande Applications. EPCC, June 1999.
18. [Shirazi et al., 1999] B. Shirazi, L. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, E. Huh. DynBench: A Dynamic Benchmark Suite for Distributed Real-Time Systems. IPDPS 1999 Workshop on Embedded HPC Systems and Applications, San Juan, Puerto Rico, April 1999.

19. [Sim et al., 2003] S. Sim, S. Easterbrook, and R. Holt. Using benchmarking to advance research: A challenge to software engineering. In Proceedings of the 25th International Conference on Software Engineering (ICSE'03), pages 74-83, Portland, OR, 2003.
20. [Dean, 1992] E.B. Dean. Quality Function Deployment for Large Systems. Proceedings of the 1992 International Engineering Management Conference, Eatontown NJ, October 1992, pp 317-321.
21. [Boehm et al., 1976] B.W. Boehm, J.R. Brown, and M. Lipow. Quantitative Evaluation of Software Quality. In Proceedings of the Second International Conference on Software Engineering. San Francisco, 1976, pp 592-605.
22. [Dongarra et al., 1987] J. Dongarra, J.L. Martin, J. Worlton. Computer benchmarking: paths and pitfalls. IEEE Spectrum, Vol. 24, N. 7, July 1987, pp 38-43.
23. [Gray, 1993] J. Gray. The Benchmark Handbook for Database and Transaction Systems (2nd Edition). Morgan Kaufmann, 1993.
24. [Fenton, 1991] N.E. Fenton. Software Metrics A Rigorous Approach. Chapman & Hall, London, UK, 1991.
25. [Mendenhall and Sincich, 1995] W. Mendenhall and T. Sincich. Statistics for Engineering and the Sciences, 4th Edition. Englewood Cliffs, NJ. Prentice Hall, 1995.
26. [Chaudri et al., 1997] V.K. Chaudhri, A. Farquhar, R. Fikes, P.D. Karp, J.P. Rice. The Generic Frame Protocol 2.0. Technical Report, Stanford University, 1997.

A Formalization of Ontology Learning From Text

Michael Sintek[1], Paul Buitelaar[2], and Daniel Olejnik[2]

[1] DFKI GmbH
Knowledge Management Department
Erwin-Schrödinger-Str., Bldg. 57
D-67663 Kaiserslautern, Germany
sintek@dfki.uni-kl.de
[2] DFKI GmbH
Language Technology Department
Stuhlsatzenhausweg 3
D-66123 Saarbrücken, Germany
{paulb,olejnik}@dfki.de

Abstract. Recent developments towards knowledge-based applications in general and Semantic Web applications in particular are leading to an increased interest in ontologies and in dynamic methods for developing and maintaining them. As human language is a primary mode of knowledge transfer, ontology learning from relevant text collections has been among the most successful strategies in this work. Such methods mostly combine a certain level of linguistic analysis with statistical and/or machine learning approaches to find potentially interesting concepts and relations between them. Here, we discuss a formalization of this process (in the specific context of the OntoLT tool for ontology learning from text) in order to arrive at a better definition of this task, which we hope to be of use in a more principled comparison of different approaches. As ontology representation formalisms we will consider those that have a model-theoretic semantics, with OWL (and subsets of OWL) being appropriate candidates.

Keywords: ontologies, ontology learning from text, linguistic analysis, OWL, formalization, ontology entailment, model-theoretic semantics

1 Motivation

Recent developments towards knowledge-based applications such as Intelligent Question-Answering, Semantic Web Services and Semantic-Level Multimedia Search are also leading to an increased interest in ontologies. Additionally, as ontologies are domain descriptions that tend to evolve rapidly over time and between different applications, there has been an increased interest also towards developing and maintaining ontologies dynamically (see also [1]).

As human language is a primary mode of knowledge transfer, ontology learning from relevant text collections has been among the most successful strategies

in this work. See, for instance, the overview of ontology learning systems and approaches as discussed by the OntoWeb deliverable 1.5 [2]. Some recent examples of systems for ontology learning from text are: ASIUM [3], TextToOnto [4], Ontolearn [5], OntoLT [6]. All of these combine a certain level of linguistic analysis with statistical and/or machine learning approaches to find potentially interesting concepts and relations between them.

In order to allow a principled comparison of these approaches and to define evaluation environments, we propose a formalization of ontology (and knowledge base) learning from text. Since the systems we want to concentrate on mainly use ontology representation languages that are frame systems or (subsets of) description logics like OWL [7], we developed a formalization approach which is suited for this class of ontology languages, namely those that have a model-theoretic semantics which we use as a basis for operations on ontologies. The OntoLT [6] tool for ontology learning from text will be used to clarify the various details of this formalization.

2 Ontology Learning From Text

A typical approach in ontology learning from text first involves term extraction from a domain-specific corpus through a statistical process that determines their relevance for the domain corpus at hand. These are then clustered into groups with the purpose of identifying a taxonomy of potential classes. Subsequently, relations can be identified by computing a statistical measure of "connectedness" between identified clusters.

In the context of this paper we assume a similar approach, but we additionally aim at a more direct connection between ontology learning and linguistic analysis. Through such an approach, relations may be identified additionally on the basis of linguistic analysis of the "dependency structure" between terms and connecting or modifying words (i.e., verbs, prepositions, adjectives) in their context.

2.1 OntoLT

The OntoLT plug-in for Protégé implements this approach through the definition of mapping rules with which classes and properties can be extracted automatically from linguistically annotated text collections. Through the use of such rules, linguistic knowledge (context words, morphological and syntactic structure, etc.) remains associated with the constructed ontology and may be used subsequently in its application and maintenance, e.g., in knowledge markup, ontology mapping, and ontology evolution.

The ontology extraction process is implemented as follows. OntoLT provides a precondition language, with which the user can define mapping rules. Preconditions are implemented as XPath [8] expressions over XML-based linguistic annotation of relevant text collections (see Section 2.2 below). If all constraints

are satisfied, the mapping rule activates one or more operators that describe in which way the ontology should be extended if a candidate is found.

Predefined preconditions select for instance the predicate of a sentence, its linguistic subject or direct object. Preconditions can also be used to check certain conditions on these linguistic entities, for instance if the subject in a sentence corresponds to a particular lemma (the morphological stem of a word).

Selected linguistic entities may be used in constructing or extending an ontology. For this purpose, OntoLT provides operators to create classes, slots and instances. According to which preconditions are satisfied, corresponding operators will be activated to create a set of candidate classes and slots that are to be validated by the user (see Fig. 1). Validated candidates are then integrated into a new or existing ontology.

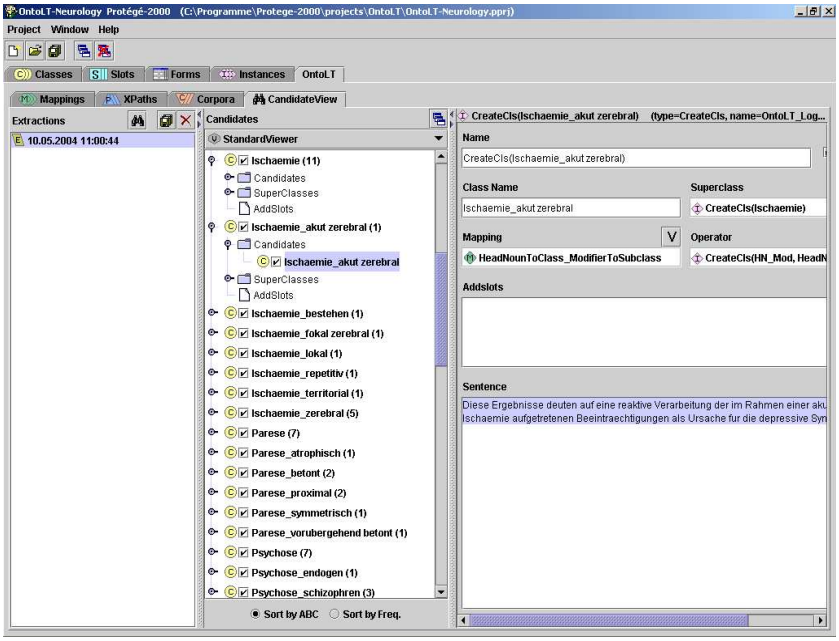


Fig. 1. OntoLT Screenshot

For example, OntoLT includes the following two mapping rules:

- HeadNounToClass_ModToSubClass: maps a head-noun to a class and in combination with its modifier(s) to one or more sub-class(es)
- SubjToClass_PredToSlot_DObjToRange: maps a linguistic subject to a class, its predicate to a corresponding slot for this class and the direct object to the range of this slot

Consider the following sentence to which these rules can be applied:

This disease is characterized primarily by impaired mental function caused by damage to the brain.

Result for the HeadNounToClass_ModToSubClass rule:
Class('impaired mental function') subClassOf Class('function')
Result for the SubjToClass_PredToSlot_DObjToRange rule:
Property('characterize') domain Class('function') range Class('disease')

2.2 Linguistic Analysis and Annotation

We consider linguistically annotated text corpora, using an XML-based annotation format, which integrates multiple levels of linguistic and semantic analysis in a multi-layered DTD with each analysis level (e.g., morphological, syntactic and dependency structure) organized as a separate track with options of reference between them via indices.

Linguistic annotation is currently provided by SCHUG, a rule-based system for German and English analysis [9] that implements a cascade of increasingly complex linguistic fragment recognition processes. SCHUG provides annotation of part-of-speech (through integration of TnT [10]), morphological inflection and decomposition (based on Mmorph [11]), phrase and dependency structure (head-complement, head-modifier and grammatical functions).

In Fig. 2, we present a tree representation of the linguistic annotation for (part of) the following sentence (German with translation in English):

An 40 Kniegelenkpräparaten wurden mittlere Patellarsehnendrittel mit einer neuen Knochenverblockungstechnik in einem zweistufigen Bohrkanal bzw. mit konventioneller Interferenzschraubentechnik femoral fixiert.

(In 40 human cadaver knees, either a mid patellar ligament third with a trapezoid bone block on one side was fixed on the femoral side in a 2-diameter drill hole, or a conventional interference screw fixation was applied.)

The linguistic annotation for this sentence consists of part-of-speech and lemmatization information in the `<text>` level, phrase structure (including head-modifier analysis) in the `<phrases>` level and grammatical function analysis in the `<clauses>` level (in this sentence there is only one clause, but more than one clause per sentence is possible).

Part-of-speech information consists of the correct syntactic class (e.g., noun, verb) for a particular word given its current context. For instance, the word *works* will be either a verb (working the whole day) or a noun (all his works have been sold).

Morphological information consists of inflectional, derivational or compound information of a word. In many languages other than English the morphological system is very rich and enables the construction of semantically complex compound words. For instance the German word *Kreuzbandverletzung* corresponds in

English with three words: *cruciate ligament injury*. Phrase structure information consists of an analysis of the syntactic structure of a sentence into constituents that are headed by an adjective, a noun or a preposition. Additionally, the internal structure of the phrase will be analyzed and represented, which includes information on modifiers that further specify the head. For instance, in the nominal phrase *neue Technik* (*new technology*) the modifier *neu* further specifies the head *Technik*.

Clause structure information consists of an analysis of the core semantic units (clauses) in a sentence with each clause consisting of a predicate (mostly a verb) with its arguments and adjuncts. Arguments are expressed by grammatical functions such as the subject or direct object of a verb. Adjuncts are mostly prepositional phrases, which further specify the clause. For instance, in *John played football in the garden*, the prepositional phrase in the garden further specifies the clause "play (John, football)."

All such information is provided by the annotation format that is illustrated in Fig. 2. For instance, the direct object (`DOBJ`) in the sentence above covers the nominal phrase `p2`, which in turn corresponds to tokens `t5` to `t10` (*mittlere Patellarsehnendrittel mit einer neuen Knochenverblockungstechnik*). As token `t6` is a German compound word, a morphological analysis is included that corresponds to lemmas `t6.l1`, `t6.l2`, `t6.l3`.

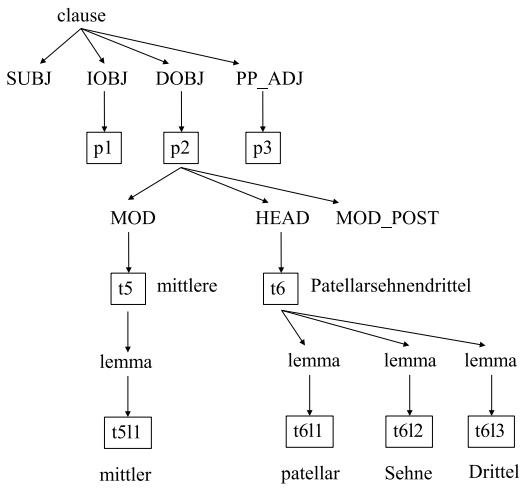


Fig. 2. Linguistic Annotation Example

3 Formalizing Ontology Learning From Text

In this section, we will describe our general approach for formalizing ontology learning from text and instantiate this for the OntoLT system. The general

approach and the OntoLT system do not only apply to ontologies, but also to knowledge bases (i.e., ontologies together with instances), but we will mainly concentrate on ontology learning in the following.

For describing the problem of ontology learning from text, we consider a suggestion function σ which maps a text corpus C, an ontology O, and (background) knowledge K to suggestions S:

$$\sigma : \ C \times O \times K \ \rightarrow \ S$$

The background knowledge will often contain mapping specifications that map a part of the corpus (e.g., one sentence) to some suggestions. It might also contain results of statistical analyses of the text corpus (like the distribution of nouns), thesaurus information (like WordNet), etc.

The suggestions will usually be a (possibly weighted) set of operations changing the ontology. They will be presented to the user who can select, possibly correct, and then apply them to the old (possibly empty) ontology. This process can be repeated as new documents come in, or as new background knowledge has been defined.

3.1 Operations on Ontologies

In order to specify suggestions, we will consider a minimal set of operations on ontologies, namely + and −:

$$+ : \ O \times O \ \rightarrow O$$

$$- : \ O \times O \ \rightarrow O$$

Their semantics will be defined without referring to concrete ontology languages, which allows our formalization to be applied to a wide range of systems. The only thing we require is that the ontology language has a model-theoretic semantics. This approach is radically different from approaches like ontology algebrae (e.g., [12]) which seem not to be applicable to such a wide range of ontology languages as our approach.

We define the semantics of + and − with the help of ontology *entailment*. An ontology O_1 is said to *entail* an ontology O_2, written $O_1 \models O_2$, if every model for O_1 is also a model for O_2 (cf. [13]).

Two ontologies O_1 and O_2 are said to be semantically equal ($O_1 \doteq O_2$) iff $O_1 \models O_2$ and $O_2 \models O_1$.

An ontology O is said to be a *most general* ontology for a condition C if O fulfills C and there exists no other ontology $O' \not\doteq O$ which fulfills C and for which $O \models O'$ holds. Note that, in general, more than one most general ontology for a condition exists. A *least general* ontology for a given condition is specified analogously.

We now define + and − as follows:

$O_1 + O_2$ is a most general ontology O with $O \models O_1 \ \wedge O \models O_2$.
$O_1 - O_2$ is a least general ontology O with $O_1 \models O \ \wedge O \not\models O_2$.

Note that, in general, the result of $O_1 + O_2$ and $O_1 - O_2$ is not well-defined, depending on the choice of the ontology language. Furthermore, + and − are not symmetric: + adds *all* of O_2 to O_1, while − removes only a minimal portion of O_2 from O_1. While + is usually well-defined since most ontology languages allow the statements that are used to define ontologies to be joined in some simple way, − is usually not well-defined, thus leaving several choices to the user. This does, however, not cause any problems in our scenario since the user has to interact with the suggestions anyway.

3.2 Ontology Language

As ontology language, we consider in OntoLT a simple subset of standard description logics where we only allow "subclass of" axioms $C \sqsubseteq CE$ where C is a class name and CE a class expression that uses only class names, intersection ($\sqcap$), and range restrictions ($\forall P.C$). $\top$ is the superclass of all classes.

Because of $\{C \sqsubseteq C_1 \sqcap C_2\} \doteq \{C \sqsubseteq C_1,\ C \sqsubseteq C_2\}$ we will in the following only consider axioms without intersection, i.e., we allow only axioms of the form $C_1 \sqsubseteq C_2$ and $C_1 \sqsubseteq \forall P.C_2$, where C_i are class names and P is a property name.

This results in a description logics that is the subset of OWL Lite [7] with the expressiveness of frame systems like RDF Schema [14] and Protégé [15]

In the OntoLT system, we also take instances into account. The extensions necessary for this are straight-forward and will not be further described here.

Note that the proposed formalization allows the use of any other ontology (or knowledge-base) language as long as it has a model-theoretic semantics,

3.3 Suggestions

For suggestions, we consider sequences of ontology operations, written as $\pm\{a_1, a_2, \ldots\}, \{b_1, b_2, \ldots\}, \ldots$ where $\pm$ is either + or − and a_i and b_i are axioms as described above. For $\pm\{a\}$, we also write $\pm a$.

The current implementation of OntoLT only handles suggestions for ontology *extensions*, so we only need the + operation. For ontology languages based on description logics, where an ontology is a set of axioms, + directly maps to set union $\cup$:

$$O_1 + O_2 \ \doteq O_1 \cup O_2$$

Note that even in the case of our simple ontology language, the − operator is not well-defined. Nevertheless, this would not be a big problem in a concrete tool, as the user could then be presented with several choices of how to remove a certain axiom from the ontology, as he will have to choose which suggestions to apply anyway.

3.4 Mapping Rules

OntoLT uses a set of mapping rules $R(\Sigma)$ that maps a *single* sentence (which is the parameter Σ for the rule set) from the text corpus to a set of suggestions. These rules are of the following form:

$$P \mapsto S$$

P is the precondition and expressed as a formula in FOL. S is a sequence of ontology operations as defined above, or, to allow a shorthand for alternatives, an expression of the form $S_1 \mid S_2 \mid \ldots$ where S_i is a sequence of operations. S will usually share some variables with P; these variables must be introduced with a common *forall* quantifier:

$$\forall V_1, \ldots, V_n \; P(V_1, \ldots, V_n) \mapsto S(V_1, \ldots, V_n)$$

We also allow the specification of Horn rules [16] (with FOL syntax) $H \leftarrow B$ that can be accessed in the precondition.

The predicate $\text{xpath}(X, E, V)$ applies an XPath [8] expression E to an XML fragment X and enumerates the results in V. This predicate is used to extract information from the corpus (by applying it on Σ).

The preconditions of the mapping rules and the bodies of the Horn rules may also access the ontology. For this, DL axioms ($C_1 \sqsubseteq C_2$, $C_1 \sqsubseteq \forall P.C_2$) can be used as literals.

In OntoLT, we also allow access to thesauri, in particular WordNet, via additional builtin predicates.

3.5 Enactment

For the enactment of rules, all mapping rules are transformed into Horn rules, as shown in the following table:

$P \mapsto S$	$\text{sugg}(S') \leftarrow P$
$P \mapsto S_1 \mid S_2 \mid \ldots$	$\text{sugg}(S'_1) \leftarrow P$ $\text{sugg}(S'_2) \leftarrow P$ $\ldots$

Here, S' is a representation of an ontology operation sequence S with FOL function symbols. Quantifiers and other syntactic constructs which are not allowed in normal Horn rules are removed with the Lloyd-Topor transformation [17], just as has been done in SiLRI [18] and TRIPLE [19], allowing the resulting rules to be enacted by a standard PROLOG engine (with some additional builtins).

A set of mapping rules $R(\Sigma)$ can now be enacted for a sentence s by executing the query $\forall S \leftarrow \text{sugg}(S)$ for the set of Horn rules obtained from the above transformation (and after replacing Σ with s in the rule set).

To evaluate the mapping rules for the whole text corpus C, we simply have to evaluate $R(\Sigma)$ for each sentence s from the text corpus, thus obtaining a set of suggestions that will be presented to the user.

3.6 Example

The following example are the rules for turning the subject of a sentence to a class and the predicate to a slot with the direct object as range:

$$
\begin{aligned}
&\forall S\ \text{subject}(S) \leftarrow \\
&\quad \text{xpath}(\Sigma, \text{"}\texttt{.//phrase[@id=...[@type="SUBJ"]/@phrase]/head}\text{"}, S) \\
&\quad \wedge\ \ldots\ . \\
&\forall P\ \text{predicate}(P) \leftarrow \\
&\quad \text{xpath}(\Sigma, \text{"}\texttt{.//phrase[@id=./../..//clause/@pred]}\text{"}, P) \\
&\quad \wedge\ \ldots\ . \\
&\forall O\ \text{directObject}(O) \leftarrow \\
&\quad \text{xpath}(\Sigma, \text{"}\texttt{.//phrase[@id=...[@type="DOBJ"]/@phrase]/head}\text{"}, O) \\
&\quad \wedge\ \ldots\ . \\
&\forall S, P, O\ \text{subject}(S) \wedge \text{predicate}(P) \wedge \text{directObject}(O) \mapsto \\
&\quad +\{S \sqsubseteq \top, O \sqsubseteq \top, S \sqsubseteq \forall P.O\}.
\end{aligned}
$$

The first three rules simply define how to find subjects, predicates, and direct objects in sentences with the help of some XPath expressions. The fourth rule generates the suggestion that S and O should become classes ($S \sqsubseteq \top, O \sqsubseteq \top$) with P a property on S with range O ($S \sqsubseteq \forall P.O\}$).

4 Conclusions

Automatic methods for text-based ontology learning have developed over recent years, see, e.g., the proceedings of the ECAI-2000[3], IJCAI-2001[4] and ECAI-2002[5] workshops on Ontology Learning. Still, a remaining challenge is to evaluate how useful or accurate the extracted ontologies are. In fact, we believe this to be a central issue as it is currently very hard to compare methods and approaches, due to the lack of a shared and formal understanding of the task at hand. By the work described in this paper, we hope to have contributed to such a shared understanding by providing a formal definition of the OntoLT approach to ontology learning from text. We believe that such a formal definition will allow for a better comparison with similar approaches, not so much on the level of specific methods, but rather on the level of preconditions, inputs, and results.

5 Acknowledgements

This research has in part been supported by EC grants IST-2000-29243 for the OntoWeb project and IST-2000-25045 for the MEMPHIS project, and by bmb+f grant 01 IMD01 A for the SmartWeb project.

[3] `http://ol2000.aifb.uni-karlsruhe.de/`
[4] `http://ol2001.aifb.uni-karlsruhe.de/home.html`
[5] `http://www-sop.inria.fr/acacia/WORKSHOPS/ECAI2002-OLT/`

References

1. Maedche, A.: Ontology Learning for the Semantic Web. Volume 665 of International Series in Engineering and Computer Science. Kluwer (2003)
2. Gomez-Perez, A., Manzano-Macho, D.: A survey of ontology learning methods and techniques. Technical Report Deliverable 1.5, OntoWeb Project (2003) Available from `http://ontoweb.aifb.uni-karlsruhe.de/`.
3. Faure, D., Nédellec, C., Rouveirol, C.: Acquisition of semantic knowledge using machine learning methods: The system ASIUM. Technical Report ICS-TR-88-16 (1998)
4. Maedche, A., Staab, S.: Semi-automatic engineering of ontologies from text. In: Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering. (2000)
5. Navigli, R., Velardi, P., Gangemi, A.: Ontology learning and its application to automated terminology translation. IEEE Intelligent Systems (2003)
6. Buitelaar, P., Olejnik, D., Sintek, M.: A Protégé plug-in for ontology extraction from text based on linguistic analysis. In: Proceedings of the European Semantic Web Symposium (ESWS). (2004)
7. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinnes, D., Patel-Schneider, P., Stein, L.: OWL web ontology language reference (2004) `http://www.w3.org/TR/owl-ref/`.
8. Clark, J., DeRose, S.: XML path language (XPath) (1999) `http://www.w3.org/TR/xpath`.
9. Declerck, T.: A set of tools for integrating linguistic and non-linguistic information. In: Proceedings of the SAAKM workshop at ECAI. (2002)
10. Brants, T.: TnT — a statistical part-of-speech tagger. In: Proceedings of the 6th ANLP Conference. (2000)
11. Petitpierre, D., Russell, G.: MMORPH — the Multext morphology program. Technical Report Deliverable for task 2.3.1, Multext Project (1995) Available from ISSCO, University of Geneva.
12. Mitra, P., Wiederhold, G.: An ontology-composition algebra. Springer Series: International Handbooks on Information Systems (2004) 93–113
13. Horrocks, I., Patel-Schneider, P.: Reducing OWL entailment to description logic satisfiability. In: Proceedings of the International Semantic Web Conference. (2003)
14. Brickley, D., Guha, R.: RDF vocabulary description language 1.0: RDF Schema (2004) `http://www.w3.org/TR/rdf-schema/`.
15. Knublauch, H.: An AI tool for the real world: Knowledge modeling with Protégé. JavaWorld (2003) Protégé is available from `http://protege.stanford.edu/`.
16. Lloyd, J.: Foundations of logic programming. Springer-Verlag New York, Inc. (1984)
17. Lloyd, J., Topor, R.: Making Prolog More Expressive. Journal of Logic Programming **3** (1984) 225–240
18. Decker, S., Brickley, D., Saarela, J., Angele, J.: A query and inference service for RDF. In: QL'98 — The Query Languages Workshop, Boston, USA, World-WideWeb Consortium (W3C) (1998)
19. Sintek, M., Decker, S.: TRIPLE—A query, inference, and transformation language for the Semantic Web. In: 1st International Semantic Web Conference (ISWC2002). (2002)

DODDLE-OWL: OWL-based Semi-Automatic Ontology Development Environment

Takeshi Morita[1], Yoshihiro Shigeta[1], Naoki Sugiura[1], Naoki Fukuta[1], Noriaki Izumi[2], and Takahira Yamaguchi[3]

[1] Shizuoka University, 3-5-1 Johoku, Hamamatsu, Shizuoka 432-8011, Japan, morita@ks.cs.inf.shizuoka.ac.jp, http://mmm.semanticweb.org
[2] National Institute of AIST, 2-41-6, Aomi, Koto-ku, Tokyo, Japan
[3] Keio University, 4-1-1 Hiyoshi, Kohoku-ku, Yokohama-shi, Kanagawa, Japan

Abstract. In this paper, we propose an ontology development support environment for the Semantic Web. The advantage of our environment is focusing the quality refinement phase of ontology construction. Through interactive support for refining the initial ontology, OWL-Lite level ontology, which consists of taxonomic relationships (class - sub class relationship) and non-taxonomic relationships (defined as property), is constructed effectively. The environment also provides semi-automatic generation of the initial ontology.

1 Introduction

The Semantic Web [1] is now gathering attentions from researchers in wide area. Adding semantics (meta-data) to the Web contents, software agents are able to understand and even infer Web resources. To realize such paradigm, the role of ontologies [2] is important in terms of sharing common understanding among both people and software agents [3]. In knowledge engineering field ontologies have been developed for particular knowledge system mainly to reuse domain knowledge. On the other hand, for the Semantic Web, ontologies are constructed in distributed places or domain, and then mapped each other. For this purpose, it is an important task to realize a software environment for rapid construction of ontologies for each domain. Towards the on-the-fly ontology construction, many researches are focusing on automatic ontology construction from existing Web resources, such as dictionaries, by machine processing with concept extraction algorithms. However, depending on domains (a law domain etc.), the important concepts which doesn't occur frequently in the resources may be required to be added by hand for ontology construction. In such a domain, if a user doesn't intervene, constructing ontologies cannot readily be done. Considering such situation, we believe that the most important aspect of the on-the-fly ontology construction is that how efficiently the user is able to complete making the ontology for the Semantic Web contents available to the public. For this reason, ontologies should be constructed not fully automatically, but through interactive support by software environment from the early stage of ontology construction.

Although it may seem to be contradiction in terms of efficiency, the total cost of ontology construction would become less than automatic construction since if the ontology is constructed with careful interaction between the system and the user, less miss-construction will be happened. It also means that high-quality ontology would be constructed.

In this paper, we propose a software environment for user-centered on-the-fly ontology construction named DODDLE-OWL (Domain Ontology rapiD DeveLopment Environment - OWL [4] extension). The architecture of DODDLE-OWL is re-designed based on DODDLE-II [5], the former version of DODDLE-OWL. DODDLE-OWL has the following five modules: Input Module, Construction Module, Refinement Module, Visualization Module, and Translation Module. Especially, to realize the user-centered environment, DODDLE-OWL dedicates to the Refinement Module. It enables us to develop ontologies with interactive indication of which part of ontology should be refined. DODDLE-OWL supports the construction of both taxonomic relationships and non-taxonomic relationships in ontologies. Since DODDLE-II has been built for ontology construction not for the Semantic Web but for typical knowledge systems, it needs some extensions for the Semantic Web such as OWL (Web Ontology Language) [4] export facility. DODDLE-OWL contributes the evolution of ontology construction and the Semantic Web.

2 The DODDLE-OWL Architecture

Figure 1 shows the overview of DODDLE-OWL. The main feature of DODDLE-OWL has the following five modules: Input Module, Construction Module, Refinement Module, Visualization Module, and Translation Module. The Input Module, the Hierarchy Construction Module, and the Hierarchy Refinement Module are included in DODDLE-I to support a user to construct taxonomic relationship. The Relationship Construction Module and the Relationship Refinement Module were added on DODDLE-II development that supports constructing both taxonomic and non-taxonomic relationships. The Visualization Module and the Translation Module are newly developed in DODDLE-OWL to extend functionalities for the Semantic Web such as exporting constructed ontology in OWL format. Here, we assume that there are one or more domain specific documents, and we also assume that the user can select important terms that are needed to construct a domain ontology. First, the user selects input concepts (important terms) in the Input Module. In the Construction Module, DODDLE-OWL generates the basis of the ontology, an initial concept hierarchy and set of concept pairs, based on input concepts by referring to WordNet [6] as an MRD (Machine Readable Dictionary) and documents. An initial concept hierarchy is constructed as an IS-A hierarchy of terms. Set of concept pairs are extracted by using co-occurrency based statistic methods. These pairs are considered to be closely related and that will be used as candidates to refine and add non-taxonomic relations. The user identifies some relationship between concepts in the pairs. In the Refinement Module, the initial ontology produced

by the Construction Module is refined by the user through interactive support by DODDLE-OWL. In order to refine the initial ontology, we manage concept drift and evaluate set of concept pairs. Since the initial concept hierarchy is constructed from a general ontology, we need to adjust the initial concept hierarchy to the specific domain considering an issue called Concept Drift. It means that the position of particular concepts changes depending on the domain. For concept drift management, DODDLE-OWL applies two strategies: Matched Result Analysis and Trimmed Result Analysis. These strategies are described in our former study [5]. At the construction phase of concept specification template from set of concept pairs generated by the Construction Module, DODDLE-OWL needs a criterion to evaluate significant concept pairs. In [5], two statistics based methods are investigated: the value of context similarity by WordSpace method [7] and the value of confidence by the association rule learner [8]. Those methods and values based on co-occurrency of concepts work well in terms of wide use (do not depend on some particular domains), its cost of preparation. The ontology constructed by DODDLE-OWL can be exported with the representation of OWL. Finally, MR^3 (Meta-Model Management based on RDF(S) [9] Revision Reflection) [10] is connected with DODDLE-OWL and works with an RDF(S) graphical editor.

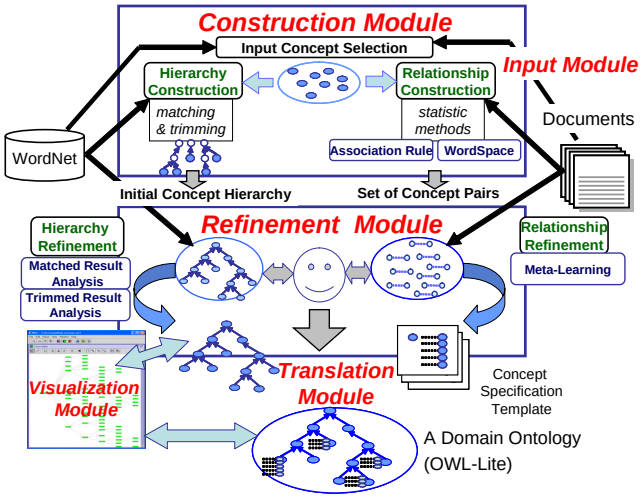


Fig. 1. DODDLE-OWL overview

3 Implementation

In this section, we describe the system architecture from the aspect of system implementation. DODDLE-OWL is realized in conjunction with MR^3 [10].

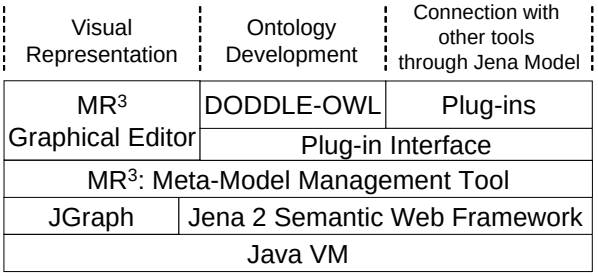


Fig. 2. DODDLE-OWL architecture

MR^3 is an RDF(S) graphical editor with meta-model management facility such as consistency checking of classes and a model in which these classes are used as the type of instances. Figure 2 shows the relationship between DODDLE-OWL and MR^3 in terms of system implementation. Both MR^3 and DODDLE-OWL are implemented in Java language. MR^3 is implemented using JGraph [11] for RDF(S) graph visualization, and Jena 2 Semantic Web Framework [12] for enabling the use of Semantic Web standards such as RDF, RDFS, N-triple and OWL. By using these libraries, MR^3 is implemented as an environment for graphical representation of the Semantic Web contents. Additionally, MR^3 also has plug-in facility to extend its functionality.

Figure 3 shows a typical usage of DODDLE-OWL. DODDLE-OWL's user interface consists of Input Module, Construction & Refinement Modules for Hierarchy, Construction & Refinement Modules for Relationships, Visualization Module MR^3 , and Translation Module into OWL-Lite. First, the user opens a document ((1) in Figure 3). Then, in the Input Module ((2) in Figure 3), the user can see noun terms that come up sorted frequently in the document. The user selects some terms as the input from the noun terms. As input of DODDLE-OWL, the user associates those terms with concepts by referring "the WordNet concepts" in (2) of Figure 3. For example, the user decide which "concept" (i.e. synset in WordNet) is suitable for the term "party". By referring to the synset and term's definition, the user selects an appropriate concept for the word "party". After mapping terms and their synsets, an initial concept hierarchy is produced. Also set of concept pairs are extracted by statistic methods such as WordSpace method and the association rule learner by default parameters. (3) of Figure 3 shows the Construction & Refinement Modules for Hierarchy. This module indicates some groups of concepts in the taxonomy so that the user can decide which part should be refined. (4) of Figure 3 shows the display of concept drift management in the Visualization Module MR^3 . (5) of Figure 3 shows the Construction & Refinement Modules for Relationships. This module is used for setting parameters used in the WordSpace method and the association rule learner to apply to documents in order to generate significantly related concept pairs. In WordSpace method, there are parameters such as the gram number (de-

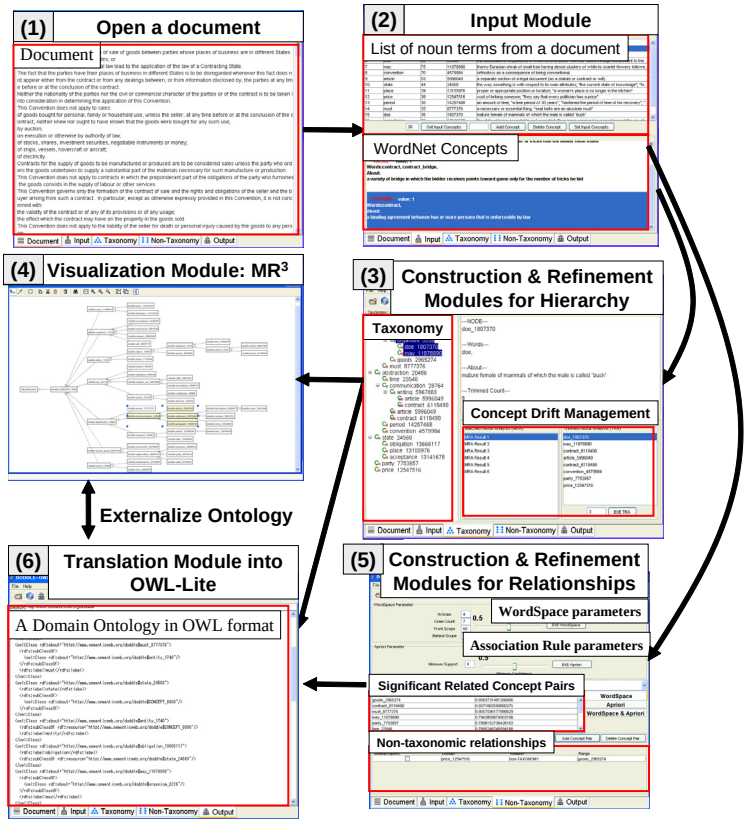


Fig. 3. A typical usage of DODDLE-OWL

fault gram number is four), minimum N-gram count (to extract high-frequency grams only), front scope and behind scope in the text. In the association rule learner, minimum confidence and minimum support are set by the user. As a result, the user got a domain ontology as (6) in Figure 3.

4 Case Studies

In order to evaluate how DODDLE-OWL is doing in a practical field, case studies have been done in particular field of business. The particular field of business is called "XML Common Business Library"(xCBL) [13].

4.1 A Case Study in the Business Field

Input terms in the Case Study with xCBL Table 1 shows input terms in this case study. They are 57 business terms extracted by a user from xCBL Document Reference. The user is not a expert but has business knowledge.

Table 1. Significant 57 Concepts in xCBL

acceptance	agreement	auction	availability	business
buyer	change	contract	customer	data
date	delivery	document	exchange rate	financial institution
foreign exchange	goods	information	invoice	item
line item	location	marketplace	message	money
order	organization	partner	party	payee
payer	payment	period of time	price	process
product	purchase	purcahse agreement	purchase order	quantity
quotation	quote	receipt	rejection	request
resource	response	schedule	seller	service
shipper	status	supplier	system	third party
transaction	user			

Table 2. The Change of the Number of Concepts under Taxonomic Relationship Acquisision

Model	Input Terms	Initial Model	Triimed Model	Concept Hierarchy
# Concept	57	152	83	82

Taxonomic Relationship Acquisition Table 2 shows the number of concepts in each model under taxonomic relationship acquisition and table 3 shows the evaluation of two strategies by the user. The recall per subtree is more than 0.5 and is good. The precision and the recall per path are less than 0.3 and are not so good, but about 80 % portion of taxonomic relationships were constructed with Hierarchy Construction Module and Hierarchy Refinement Module support.

Non-Taxonomic Relationship Learning

1. Construction of WordSpace

High-frequency 4-grams (sets of four words that co-occur term-by-term) were extracted from xCBL Document Description (about 2,500 words) standard form conversion removed duplication, and 1240 kinds of 4-grams were obtained. In order to keep density of a collocation matrix high, the extraction frequency of 4-grams must be adjusted according to the scale of text corpus. In order to construct a context vector, a sum of 4-gram vectors around appearance place circumference of each of 57 concepts was calculated. In order to construct a context scope from some 4-grams, it consists of putting together 10 4-grams before the 4-gram and 10 4-grams after the 4-grams independently of length of a sentence. For each of 57 concepts, the sum of context vectors in all the appearance places of the concept in xCBL was calculated, and the vector representations of the concepts were obtained. The set of these vectors is used as WordSpace to extract concept pairs with context similarity. Having calculated the similarity from the inner product for concept pairs which is all the combination of 57 concepts, 40 concept pairs were extracted.

Table 3. Precision and Recall in the Case Study with xCBL

	Precision	Recall per Path	Recall per Subtree
Matched Result	0.2 (5/25)	0.29 (5/17)	0.71 (5/7)
Trimmed Result	0.22 (2/9)	0.13 (2/15)	0.5 (2/4)

Table 4. Evaluation by the User with xCBL definition

	WordSPace (WS)	Association Rules (AR)	The Join of WS and AR
# Extracted concept pairs	40	39	66
# Accepted concept pairs	30	20	39
# Rejected concept pairs	10	19	27
Precision	0.75 (30/40)	0.51 (20/30)	0.59 (39/66)

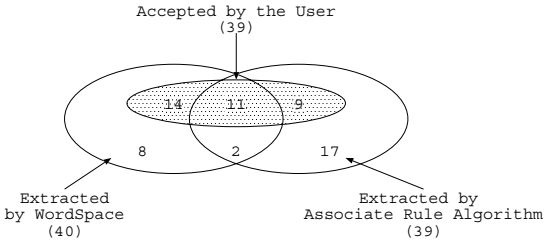


Fig. 4. Two Difference Sets of Concept Pairs from WS and AR and Concept Sets have Relationships

2. Relationship Construction Module
DODDLE-OWL extracted 39 pairs of terms from text corpus using the above-mentioned association rule algorithm. There are 13 pairs out of them in a set of similar concept pairs extracted using WordSpace. Then, DODDLE-OWL constructed concept specification templates from two sets of concept pairs extracted by WordSpace and Associated Rule algorithm.
3. Evaluation of Results of Relationship Refinement Module
The user evaluated the following two sets of concept pairs: one is extracted by WS (WordSpace) and the other is extracted by AR (Association Rule algorithm). Figure 4 shows two different sets of concept pairs from WS and AR. It also shows portion of extracted concept pairs that were accepted by the user. Table 4 shows the details of evaluation by the user, computing precision only. Since the user didn't define concept definition in advance, we can not compute recall. Looking at the field of precision in Table 4, the precision from WS is higher than others. Most of concept pairs which have relationships were extracted by WS. The percentage is about 77% (30/39). But there are some concept pairs which were not extracted by WS. Therefore taking the join of WS and AR is the best method to support a user to construct non-taxonomic relationships.

4.2 Results and Evaluation of Case Studies

In regards to support in constructing taxonomic relationships, the precision and recall are less than 0.3 in the case study. Generally, 70 % or more support comes from Hierarchy Construction Module and Hierarchy Refinement Module. About more than half portion of the final domain ontology results in the information extracted from WordNet. Since the two strategies just imply the part where concept drift may come up, the part generated by them has low component rates and about 30 % hit rates. So one out of three indications based on the two strategies work well in order to manage concept drift. The two strategies use matched and trimmed results, therefore based on structural information of an MRD only, the hit rates are not so bad. In order to manage concept drift

smartly, we may need to use more semantic information that is not easy to come up in advance in the strategies, and we also may need to use domain specific text corpus and other information resource to improve supporting a user in constructing taxonomic relationships.

In regards to construction of non-taxonomic relationships, the precision in the case study with xCBL is good. Generating non-taxonomic relationships of concepts is harder than modifying and deleting them. Therefore, DODDLE-OWL supports the user in constructing non-taxonomic relationships.

After analyzing results of case studies, we have the following problems.

1. Determination of a Threshold: Threshold of the context similarity changes in effective value with each domain. It is hard to set up the most effective value in advance.

2. Specification of a Concept Relation: Concept specification templates have only concept pairs based on the context similarity, it requires still high cost to specify relationships between them. It is needed to support specification of concept relationships on this system in the future work.

3. Ambiguity of Multiple Terminology: For example, the term "transmission" is used in two meanings, "transmission (of goods)" and "transmission (of communication)", in a document, but DODDLE-OWL considers these terms as the same and creates WordSpace as it is. Therefore constructed vector expression may not be exact. In order to extract more useful concept pairs, semantic specialization of a multisense word is necessary, and it should be considered that the 4-grams with same appearance and different meaning are different 4-grams.

5 Related Work

Navigli et,al. proposed OntoLearn [14], that supports domain ontology construction by using existing ontologies and natural language processing techniques. In their approach, existing concepts from WordNet are enriched and pruned to fit the domain concepts by using NLP (Natural Language Processing) techniques. They argue that the automatically constructed ontologies are practically usable in the case study of a terminology translation application. However, they did not show any evaluations of the generated ontologies themselves that might be done by domain experts. Although a lot of useful information is in the machine readable dictionaries and documents in the application domain, some essential concepts and knowledge are still in the minds of domain experts. We did not generate the ontologies themselves automatically, but suggests relevant alternatives to the human experts interactively while the experts' construction of domain ontologies. In another case study [15], we had an experience that even if the concepts are in the MRD (Machine Readable Dictionary), they are not sufficient to use. In the case study, some parts of hierarchical relations are counterchanged between the generic ontology (WordNet) and the domain ontology, which are called "Concept Drift". In that case, presenting automatically generated ontology that contains concept drifts may cause confusion of domain experts. We argue that the initiative should be kept not on the machine, but on the hand

of the domain experts at the domain ontology construction phase. This is the difference between our approach and Navigli's. Our human-centered approach enabled us to cooperate with human experts tightly.

From the technological viewpoint, there are two different related research areas. In the research using verb-oriented method, the relation of a verb and nouns modified with it is described, and the concept definition is constructed from this information (e.g. [16]). In [17], taxonomic relationships and Subcategorization Frame of verbs (SF) are extracted from technical texts using a machine learning method. The nouns in two or more kinds of different SF with the same frame-name and slot-name are gathered as one concept, base class. And ontology with only taxonomic relationships is built by carrying out clustering of the base class further. Moreover, in parallel, Restriction of Selection (RS) which is slot-value in SF is also replaced with the concept with which it is satisfied instantiated SF. However, proper evaluation is not yet done. Since SF represents the syntactic relationships between verb and noun, the step for the conversion to non-taxonomic relationships is necessary.

On the other hand, in ontology learning using data-mining method, discovering non-taxonomic relationships using an association rule algorithm is proposed by [18]. They extract concept pairs based on the modification information between terms selected with parsing, and made the concept pairs a transaction.

By using heuristics with shallow text processing, the generation of a transaction more reflects the syntax of texts. Moreover, RLA, which is their original learning accuracy of non-taxonomic relationships using the existing taxonomic relations, is proposed. The concept pair extraction method in our paper does not need parsing, and it can also run off context similarity between the terms appeared apart each other in texts or not mediated by the same verb.

6 Conclusion

In this paper, we presented a support environment for ontology construction named DODDLE-OWL, which aims at a total support environment for user-centered on-the-fly ontology construction. Its main principle is that high-level support for users through interaction. First, the user selects some terms as the input in the Input Module. Then, the Construction Module generates the basis of ontology in the forms of an initial concept hierarchy and set of concept pairs, by referring to WordNet as an MRD and a document. The Refinement Module provides management facilities for concept drift in the taxonomy and identifying significant set of concept pairs in extracted related concept pairs. According to case studies, it is important to select combinations of algorithms to get better candidate of set of concept pairs. Finally, the Translation Module produces an OWL-Lite file, which is able to put on public as a Semantic Web ontology.

We think that meta-learning scheme can be applied to the Refinement Module of DODDLE-OWL. CAMLET [19], a constructive meta-learning scheme has been proposed that can reconstruct learning algorithms from method level. This

approach will help to determine which learning algorithm to use on extracting set of concept pairs on each domain.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
2. Heijst, G.V.: The Role of Ontologies in Knowledge Engineering. Dr.thesis, University of Amsterdam (1995)
3. Ding, Y., Foo, S.: Ontology Research and Development, Part 1 – a Review of Onlotogy. Journal of Information Science (2002) pp.123–136
4. Michael K. Smith, C.W., McGuinness, D.L.: OWL Web Ontology Language Guide (2004) http://www.w3.org/TR/owl-guide/.
5. Sugiura, N., et al.: A Domain Ontology Engineering Tool with General Ontologies and Text Corpus. Proceedings of the 2nd Workshop on Evaluation of Ontology based Tools (2003) pp.71–82
6. G.A.Miller: WordNet: A Lexical Database for English. ACM (1995) pp.39–41
7. Marti A. Hearst, H.S.: Customizing a Lexicon to Better Suit a Computational Task. Corpus Processing for Lexical Acquisition (1996) pp.77–96
8. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. Proceedings of VLDB Conference (1994) pp.487–499
9. Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: Rdf Schema. W3C Proposed Recommendation (2003) http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.
10. Takeshi Morita, Noriaki Izumi, Naoki Fukuta and Takahira Yamaguchi: MR^3: Meta-Model Management based on RDFs Revision Reflection. Proceedings of the 6th Joint Conference on Knowledge-Based Software Engineering (JCKBSE) (2004) pp.228–236
11. Alder, G.: Jgraph. (2003) http://www.jgraph.com.
12. HP Labs: Jena Semantic Web Framework. (2003) http://jena.sourceforge.net/downloads.html.
13. One, C.: (xcbl:xml common business library) http://www.xcbl.org/.
14. Navigli, R., Paola Velardi: Automatic Adaptation of WordNet to Domains. Proceedings of International Workshop on Ontologies and Lexical Knowledge Bases (2002)
15. Yamaguchi, T.: Constructing domain ontologies based on concept drift analysis. Proceedings of the IJCAI99 Workshop on Ontologies and Problem Solving methods(KRR5) (1999)
16. Hahn, U., Schnattingerg, K.: Toward text knowledge engineering. AAAI-98 proceedings (1998) pp.524–531
17. Faure, D., Nédellec, C.: Knowledge Acquisition of Predicate Argument Structures from Technical Texts. Proceedings of International Conference on Knowledge Engineering and Knowledge Management (1999)
18. Maedche, A., Staab, S.: Discovering Conceptual Relations from Text. Proceedings of 14th European Conference on Artificial Intelligence (2000) pp.321–325
19. Hidenao Abe and Takahira Yamaguchi: Constructive meta-learning with machine learning method repositories. in Proc. of the 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expoert Systems (IEA/AIE) (2004) pp.502–511

Collaborative Ontology Building with Wiki@nt
- A Multi-agent Based Ontology Building Environment

Jie Bao and Vasant G Honavar

Artificial Intelligence Research Laboratory
Department of Computer Science
Iowa State University, Ames, IA 50011-1040, USA
{baojie,honavar}@cs.iastate.edu

Abstract. Collaborative ontology building requires both knowledge integration and knowledge reconciliation. Wiki@nt is an ontology building environment that supports collaborative ontology development. Wiki@nt is based on a language extension to $\mathcal{SHOQ}(\mathbf{D})$ with $\mathcal{O}$ (partial order on axioms) and $\mathcal{P}$ (localized axioms in package) constructors. Wiki@nt supports integration and reconciliation of multiple independently developed, semantically heterogeneous, and very likely inconsistent ontology modules. A web browser based editor interface is provided, with features to support team work, version control, page locking, and navigation.

1 Introduction

1.1 Ontology Editing is a Knowledge Integration Process

Semantic Web aims to support seamless and flexible access, use of semantically heterogeneous, networked data, knowledge, and services. The success of the semantic web enterprise relies on the availability of a large collection of domain specific ontologies and mappings between ontologies to allow integration of data [12]. However, by its very nature, ontology construction is a collaborative process which involves direct cooperation among domain experts, knowledge engineers or/and software agents, or indirect cooperation through reuse or adaptation of previously published, autonomously developed ontologies.

In such settings, typically, different participants have only partial knowledge of the domain, and hence can contribute only partial ontologies of the domain.Common tasks involve refinement of a predefined ontology, and integration of several such partial ontologies to obtain a coherent ontology. Semantic mismatches and logical inconsistencies between independently developed ontologies are unavoidable. Thus, there is an urgent need for principled approaches and flexible tools for allowing individuals to collaboratively build, refine, and integrate existing ontologies as needed in specific contexts or for specific applications.

1.2 Proposed Approach

While there has been a great deal of work on ontology languages, inference mechanisms, as well as ontology editing environments, relatively little attention has been paid to the development of principled approaches and tools for collaborative ontology building. Existing ontology editing and discovery tools are mostly focused on stand-alone ontology development rather than collaborative construction of ontologies. In this paper, we propose Wiki@nt, a general architecture of an ontology editing, ontology refinement, and ontology integration environment. Wiki@nt exploits $\mathcal{OSHOQP}(\mathbf{D})$, a modular ontology representation language

with preference partial order on axioms; a light-weight, browser-based ontology editor which requires minimal user effort and allows concurrent editing and integration of ontologies, is presented.

2 Collaborative Ontology Building as Knowledge Integration and Reconciliation

We start with a brief discussion of the theoretical basis of Wiki@nt including logical foundations of ontology languages. We then introduce a modular representation of ontologies and discuss some problems in inconsistency reconciliation with modular ontologies.

2.1 Description Logic as a Knowledge Representation Language

Ontologies are typically described using ontology languages, such as DAML+OIL or OWL. Description logic(DL) can be used to express the formal semantics of an ontology written in those languages. A description logic consists of a Tbox and an Abox, where the Tbox is a finite set of terminological axioms such as $C \sqsubseteq D$, and the Abox is a finite set of assertional statements such as $C(a)$ or $R(a, b)$. In particular, $\mathcal{SHIOQ}(\mathbf{D})$ is the formal background DL for OWL. A complete list of $\mathcal{SHIOQ}(\mathbf{D})$ and OWL/DL constructors can be found in [10].

However, ontology languages with $\mathcal{I}$ (i.e. inverse roles) constructor suffers from complexity and/or intractability problems when combined with $\mathcal{O}$ or $(\mathbf{D})$. Hence, we use a subset of $\mathcal{SHIOQ}(\mathbf{D})$, $\mathcal{SHOQ}(\mathbf{D})$, as the basis for a collaborative ontology development environment.

We assume that we have an abstract domain $\triangle^{\mathcal{I}}$, and a set of data types D and associate with each $d \in D$, a set $d^D \subseteq \triangle_D$ where $\triangle_D$ is the domain of all types. An example Animal Ontology is given here:

```
SubClassOf( Dog , Carnivore )
SubClassOf( Dog , Pet )
SubClassOf( Carnivore, Animal)
  restriction(eats allValueFrom(Animal))
ObjectProperty( eats) domain( Animal )
individual ( billy type(Dog))
```

2.2 Package-Extended Ontology

Collaborative ontology building demands modularized ontology representation by its very nature. Current ontology languages like OWL, while offer some degree of modularization using XML namespaces, fail to fully support modularized semantics. In our previous work [3], we have argued for package based ontology language extensions to overcome these limitations. In the resulting ontology language $\mathcal{SHOQP}(\mathbf{D})$, a **package** is an ontology module with clearly defined access interface. Mapping between packages is performed by **views** which define a set of queries on the referred packages. Semantics are localized by hiding semantic details of a package with appropriately defined **interfaces** (special views). Packages provide an attractive way to compromise between knowledge sharing and knowledge hiding in collaborative design and use of ontologies.

Table 1 gives the syntax and semantics of $\mathcal{P}$ constructors. Let P be the set of all packages. We define $\triangle_P$ as the domain of P. We assume that the domain of

interpretation of all packages $\triangle_P$ is disjoint from the concrete datatype domain $\triangle_D$, the abstract concept domain $\triangle^{\mathcal{I}}$, the abstract role domain $\triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$ and concrete role domain $\triangle^{\mathcal{I}} \times \triangle_D$. The resulting *package-extended* description logic language is called $\mathcal{SHOQP}(\mathbf{D})$ where $\mathcal{P}$ stands for "package-extended".

Table 1. Syntax and semantics of $\mathcal{P}$ Constructors

Constructor	Syntax	Semantics
Package	p	$p^P \in \triangle_P$
View	v	$v^{\mathcal{I}} \in \triangle_P$
Global Pkg	p_0	$p_0 \in \triangle_P$
InPackage	R_P	$R_P^{\mathcal{I}} \subseteq \triangle_T^{\mathcal{I}} \times \triangle_P$
HomePackage	$HP(t)$	$HP(t)^{\mathcal{I}} = \{p \mid (t^{\mathcal{I}}, p) \in R_P^{\mathcal{I}}\}$
NestedIn	$\in_N$	$\in_N^{\mathcal{I}} \in \triangle_P \times \triangle_P$, $\in_N^{\mathcal{I}} = (\in_N^{\mathcal{I}})^+$
SLM	SLM(t,p)	$p \in \triangle_P$ can access $t \in \triangle_T^{\mathcal{I}}$iff SLM(t,p)=true
	public(t,p)	$\forall p$, public$(t,p) = true$
	private(t,p)	$\forall p$, private$(t,p) := (p = HP(t))$
	protected(t,p)	$\forall p$, protected$(t,p) := (p = HP(t)$ *or* $p \in_N HP(t))$
Import	$im(P_1, P_2)$	P_2 is imported into P_1

2.3 Ontology Reconciliation

As noted earlier, semantic mismatches and possible logical inconsistencies between independently developed ontology modules make the combining of such modules a challenging task. Specifically, in the case of two ontology modules α, β, it is possible that although $\alpha \vDash t$, the module resulting from combining α and β may not entail t i.e., $\{\alpha, \beta\} \nvDash t$ That is, any system for collaborative ontology building has to provide mechanisms for handling *nonmonotonicity*.

An example (adapted from [9]) illustrates this problem. A dog is carnivore; however, a sick dog sometimes eats grass. Formally, we add new axioms to the Animal Ontology:

```
DisjointClasses(Plant,Animal)
SubClassOf( SickDog, Dog)
   restriction(eats someValueFrom(Plant))
```

The resulting knowledge base will be inconsistent because a sick dog (which is a dog) now can eat grass (which contradicts the assertion that dogs are carnivores). Several techniques have been developed to reconcile inconsistent ontology system, such as default logic [2] and defeasible logic [13] [9]. Here we extend our $\mathcal{SHOQP}(\mathbf{D})$ with the $\mathcal{OSHOQ}(\mathbf{D})$ [13]. An axiom is said to be *defeasible* if some other axiom could *defeat* (or override) it. The resulting ontology language is called $\mathcal{OSHOQP}(\mathbf{D})$ where $\mathcal{O}$ denotes a strict partial order on the axioms.

Definition 1 *A $\mathcal{OSHOQP}(\boldsymbol{D})$-knowledge base is a tuple $\langle \mathcal{T}, < \rangle$, where $\mathcal{T}$ is a $\mathcal{SHOQP}(\boldsymbol{D})$-knowledge base and $<$ is a strict partial order between axioms of $\mathcal{T}$. For each pair $a_1 < a_2$, a_2 is said to be **defeasible** while a_1 is a (possible) **defeater** of a_2*

For example, if we revisit the Animal Ontology in $\mathcal{OSHOQP}(\mathbf{D})$, The terminology $\mathcal{T}$ could be rewritten as

$package(1)$	$package(2)$
(1a) $public(Dog, 1)$	(2a) $im(2, 1)$; `import package 1`
(1b) $1 : Dog \sqsubseteq 1 : Carnivore$	(2b) $public(Plant, 2)$
(1c) $1 : Dog \sqsubseteq 1 : Pet$	(2c) $2 : Plant \sqcap 1 : Animal \sqsubseteq \bot$
(1d) $public(Animal, 1)$	(2d) $2 : SickDog \sqsubseteq 1 : Dog$
(1e) $public(eats, 1)$	$\sqcap \exists 1 : eats.2 : Plant$
(1f) $1 : Carnivore \sqsubseteq 1 : Animal$	
(1g) $1 : Carnivore \sqsubseteq \forall 1 : eats.1 : Animal$	
(1h) $\{1 : billy\} \sqsubseteq 1 : Dog$	

A simple combination of packages 1 and 2 is inconsistent on (1g) and (2d). However, with a partial order $<$, this logical inconsistency can be eliminated. One such possible partial order is (2d)$<$(1g) (read as axiom (2d) is *stronger than* axiom (1g)). In this case, a specific axiom (2d) *defeats* the general rule (1g). When there is a logical conflict between a pair of axioms, the weaker of the two is discarded. More details of $\mathcal{OSHOQP}(\mathbf{D})$ and its reasoning problems could be found in the unabridged version of this paper [4]

3 Architecture

$\mathcal{OSHOQP}(\mathbf{D})$ gives us an expressive language to build ontology from autonomous, distributed, and possibly inconsistent ontology modules. Wiki@nt is the implementation of an ontology editor based on $\mathcal{OSHOQP}(\mathbf{D})$ to support collaborative ontology building by a community of autonomous domain experts, organizations, or even software agents.

The name "Wiki@nt" suggests that it has a wiki-like editing environment. Wiki is originally a collaborative documentation writing/website building tool. Typical wiki system includes a script language (usually a simplified subset of HTML tags), a set of wiki pages written in the script language and shown in translated HTML pages, a RCS version control system to record modification of contents, an user profile and concurrent conflict management system to enable multiple user editing the same contents, a content navigation system such as showing link-in and link-out pages, and a simple-to-use, browser-based editing environment to generate or modify content on the fly.

We find that those features are quite desirable in a collaborative ontology editor. While most widely-used ontology editors, such as Protege and OilEd, work very well for developing a single ontology module, they do not lend themselves to collaborative ontology building. This is due to the lack of a built-in formalism to support modular ontology representation, and the lack of support for communication and cooperation among multiple individuals in editing a shared ontology consisting of multiple, independently developed modules. To overcome those deficiencies, we propose using wiki to edit $\mathcal{OSHOQP}(\mathbf{D})$ ontology. An ontology module is composed of one or more wiki pages; multiple users can edit the same content, with version control and transaction management; ontology are loaded into or uploaded from a set of wiki pages and managed by an ontology repository. Figure 1 shows the architecture of Wiki@nt.

3.1 Wiki Engine

A wiki engine should do the tasks of

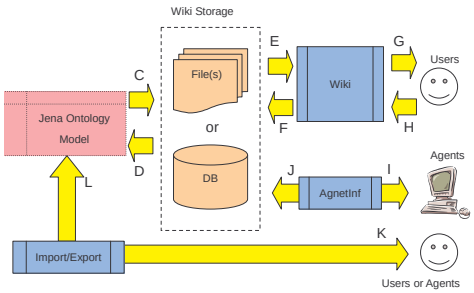


Fig. 1. The Architecture of Wiki@nt

- Provides a web interface for ontology editor and browser.
- Translates the ontology markup script to HTML for the web browser.
- Manages the storage of wiki pages, in plain file or database.
- Provides version control. When a modification for an axiom is submitted, the previous version is stored and could be restored when the committed version is found incorrect or impropriate.
- Provides transaction management.
- Generates reference report for wiki pages. Terms be used in an axiom group, and other groups that referring this group, are listed for browsing purpose.
- Generates a RSS feed for ontology repository updates.

The wiki engine we utilized is based on the JSPWiki(http://www.jspwiki.org) and implemented in Java and JSP.

3.2 Agent Management

Each participant in Wiki@nt is considered as an agent. Agent is assigned with different privileges, such as ontology administrator and package manager. Agent could join the editing of any existing ontology module or create new module.

Although our current design of Wiki@nt does not include concrete design of software agents, we do reserve an RPC interface that enables agents to communicate with Wiki@nt. The reason is while fully automatic ontology construction or mapping are still impossible, software agents can assist humans in several aspects of collaborative ontology development e.g., finding useful concepts and relations among concepts from original data sources. Small pieces of ontologies, such as consistent concept (term) in data or concurrence of two concepts, can be generated by software agents. The results may be subjected to review of domain experts, or even other software agents. Thus, in principle, it is possible for software agents to participate in collaborative ontology building using Wiki@nt.

3.3 Ontology Markup Script

We defined a set of markup script tags to correspond to the syntax of the ontologies. When a wiki page is under editing, its wiki markup script is loaded and translated to user friendly text, such as HTML web page. The syntax is a extension to OWL to support package and partial order on axioms. Wiki markup script is a human readable syntax equivalent to the N-Triple syntax. N-Triple syntax is an alternative to the RDF/XML syntax and each line in N-Triple serialization is a triple statement with subject, predicate and object.

For example, axiom SubClassOf(Dog,Carnivore) in the Animal Ontology could be represented by N-Triple syntax as: <http://mydomain.org/animal#Dog> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://mydomain.org/animal#Carnivore>, or in short form <animal:Dog> <rdfs:subClassOf> <animal:Carnivore>. It's wiki script is [animal:Dog] [rdfs:subClassOf][animal:Carnivore]

Each axiom is assigned a URI (uniform resource identifier) as label. Thus, for example, http://mydomain.org/animal/package1#Dog represents $Dog \sqsubseteq Carnivore$ in package(1), Animal Ontology.

User can create a new page or modify the source script of an existing page. The editing action is assisted by several wizards (such as class creating wizard) and a browser (eg. Show subclass and superclass of the class in question).

3.4 Memory Management

While most of the popular ontology editors have in-memory model for edited ontology, Wiki@nt doesn't maintain in-memory model for each resident ontology for several reasons: An in-memory model limits the scalability of the system with respect to both the axioms number in one ontology and the number of ontologies in the Wiki@nt ontology repository; In-memory model implicitly assumes the existence of a global ontology during the ontology development process and requires monotonic behavior of the ontology - neither of these assumptions is desirable in a collaborative ontology building scenario. In short, creating a centralized ontology model in memory defeats the very purpose of having a modular ontology

Note that that even when the size of the ontology in question is huge, usually only a small fraction of its axioms are involved during an editing action. Hence, we store the ontology as a set of separate, possibly distributed blocks in Wiki@nt. Each block is serialized to external storage when it's not being actively edited, and being loaded into the memory only if it's edited or referred. A (local or remote) partial ontology model will be dynamically loaded into local memory in a reasoning process only if it is needed. The partial model could be a package, a small part of a package, or even an axiom. This is inspired by widely used techniques of database memory management where partial content of the database is dynamically loaded and unloaded to allow manipulation of of a much larger volume of data than can fit in limited memory.

Another technique to reduce memory burden on Wiki@nt server is to build the dynamically loaded model on client side instead of server side. This is reasonable since the dynamically loaded model is just a local-interested part of whole ontology. Technically, this is done by offering a Java Applet interface to read and update Wiki@nt pages.

3.5 Modularization of Ontology

Ontology stored in Wiki@nt is managed on package level and block level. A package is a logic module for an ontology, usually from a single participant. An ontology could be composed by several packages, and one package could be reused by multiple ontologies.

A block is a set of related axioms inside a package, and will be physically mapped to a wiki page. A package will include one or more such blocks(pages).

Although different decompositions of an ontology package are logically equivalent, the size of each ontology block will affect the convenience and efficiency of ontology editing and reasoning. It should not be too big (i.e. the whole package), or too small (e.g., a single triple). In Wiki@nt, we choose axiom groups as ontology blocks. Each axiom group contains triples with same subject. For example, the axiom groups in Animal Ontology package(1) will be *Dog*, *Carnivore*, *eats*, and *billy*. Restrictions and anonymous classes, are assigned to the terms from where they are referred. Each axiom group is translated to wiki markup script and stored as a wiki page. An ontology could be stored distributedly in multiple pages, physically in file or database, and could be dynamically, partially loaded when necessary.

3.6 Ontology exporting/importing

When an ontology is needed e.g., for reasoning, we export wiki pages as a single ontology file or read an ontology file into Wiki Ontology Repository. The relevant portion of an ontology is extracted or assembled from the wiki pages. We use the Jena toolkit to create the in-memory model and as parser/writer for ontology files.

Each loaded ontology is assigned a unique name, eg. http://mydomain.org/animal/, and it's member packages, eg. http://mydomain.org/animal/package1, are registered to that ontology. It's also possible that packages from different ontologies could be reassembled into a new ontology, thus provide a flexible way for ontology reuse and integration.

3.7 Reconcile the Inconsistency

Inconsistency among modules should be resolved when integrate those modules. In Wiki@nt, we assume each package should be consistent. A partial order can be specified on package level, eg. [Package1] [wiki:stronger][Package], which means all axioms in Package1 are stronger than Package2; it can also be specified on axiom level, like [Package1:1g] [wiki:stronger][Package2:2d].

The specification of the partial order $<$ among modules and axioms may be based on principles of:

- reliability of the source of module/axiom
- the social order of the author of module/axiom
- A more recent module/axiom may be preferred over an earlier one;
- exceptions are stronger than the general rules.

Wiki@nt defined two default defeating rules if user not specifies otherwise. First, ut assigns higher priority to local package axioms relative to axioms from imported packages in cases where a local package can be seen as an extension or an exception to a general ontology. Other partial order assignment policies is based on the social order of the agents in the Wiki@nt community, such as ontology administrator, package manager and common user.

3.8 Transaction Management

Transaction management is to ensure consistency of module and protect critical resource from multiple access. It is widely used in DBMS and certain ontology

editor, such as OntoEdit [14]. However,OntoEdit only allow locking of concept hierarchy.

Wiki@nt denies the write-access of agents to a page and related pages if it is locked by some other agents. Following strategies are used for determining what pages should be locked:

- If a concept is under editing/locking, its superclasses in the class hierarchy will be locked.
- If a property is under editing/locking, its superproperties in the property hierarchy will be locked.
- If a instance is under editing/locking, its belonging class will be locked.
- If a concept, property or instance is under editing/locking, all other concepts, properties or instances in the same page(axiom group) will be locked.
- If a package is state as being locking, all importing package will be locked.

Locking could be propagated by recursively apply above-mentioned strategies.

4 Summary and Discussion

4.1 Related Work

Collaborative Ontology Editor A number of ontology editors have been reported in the literature [6,11]. However, most existing ontology editors including the most widely used ontology editors Protege and OilEd provide little support for collaborative ontology development. Representative editors that support collaborative ontology editing include CODE [8], KAON [5], OntoEdit [14], Ontolingua [7], and WebODE [1] . Most of them provide concurrent access control with transaction oriented locking, and in some cases, even rollback. However, none of the existing ontology editors, to the best of our knowledge, provides principled approaches for manipulating independently developed, semantically heterogeneous ontology modules or for reconciling logical inconsistencies between such modules.

One advantage of Wiki@nt over reported editors is its truly distributed nature for storage of ontology modules. While most editors requires in-memory model for current-editing ontology, Wiki@nt dynamically load partial model only when it is necessary, and/or share the memory burden between the server and clients.

Another distinct feature of Wiki@nt is its wiki-based design thus enable it borrow many well-stablished feature of wiki system, each as user-friendly and easy-to-use interface, version control, user management, page locking, translating from script to HTML, and persistent storage in relational database.

Collaborative Knowledge Base Construction Some collaborative knowledge base construction projects, although not focused on ontology building, address similar problems. **Nooron**(http://www.nooron.org) is a knowledge publishing system and has a wiki for ontology browsing. **MnM** [15] is an annotation tool which provides both automated and semi-automated support for annotating web pages with semantic contents. MnM integrates a web browser with an ontology editor and provides open APIs to link to ontology servers and for integrating

information extraction tools. However, it doesn't have concurrent access control. **FoaF**(http://www.foaf-project.org/) is an acronym for "Friend of a Friend" , an experimental project and vocabulary for the Semantic Web. The project is open and allows participants to add their own information. The result is an RDF based knowledge base containing contact and acquaintance information about the participants. **WikiPedia** (http://en.wikipedia.org/) is a wiki-based open-content encyclopedia that is editable by participants. Articles in WikiPedia are written in natural language, and the relation between items is not formal. Nevertheless, articles can be seen as concepts and links between them seen as properties among them, in a informal sense.**Open Directory Project** or called **DMOZ** (http://www.dmoz.org/) is an online, open, collaborative taxonomy building project for web catalog. Now it has a taxonomy tree of over 590,000 categories and over 4 million classified sites. The relations between DMOZ concepts is strict "subClassOf".

Although these projects lack formalized and full-fledged ontologies, they offer interesting demonstrations of the feasibility of collaborative ontology development. Wiki@nt proposed in this paper is inspired by the success of DMOZ and WikiPedia, and aims to provide support for such efforts using a formal ontology language to facilitate machine interpretable annotations of data.

4.2 Summary and Outlook

In this paper we have described

- The ontology representation formalism to support modularity and axiom order.
- A distributed ontology representation and storage methodology based on wiki.
- A Light-weight ontology editor to support collaborative ontology building.
- Important issues in wiki-based ontology editing, such as transaction management, memory management, agent management, modularization of ontology and inconsistency reconciliation.

Some interesting directions for future work include:

- Incorporation of more advanced transaction management and incorporation of safe mechanisms for handling simultaneous editing and modification of ontologies
- Investigation of useful policies for assigning partial order among axioms, including those that are base on machine learning or probabilistic approaches.
- Applications of collaborative ontology building environments for information integration from autonomous, distributed, semantically heterogeneous information sources.
- Detailed study of scalability of Wiki@nt, test it with big ontologies such as WordNet.

Acknowledgments

This research is supported in part by grants from the NSF (0219699) and the NIH(GM 066387) to Vasant Honavar

References

1. J. C. Arpirez, O. Corcho, M. Fernandez-Lopez, and A. Gomez-Perez. Webode: a scalable workbench for ontological engineering. In *Proceedings of the international conference on Knowledge capture*, pages 6–13. ACM Press, 2001.
2. F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. Technical Report RR-93-20, 1993.
3. J. Bao and V. Honavar. Ontology language extensions to support localized semantics, modular reasoning, and collaborative ontology design and ontology reuse. Technical report, TR00000341, Computer Sicence, Iowa State University.
4. J. Bao and V. Honavar. Collaborative ontology building with wiki@nt - a multi-agent based ontology building environment. Technical report, TR00000343, Computer Sicence, Iowa State University, 2004.
5. E. Bozsak, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic, N. Stojanovic, R. Studer, G. Stumme, Y. Sure, J. Tane, R. Volz, and V. Zacharias. Kaon - towards a large scale semantic web. In K. Bauknecht, A. M. Tjoa, and G. Quirchmayr, editors, *E-Commerce and Web Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings*, volume 2455 of *Lecture Notes in Computer Science*, pages 304–313. Springer, 2002.
6. M. Denny. Ontology building: A survey of editing tools. Technical report, O'Reilly XML.com, November 06, 2002.
7. A. Farquhar, R. Fikes, W. Pratt, and J. Rice. Collaborative ontology construction for information integration, 1995.
8. P. Hayes, R. Saavedra, and T. Reichherzer. A collaboration development environment for ontologies. In *Proceedings of the Semantic Integration Workshop, Sanibel Island, Florida,*, 2003.
9. S. Heymans and D. Vermeir. Using preference order in ontologies, 2002.
10. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1), 2003.
11. Ontoweb. Deliverable 1.3: A survey on ontology tools.
12. J. A. Reinoso-Castillo, A. Silvescu, D. Caragea, J. Pathak, and V. G. Honavar. Information extraction and integration from heterogeneous, distributed,autonomous information sources - a federated ontology-driven query-centric approach. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*, 2003.
13. D. V. S. Heymans. A defeasible ontology language. In e. a. E. R. Meersman, Z. Tari, editor, *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE : Confederated International Conferences CoopIS, DOA, and ODBASE 2002, Lecture Notes in Computer Science*, volume 2519, pages 1033–1046. Springer-Verlag Heidelberg, 2002.
14. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. OntoEdit: Collaborative ontology development for the semantic web. In *Proceedings of the first International Semantic Web Conference 2002 (ISWC 2002), June 9-12 2002, Sardinia, Italia.* Springer, LNCS 2342, 2002.
15. M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. Mnm: Ontology-driven tool for semantic markup. In S. Handschuh, N. Collier, R. Dieng, and S. Staab, editors, *Proceedings Workshop on Semantic Authoring, Annotation and Knowledge Markup (SAAKM 2002)*, pages 43–47, 2002.

Introduction to the EON Ontology alignment context

Jérôme Euzenat

INRIA Rhône-Alpes,
Jerome.Euzenat@inrialpes.fr

The EON so-called "Ontology alignment contest"[1] has been designed for providing some evaluation of ontology alignment algorithms. This is an introduction, rather than a synthesis, because we have not had enough time to draw real conclusions from the results provided by the participants. It will however provide the context for this evaluation.

1 Goals

The goal of the contest was firstly to illustrate how it is possible to evaluate ontology alignment tools.

The medium term goal is to set up a set of benchmark tests for assessing the strengths and weaknesses of the available tools and to compare them. These tests are focussing the characterisation of the behaviour of the tools rather than having them compete on real-life problems. It is expected that the set of tests could be a first version of a reference benchmark that tool developers can run in order to improve their tools and measure where they are.

Because of its emphasis on evaluating the performances of tools instead of the competition between them, the term contest was not the best one.

2 Method

The evaluation methodology consisted in publishing a set of ontologies to be compared with another ontology. The participants were asked to run one tool in one configuration on all the tests and to provide the results in a particular format. In this format[2], an alignment is a set of pairs of entities from the ontologies, a relation supposed to hold between these entities and a confidence measure in the aligned pair. The tools could use any kind of available resources, but human intervention. The participants were also asked to provide a paper, in a predefined format, describing their tools, their results and comments on the tests. These are the papers that are compiled here.

Along with the ontologies, a reference alignment was provided (in the same format). This alignment is the target alignment that the tools are expected to find. The reference alignment has all its confidence measures to the value 1 and most of the relations were equivalence (with very few subsumption relations). Because of the way the tests have been designed (see below), these alignments should not be contested. The participant

[1] http://co4.inrialpes.fr/align/Contest

[2] http://www.inrialpes.fr/exmo/software/ontoalign/

were allowed to compare their results to the output of their systems and the reference alignment and to chose the best tuning of their tools (overall).

The full test bench was proposed for examination to potential participants for 15 days prior to the final version. This allowed participants to provide some comments that could be corrected beforehand. Unfortunately, the real comments came later.

The results of the tests were expected to be given in terms of precision and recall of correspondences found in the produced alignment compared to the reference alignment. No performance time measures were required.

Tools were provided for manipulating the alignments and evaluate their precision, recall and other measures[2].

3 Test set

The set of tests consisted in one medium ontology (33 named classes, 39 object properties, 20 data properties, 56 named individuals and 20 anonymous individuals) to be compared to other ontologies. All ontologies were provided in OWL under its RDF/XML format.

This initial ontology was about a very narrow domain (bibliographical references). It was designed by hand from two previous efforts. This ontology took advantage of other resources whenever they were available. To that extent the reference ontology refers to the FOAF (Friend-of-a-friend) ontology and the iCalendar ontology.

There were three series of tests:

- simple tests such as comparing the reference ontology with itself, with another irrelevant ontology (the wine ontology used in the OWL primer) or the same ontology in its restriction to OWL-Lite;
- systematic tests that were obtained by discarding some features of the initial ontology leaving the remainder untouched. The considered features were (names, comments, hierarchy, instances, relations, restrictions, etc.). This approach aimed at recognising what tools really need. Our initial goal was to propose not just one feature discard but all the combinations of such. Unfortunately, we were unable to provide them before the launch of the contest.
- four real-life ontologies of bibliographic references that were found on the web and left untouched.

All the ontologies and reference alignments were produced by hand in a very short time. This caused a number of problems in the initial test base that were corrected later.

4 Results

As a first note, we expected five participants but finally only four entered. This is few, especially with regard to all the alignments algorithms out there. We hope that these four participants are the pionneer who will induce the others to put their work under comparison.

We would have liked to have a clear picture of the results before commenting. It happened that we were not able to get all the results in the appropriate format to compile

a table of the results based on similar ground for all participants. We expect to provide this on our web site and at the EON workshop (with certainly more participants).

The best way to learn about the results so far is to read what follows. The participants made their best to highlight why their tools were weak or strong and how to improve them.

5 Lesson learned

The first good thing that we learnt is that it is indeed possible to run such a test.

Another lesson that we have learnt is that OWL is not that homogeneous when tools have to manipulate it. Parsers and API for OWL (e.g., Jena and OWL-API) are not really aligned in their way to handle OWL ontologies. This can be related to very small matters which can indeed render difficult entering the challenge. It is our expectation that these products will improve in the coming year. For the moment we modified the files in order to avoid these problems.

People appreciated to be given tools to manipulate the required formats. It is clear that in order to attract participants, the test process should be easy.

We also realised that the production of an incomplete test bench (not proposing all combinations of discarded features) had an influence on the result. As a matter of fact, algorithms working on one feature only were advantaged because in most of the tests this feature was preserved.

Another lesson we learned is that asking for a detailed paper was a very good idea. We have been pleased of how much insight can be found in the comments of the competitors.

6 Future plans

We have shown that we can do some evaluation in which people can relatively easily jump in, even within a short span of time. The results given by the systems make sense and certainly made the tool designers think. So we think that such an evaluation is worthwhile and must be continued.

We plan to merge the two events which occurred this year:

- The Information Interpretation and Integration Conference (I3CON), held at the NIST Performance Metrics for Intelligent Systems (PerMIS) Workshop which focused on "real-life" test cases and compare algorithm global performance[3].
- This Ontology Alignment Contest at the 3rd Evaluation of Ontology-based Tools (EON) Workshop.

The combination of these events can feature a benchmark series like the one proposed at this workshop in order to calibrate the systems and some medium- to large-scale experiment, possibly made on purpose but supposed to reproduce real-life situation (with no reference alignment published).

[3] http://www.atl.external.lmco.com/projects/ontology/i3con.html

However, people coming from different views with different kind of tools do not naturally agree on what is a good test. In order to overcome this problem, the evaluation must be prepared by a committee, not from just one group.

Finally, in order to facilitate the participation to the contests, we must develop tools in which participants can plug and play their systems. In addition to the current evaluators and alignment loaders, we could provide some iterators on a set of tests for automating the process and we must automate more of the test generation process.

7 Acknowledgements

We warmly thanks each participant of this first contest. We know that they worked hard for having their results ready and they provided good papers presenting their experience. The evaluation improved a lot from their comments.

Heiner Stuckenschmidt and Liz Palmer proposed some variation of the tests (Natasha Noy proposed a couple of others in her paper).

We also thanks York Sure and Oscar Corcho, organisers of the EON workshop, for proposing to run this contest in the framework of EON.

Finally, Todd Hughes, who organized the I3Con competition, has always been supportive of this one and at clarifying their goals and relations.

Montbonnot, October 1st, 2004

Ontology Alignment - Karlsruhe

Marc Ehrig and York Sure

Institute AIFB, University of Karlsruhe

1 Introduction

This short paper accompanies the EON Ontology Alignment Contest at the International Semantic Web Conference 2004 in Hiroshima. First, it describes the system as used in Karlsruhe. The results of the actual experiments follow. Then we have a short discussion and interpretation of the results. In the end we provide a link to the actual results.

2 System

2.1 Alignment Process

We introduce the process of ontology alignment we use for the experiments here. The general process and an efficient algorithm for it have been presented in [ES04b,ES04a]. The presented approach is only capable of extracting semantical identity. Subsumption or complex alignments are currently not supported.

To find out whether two entities can be aligned we rely on rules which indicate support for an alignment. A rule consists of two main parts: an entity feature, and a comparison function. The result is a value between 0 and 1.

An entity is described through features which are represented in the ontology. A feature can be as simple as the label of entity. It can include intensional structural elements such as super- or sub-concepts for concepts, or domain and range for relations. Further we use extensional elements: a concept is described through its instances. Instance features can be instantiated attributes. According to our assumption we restrict our focus on these simple features, where simple means that only directly connected relations are respected, i.e. we do not consider the relations of a sub-concept of a super-concept of a concept. The following example will clarify the used features.

```
<owl:Class rdf:about=‘‘automobile’’/>
<owl:Class rdf:about=‘‘car’’/>
<rdf:Description rdf:about=‘‘porsche’’>
    <rdf:type rdf:resource=‘‘automobile’’>
    <rdf:type rdf:resource=‘‘car’’>
    <auto:speed rdf:resource=‘‘fast’’>
</rdf:Description>
<rdf:Description rdf:about=‘‘mercedes’’>
    <rdf:type rdf:resource=‘‘automobile’’>
    <auto:speed rdf:resource=‘‘fast’’>
</rdf:Description>
```

Example 1. Two entities with the similar super-concepts.

In Example 1 both `Porsche` and `Mercedes` have the feature `type`. Extracting the instantiation of the feature we receive `automobile` and `car`, respectively only `automobile`. Further as stated before also domain-ontology features can be included through this step e.g. `auto:speed`, which is `fast` for both the first and second entity.

Next, the comparison of objects can normally be expressed through exact logical rules. Equality can easily be checked by comparing the unique resource identifiers (URI); for sets one can use the Dice-coefficient [CAFP98]. According to our assumption this is not suitable for the alignment case. We need inexact heuristical ways of comparing objects. These are functions expressing similarity on strings [Lev66], object sets [CC94], checks for inclusion or inequality, rather than exact logical identity. The heuristics can also make use of additional background knowledge e.g. the use of WordNet [Fel99] when checking the identity of labels of entities. Returning back to our example we use a similarity function based on the instantiated results, i.e. we check whether the two concept sets parent concepts of `Porsche` (`automobile` and `car`) and of `Mercedes` (only `automobile`) are the same. In the given case this is true to a certain degree, effectively returning a similarity value of 0.5. Extending the approach of previous work, we do not only compare the same features of two entities, but also allow different ones, e.g. comparing sub-concepts of one concept with super-concepts of another one (which have to be disjunct to support the equality of two entities).

Finally the individual feature-heuristic combinations indicating an alignment are combined to calculate the overall support function. This can be achieved through a simple averaging step, but also through complex aggregation functions. Thereafter the value is interpreted and the algorithm decides whether the entity pair actually aligns i.e. the two entities are semantically identical. The weights for the combination and the interpretation threshold are set manually once i.e. they are the same for every mapping run.

Further we iterate over the whole process in order to bootstrap the amount of structural knowledge. In the given approach we perform five rounds.

2.2 System

The presented approach has been implemented in Java using the capabilities of the KAON[1]-framework. The framework uses its own KAON-ontology format, but can handle RDFS imports. Using KAON allows for easy use and extension of ontology technology.

2.3 Adaptations for the Contest

There were no substantial adaptations to the actual system for the contest. Minor adaptations were needed for input and output.

The input ontologies had to be transformed into a plain RDF(S)-ontology-format only. For the given case this meant the following steps:

– Transformation of owl:Class into rdfs:Class

[1] http://kaon.semanticweb.org

– Replacement of owl:Restriction through rdfs:Class. This meant that every restriction is treated as an own (anonymous) class.
– Transformation of owl:ObjectProperty and owl:DatatypeProperty into rdf:Property.
– Further, blank nodes originating from restrictions or unions were materialized. Otherwise they would have not been internally alignable, thus reducing the alignment quality of derived classes.

For the output the saving method was adjusted in the following way:

– Deletion of KAON internal constructs, mostly language identifiers.
– Export of alignments conforming to the given alignment format.

3 Results

In this section we present the results of the individual alignment experiments. As the system can neither handle OWL constructs nor alignments beyond identity, we expect to lose quality in comparison with systems that do handle both.

3.1 Experiments

Please note, that we have not performed every ontology pair of the experiment. Considering the presented algorithm and the restrictions of RDF(S), the skipped experiments were either identical to others or only provided very marginal differences. Additionally the used KAON system adds own language constructs which were filtered out for the evaluation.

3.2 Measures

We use standard information retrieval metrics to assess the different ontology pairs (cf. [DMR02]):

Precision $p = \frac{\#correct\_found\_alignments}{\#found\_alignments}$
Recall $r = \frac{\#correct\_found\_alignments}{\#existing\_alignmentss}$
F-Measure $f_1 = \frac{2pr}{p+r}$, the harmonic mean of precision and recall

Depending on the application scenario different evaluation values can be of importance. The following results therefore always indicate the results for the best possible precision, the best possible f-measure, and the best possible recall.

Nevertheless to allow strict comparability across the EON tests and with other algorithms we provide the numbers for a fixed similarity threshold across all tests, in our case this was 0.148. The threshold has been determined through a big number of tests beyond this EON experiment. If no other information about the application is given these numbers can be seen as the most objective and comparable ones.

3.3 101 concept test: ID

	precision	recall	f-measure
best precision	1.0	1.0	1.0
best f-measure	1.0	1.0	1.0
best recall	1.0	1.0	1.0
fixed threshold	1.0	1.0	1.0

The system loads the ontologies. If the URIs of two entities are identical, which is the case here, this automatically means that the entities are identical. No alignment is required for this.

3.4 102 concept test: ?

	precision	recall	f-measure
best precision	undefined	undefined	undefined
best f-measure	undefined	undefined	undefined
best recall	undefined	undefined	undefined
fixed threshold	undefined	undefined	undefined

No alignments were retrieved by the system. Having no outcome it is not possible to calculate the presented evaluation measures. This is the expected outcome when trying to align two ontologies of completely different domains.

3.5 201 systematic: no names

	precision	recall	f-measure
best precision	1.0	0.2826	0.4407
best f-measure	0.9057	0.5217	0.6621
best recall	0.9057	0.5217	0.6621
fixed threshold	0.9524	0.4348	0.5971

This represents one of the most difficult runs. The entry point of our process is identification of equal entities through similar labels. Only then further alignments can be extracted via the structure. This is especially a problem for our system, because apart from the labels we do not consider further documentation or comments.

3.6 204 systematic: naming conventions

	precision	recall	f-measure
best precision	1.0	1.0	1.0
best f-measure	1.0	1.0	1.0
best recall	1.0	1.0	1.0
fixed threshold	1.0	1.0	1.0

Smaller differences in label naming are either already filtered out by the used syntactic similarity of strings based on the edit distance. Apart from this entity pairs experience massive structural identity support from neighboring entity pairs, who have previously been identified as equal.

3.7 205 systematic: synonyms

	precision	recall	f-measure
best precision	1.0	0.1196	0.2136
best f-measure	0.8730	0.5978	0.7097
best recall	0.8261	0.6196	0.7125
fixed threshold	0.8710	0.5870	0.7012

Synonyms are more challenging than the inexact naming of the previous data set. Our system can only identify alignments based on similar labels at first. Currently we do not use any dictionary. Having completely different words means no matches in the first place.

3.8 206 systematic: foreign names

	precision	recall	f-measure
best precision	1.0	0.0109	0.0215
best f-measure	0.9677	0.6522	0.7792
best recall	0.9677	0.6522	0.7792
fixed threshold	0.98	0.5326	0.6901

There is basically no difference for the system of not knowing a synonym term or not knowing a term because it is written in a different language. It seems that French expressions are more similar to the basic English ontology than the English synonyms are.

3.9 223 systematic: expanded hierarchy

	precision	recall	f-measure
best precision	1.0	0.9783	0.9890
best f-measure	1.0	0.9783	0.9890
best recall	0.9891	0.9891	0.9891
fixed threshold	0.9891	0.9891	0.9891

The additional classes in the hierarchy don’t affect the system very much. Obviously the labeling is strong enough to not be distracted by additional classes.

3.10 224 systematic: no instances

	precision	recall	f-measure
best precision	1.0	1.0	1.0
best f-measure	1.0	1.0	1.0
best recall	1.0	1.0	1.0
fixed threshold	1.0	1.0	1.0

Again the labels seem to be strong enough themselves. Removing instances doesn’t affect the outcome. However, it will though when the labels become less reliable as in the synonym or foreign language case.

3.11 230 systematic: flattening entities

	precision	recall	f-measure
best precision	1.0	1.0	1.0
best f-measure	1.0	1.0	1.0
best recall	1.0	1.0	1.0
fixed threshold	1.0	1.0	1.0

Removing some entities doesn't affect the evaluation. Besides the fact of having less alignments, all aligning is still correct.

3.12 301 real ontology: BibTeX/MIT

	precision	recall	f-measure
best precision	1.0	0.0806	0.1471
best f-measure	0.9231	0.3871	0.5455
best recall	0.9231	0.3871	0.5455
fixed threshold	0.9231	0.3871	0.5455

The real interesting evaluations are, whether the system can automatically align two different ontologies covering the same domain. Unfortunately the results are not too promising. We do not receive high levels of quality. Even if we remove the special alignment subsumption, which our system can't extract, the results are not too good. Obviously the good labels are superposed by the big differences in structure. One consequence could be to reduce the impact of structure. People describe their world through natural language rather than through intensional structures. As another consequence one could raise the question: Despite the same labels, are the entities actually the same, as they have a very different intensional structure?

3.13 302 real ontology: BibTeX/UMBC

	precision	recall	f-measure
best precision	1.0	0.2	0.3333
best f-measure	1.0	0.2	0.3333
best recall	1.0	0.2	0.3333
fixed threshold	1.0	0.2	0.3333

The same comments as for the previous experiment still hold. Having a look at the UMBC ontology one can see that they have many bibliographic properties pointing from publications to literals/Strings. With the label being different and the range of a property being different the system can not identify these relations as equal. The classes on the other hand were well identified. Interesting for this one is the very high precision with a very low recall. Our system obviously prefers losing information for the sake of keeping the error number small. If the alignments are used for further mapping with inferencing high error rates can seriously harm the results.

3.14 303 real ontology: Karlsruhe

	precision	recall	f-measure
best precision	1.0	0.24	0.3871
best f-measure	0.9512	0.78	0.8571
best recall	0.9512	0.78	0.8571
fixed threshold	0.9286	0.78	0.8478

Karlsruhe and INRIA seem to have a closer understanding of the bibliographic world than MIT and UMBC with INRIA.

3.15 304 real ontology: INRIA

	precision	recall	f-measure
best precision	1.0	0.7692	0.8670
best f-measure	0.9867	0.9487	0.9673
best recall	0.9494	0.9615	0.9554
fixed threshold	0.9867	0.9487	0.9673

Comparing one INRIA ontology with another INRIA one yields very good results, as one can expect.

4 General Comments

4.1 Lessons Learnt

From the experiments one can generally say that labels have a high value for the alignment process. If these labels are the same the probability that they actually indicate the same is very high. Humans tend to express the same things through the same natural language; structure is only modelled in the second place. When having two ontologies where many labels are the same it might make sense to stop at this point or at least fix the alignments before continuing with the structure.

On the other hand if the labels differ, just a little or even considerably, the structural elements become more important. Through structural similarity the differences in labelling can be overcome to a certain degree.

4.2 Comments on the test cases

It is always difficult to create good test cases. The cases covered a wide range of discrepancies occurring when having two ontologies. For the cases always one feature was explicitly changed and the alignment algorithm had to cover this. However in a real world scenario, as presented by the last four test cases, the ontologies vary in many features at the same time. There will always be an optimal algorithm to circumvent one feature of ontology mismatch. The real challenge is to manage many differences at the same time. Focus should therefore lie more on real world ontologies.

4.3 Comments on measures

The measures always reflect the goal one wants to reach. In this paper the measures were purely qualitative. And even there it is difficult to define a *good* result. For querying retrieving more false answers is tolerable, as long as the correct answers are included (high recall). If further inferencing is the goal, every mistake seriously changes the whole result. Here we need a high precision. The used f-measure is a compromise to handle both scenarios adequately. It might also make sense to measure several points rather than just the f-measure.

Besides the qualitative measures it can also be interesting to measure scalability and efficiency (e.g. for on the fly alignment in peer-to-peer systems).

5 Raw Results

The results presented in this paper can be downloaded at http://www.aifb.uni-karlsruhe.de/WBS/meh/mapping/

Not all alignments in the result files are actually interpreted as an alignment; too low values are removed through the cut-off measure. Nevertheless, for better interpretability the whole files were put online.

For a better overview the results of all the tests are shown once more in one table, each time using the fixed similarity threshold:

	precision	recall	f-measure
101 ID	1.0	1.0	1.0
102 ?	undefined	undefined	undefined
201 no names	0.9524	0.4348	0.5971
204 naming conventions	1.0	1.0	1.0
205 synonyms	0.8710	0.5870	0.7012
206 foreign names	0.98	0.5326	0.6901
223 expanded hierarchy	0.9891	0.9891	0.9891
224 no instances	1.0	1.0	1.0
230 flattening entities	1.0	1.0	1.0
301 BibTeX/MIT	0.9231	0.3871	0.5455
302 BibTeX/UMBC	1.0	0.2	0.3333
303 Karlsruhe	0.9286	0.78	0.8478
304 INRIA	0.9867	0.9487	0.9673

6 Conclusion

We can draw the following conclusion from the experiment. The Karlsruhe process performs differently on the various kinds of ontology pairs. Even though the results were not too bad in general, some optimization, possibly by performing different actions for different alignment steps. Nevertheless ontology alignment will never yield 100% correct results, and any application relying thereon should be aware of this.

References

[CAFP98] S. V. Castano, M. G. De Antonellis, B. Fugini, and C. Pernici. Schema analysis: Techniques and applications. *ACM Trans. Systems*, 23(3):286–333, 1998.

[CC94] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman and Hall, 1994.

[DMR02] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the second int. workshop on Web Databases (German Informatics Society)*, 2002.

[ES04a] Marc Ehrig and Steffen Staab. QOM - quick ontology mapping. In *Proceedings of the Third International Semantic Web Conference (ISWC-2004)*, Hiroshima, Japan, 2004.

[ES04b] Marc Ehrig and York Sure. Ontology mapping - an integrated approach. In *Proceedings of the First European Semantic Web Symposium, ESWS 2004*, volume 3053 of *Lecture Notes in Computer Science*, pages 76–91, Heraklion, Greece, May 2004. Springer Verlag.

[Fel99] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. 1999.

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

Ontology alignment with OLA

Jérôme Euzenat [1], David Loup [2], Mohamed Touzani [3], Petko Valtchev [4]

Abstract

Using ontologies is the standard way to achieve interoperability of heterogeneous systems within the Semantic web. However, as the ontologies underlying two systems are not necessarily compatible, they may in turn need to be aligned. Similarity-based approaches to alignment seems to be both powerful and flexible enough to match the expressive power of languages like OWL. We present an alignment tool that follows the similarity-based paradigm, called OLA. OLA relies on a universal measure for comparing the entities of two ontologies that combines in a homogeneous way the entire amount of knowledge used in entity descriptions. The measure is computed by an iterative fixed-point-bound process producing subsequent approximations of the target solution. The alignments produce by OLA on the contest ontology pairs and the way they relate to the expected alignments is discussed and some preliminary conclusions about the relevance of the similarity-based approach as well as about the experimental settings of the contest are drawn.

1 Presentation of the system

The ontology alignment tool OLA (for *O*WL-*L*ite *A*lignment) is jointly developed by the teams at DIRO, University of Montréal and INRIA Rhône Alpes. Its main features are presented below (see also [5, 6]).

1.1 General purpose statement

OLA is dedicated to the alignment of ontologies expressed in OWL, with an emphasis on its restricted dialect, called OWL-Lite.

1.1.1 Functional specifications

More than a simple tool for automated alignment construction, OLA is designed as an environment for manipulating alignments. Indeed, in its current version, the system offers the following services:

- parsing and visualization of (pairs of) ontologies,
- automated computation of similarities between entities from different ontologies,
- automated extraction of alignments from a pair of ontologies,
- manual construction of alignments,
- initialization of automated alignment construction by an existing alignment,
- visualization of alignments,
- comparison of alignments.

In the following sections we focus exclusively on the automated alignment construction and the related services offered by OLA.

[1] INRIA Rhône-Alpes, `Jerome.Euzenat@inrialpes.fr`
[2] Université de Montréal
[3] Université de Montréal
[4] Université de Montréal, `Petko.Valtchev@umontreal.ca`

1.1.2 Basic assumptions

Universality : All the available knowledge about ontology entities should be taken into account when aligning.

Automation : We required the alignment mechanisms to be of the highest possible automation degree. In other terms, although the entire alignment process may be set on a semi-automated basis, the production of an alignment should not require user intervention at the intermediate steps. Thus, we expect the user to provide a minimal set of parameters for the alignment process whereas the tool will suggest one or more candidate alignments at the end. This may be performed in a loop aimed at establishing the optimal parameters for a specific case or domain. Automation may be used for optimal parameter learning as well.

Uniform comparison : Following the syntactic structure of the OWL language, entities are divided into categories, e.g., *classes*, *objects*, *properties*, *relations*, and only entities of the same category are compared. Moreover, the entities of a category are compared using the same similarity function and on the same feature space. In other words, for each pair of entities of a given category, the same set of similarity factors are considered and the respective contributions of those factors to the overall similarity of the pair are combined in a way that depends only on the category.

Comparability of similarity results : To enable comparison of similarity scores between different alignment tasks, the values of the similarity measure are normalized. It is noteworthy that normalization is enforced throughout the entire iterative computation process via an appropriate function definition. Moreover, useful properties of the function as proximity measure are ensured such as *positiveness*, maximalness[5], and *symmetry*[6].

1.1.3 Specific restrictions

- Primary focus is on a rather restricted sublanguage, OWL-Lite. However, some constructs from the richer OWL-DL are also supported. As a long-term goal, the coverage of the entire OWL-DL language will be sought.
- No inference is performed on the ontology, in particular inheritance is not used to expand entity descriptions. This choice has been motivated by efficiency considerations. It could be easily altered by applying a limited form of reasoning to the available descriptive knowledge as a pre-processing step.
- Only descriptive knowledge is taken into account: The similarity of an entity pair depends on all the similarities of neighbor pairs whose members **describe** the respective initial entities. In other terms, given two neighbor entities e_1 and e_2, e_2 may appear in a similarity expression for e_1 if the link between both is considered as a part of the description of e_1. For instance, we consider that a data type is not described by a property whose range the datatype represents. Consequently, datatypes are compared in an ontology-independent manner.
- Entity category separation is enforced in similarity definition: Only entities from the same category are compared for similarity and hence for alignment. Thus, classes from the first ontology are compared to classes from the second one, and datatypes to datatypes, respectively[7].

1.2 Specific techniques used

OLA relies on an all-encompassing similarity measure that is defined by a system of quasi-linear equations. Its actual values are computed as the fixed point of an iterative approximation process which starts with a lexical similarity measure and gradually brings in contributions from structure comparing functions. The entire computation process is supported by a graphical representation of the ontology structure, the OL-Graph of the ontology.

1.2.1 OL-Graph construction

To provide an easy-to-process inner representation of OWL ontologies, we use graph structure that we called OL-Graph. An OL-Graph is a labeled graph where vertices correspond to OWL

[5] With normalization, this amounts to forcing scores of 1 for identical entities within identical ontologies

[6] The price to pay for symmetry is the impossibility of detecting subsumption by this purely numerical procedure.

[7] However, some test cases, e.g., the alignment of ontology 301, suggest that a class may be more advantageously aligned to a datatype.

entities and edges to inter-entity relationships. As described in [6], the set of different vertex categories is: class (C), object (O), relation (R), property (P), property instance (A), datatype (D), datavalue (V), property restriction labels (L). Moreover, distinction is made between datatype relations (R_{dt}) and object relations (R_o), as well as between datatype properties (P_{dt}) and object ones (P_o).

The relationships expressed in the OL-Graph are:

- *specialization* between classes or relations ($\mathcal{S}$),
- *instanciation* ($\mathcal{I}$) between objects and classes, property instances and properties, values and datatypes,
- *attribution* ($\mathcal{A}$) between classes and properties, objects and property instances;
- *restriction* ($\mathcal{R}$) expressing the restriction on a property in a class,
- *valuation* ($\mathcal{U}$) of a property in an object.

The OL-Graph of an ontology is constructed after the ontology is parsed and its entities and their relationships extracted. So far, we use the OWL API [1] for the parsing of OWL files, but other possibilities remain open. It is noteworthy that OL-Graph supports well inference process. For instance, the graphs of an ontology can be easily extended with the descriptive knowledge derived by inheritance between classes or relations, or by saturation following the property types (e.g., by adding transitivity arcs for a `owl:TransitiveProperty`). Further details on OL-Graph construction will be given in [8]. It does not, however, scale to OWL-Full.

1.2.2 Integrative similarity measure

Our similarity model assigns a specific function to each node category in the OL-Graph. The functions are designed in a way to cover the greatest possible part of the available descriptive knowledge about a couple of entities. Thus, given a category X, the similarity of two nodes from X depends on:

- the similarities of the terms used to designate them (may be URIs, labels, names, etc.),
- the similarity of the pairs of neighbor nodes in the respective OL-Graphs that are linked by edges expressing the same relationships (e.g., class node similarity depends on similarity of superclasses, of property restrictions and of member objects),
- the similarity of other local descriptive features depending on the specific category (e.g., cardinality intervals, property types)

Datatype and datavalue similarities are external to our model. As such, they are provided by the user or measured by a standard function (e.g., string identity of values and datatype names/URIs).

Formally, given a category X together with the set of relationships it is involved in, $\mathcal{N}(X)$, the similarity measure $Sim_X : X^2 \rightarrow [0, 1]$ is defined as follows:

$$Sim_X(x, x') = \sum_{\mathcal{F} \in \mathcal{N}(X)} \pi^X_{\mathcal{F}} MSim_Y(\mathcal{F}(x), \mathcal{F}(x')).$$

The function is normalized, i.e., the weights $\pi^X_{\mathcal{F}}$ sum to a unit, $\sum_{\mathcal{F} \in \mathcal{N}(X)} \pi^X_{\mathcal{F}} = 1$. for the computability For instance, for two classes c, c' :

$$\begin{aligned} Sim_C(c, c') = {} & \pi^C_L sim_L(\lambda(c), \lambda(c')) \\ & + \pi^C_{\mathcal{I}} MSim_O(\mathcal{I}(c), \mathcal{I}'(c')) \\ & + \pi^C_{\mathcal{S}} MSim_C(\mathcal{S}(c), \mathcal{S}'(c')) \\ & + \pi^C_{\mathcal{A}_{dt}} MSim_P(\mathcal{A}_{dt}(c), \mathcal{A}'_{dt}(c')) \\ & + \pi^C_{\mathcal{A}_o} MSim_P(\mathcal{A}_o(c), \mathcal{A}'_o(c')) \end{aligned}$$

The set functions $MSim_Y$ compare two sets of nodes of the same category. They are presented in the next paragraph. Table 1 illustrates the set of similarities in our model.

1.2.3 Similarity-based matching of entity sets

In order to ensure equity between factors in $Sim_X(n_1, n_2)$, all similarities of pairs linked with the same type of links are combined into a unique value. This is achieved by means of a generic set similarity function $MSim$. Its arguments are two sets S_1 and S_2 of entities of the same category Y and the respective measure Sim_Y. The result is an average of the similarities of a

Funct.	Node	Factor	Measure
Sim_O	$o \in O$	$\lambda(o)$	sim_L
		$a \in A, (o, a) \in \mathcal{A}$	$MSim_A$
Sim_A	$a \in A$	$r \in R, (a, r) \in \mathcal{R}$	Sim_R
		$o \in O, (a, o) \in \mathcal{U}$	$MSim_O$
		$v \in V, (a, v) \in \mathcal{U}$	$MSim_V$
Sim_V	$v \in V$	value literal	type dependent
Sim_C	$c \in C$	$\lambda(c)$	sim_L
		$p \in P, (c, p) \in \mathcal{A}$	$MSim_P$
		$c' \in C, (c, c') \in \mathcal{S}$	$MSim_C$
sim_D	$d \in D$	$\lambda(r)$	XML-Schema
Sim_R	$r \in R$	$\lambda(r)$	sim_L
		$c \in C, (r, \texttt{domain}, c) \in \mathcal{R}$	$MSim_C$
		$c \in C, (r, \texttt{range}, c) \in \mathcal{R}$	$MSim_C$
		$d \in D, (r, \texttt{range}, d) \in \mathcal{R}$	Sim_D
		$r' \in R, (r, r') \in \mathcal{S}$	$MSim_R$
Sim_P	$p \in P$	$r \in R, (p, r') \in \mathcal{S}$	Sim_R
		$c \in C, (p, \texttt{all}, c) \in \mathcal{R}$	$MSim_C$
		$n \in \{0, 1, \infty\}, (p, \texttt{card}, n) \in \mathcal{R}$	equality

Table 1. Similarity function decomposition (`card` = `cardinality` and `all` = `allValuesFrom`).

limited subset of the product $S_1 \times S_2$. The subset represents a matching that optimizes the total similarity [9]:

$$MSim_C(S_1, S_2) = \frac{\sum_{\langle c_1, c_2 \rangle \in Pairing(S_1, S_2)} Sim_C(c_1, c_2)}{max(|S_1|, |S_2|)}$$

where $Pairing(S_1, S_2)$ is the optimal matching. For normalization reasons, the sum of the similarities of the pairs in $Pairing(S_1, S_2)$ is divided by the size of the larger set.

1.2.4 Equation system definition and iterative resolution

As many of the relationships are both-ways, it may be impossible to follow standard procedures in computing the similarity values. Indeed, the recursive definition of the similarity may easily lead to circular dependancies of the similarity values for two or more node pairs. In such cases, an equation system is composed (see [2, 9]) out of the similarity definitions where variables correspond to similarities of node pairs while coefficients come from weights.

Because of the uncertainty due to the matching functions whose outcome cannot be fixed *a priori*, the resulting system is not linear and therefore cannot be solved in a direct way. Instead, an iterative method is used to approximate the solution (which always exist) as the fixed point of a vector function. The process starts with the local similarity, i.e., the one computed without looking at neighbor nodes. It then integrates neighbor similarities and lets them grow as a result of mutual influence. The growth is steady at each step of the iterative process but is nevertheless limited from above since neither of the functions from table 1 can reach values greater than 1. Thus, the process necessarily ends with a vector fixed point whose components are the similarity values sought.

1.2.5 Lexical similarity measures

OLA relies on WordNet 2.0 [7] for comparing identifiers. For that purpose, it applies a measure of "relatedness" between two terms. Given a pair of identifiers, the lexical similarity mechanisms retrieve the sets of synonyms (the *synsets*) for each term. A normalized Hamming distance is then applied to these sets. A variant of the substring distance is used to establish a default similarity value for identifier pairs. Such a default mechanism allows identifiers that are not entries in WordNet, e.g., compound identifiers or abbreviations, to be processed in a sensible way.

1.3 Implementation

OLA is implemented in JAVA. Its architecture follows the one of the Alignment API and the recent implementation that was described in [4]. OLA relies on the OWL API [1] for parsing OWL files. An entire subsystem is dedicated to the onstruction of OL-Graphs on top of the parsed ontologies. A set of further components offer similarity computation services: substring distances, edit distances, Hamming distance, etc. A specific component extracts similarity values from the limited WordNet interface provided by the JWNL library [3]. Similarity-based matching between sets of entities is performed by another component. Similarity and matching mechanisms are integrated into the alignment producing subsystem which supports the entire iterative computation

process. Finally, the VISON subsystem provides a uniform interface to all the automated tools and visualizes both the input data, i.e., the OL-Graphs, and the final result, i.e., the alignment.

1.4 Adaptation made for the proposal

Several changes have been made to fit the complexity of the comparison. The most noteworthy one is the abandon of the requirement that all entities of the same category are compared along the same feature space.

1.4.1 Adaptive description space

We found that the "uniform factor weights" condition tends to favor pairs of entities that have complete descriptions, i.e., pairs where both the members are connected to at least one descriptive entity for each of the similarity factors in the respective formula. Conversely, pairs where a particular factor is void tend to score to lesser similarity values. The extreme case is the pair of `Thing` classes which, if present, usually have almost no description. With fixed weights for similarity factors, and hence universal feature space for comparison, the `Thing` class pair will be evaluated to a relatively weak similarity value and the chances are high for it to be skipped from the alignment.

For the above reasons, we decided to limit the comparison of two entities to the strict set of factors which are non void in both. This has been achieved in an uniform way, i.e., through a division of the initial linear combination formula by the sum of the weights of all non-void factors. Thus, for a category X, the similarity measure $Sim_X^+ : X^2 \rightarrow [0, 1]$ becomes:

$$Sim_X^+(x, x') = \frac{Sim_X(x, x')}{\sum_{\mathcal{F} \in \mathcal{N}^+(x,x')} \pi_{\mathcal{F}}}$$

where $\mathcal{N}^+(x, x')$ is the set of all relationships $\mathcal{F}$ for which $\mathcal{F}(x) \cup \mathcal{F}(x') \neq \emptyset$ [8].

1.4.2 Lexical similarity measure

The initial straightforward similarity measure has been replaced by a more sophisticated one that better accounts for semantic proximity between compound identifiers. Thus, given a pair of identifiers, they are first "tokenized", i.e., split into a set of atomic terms. Then, the respective pairs of terms are compared using WordNet. In fact, their degree of relatedness is computed as the ratio between the depth of the most specific common hypernym and the sum of both term depths. Finally, a similarity-based match is performed to establish a degree of proximity between the sets of terms.

1.4.3 Weight finding mechanism

To increase the level of automation in OLA, a weight-search mechanism was added to the initial architecture. Indeed, it is far from obvious for a novice user how to weight the different similarity factors. The underlying module performs several runs of the alignment producing subsystem with various weight combinations. It keeps only the combination that has resulted in the best alignment, i.e., the one of the highest total similarity between aligned entities. On the one hand, this procedure is not realistic in a setting where reference alignments are not given. On the other hand, if the tests a realistic, then what is learned is the best behaviour of the system in general.

2 Results

The test protocol was as follows. We first looked for the typical weight combinations with an exhaustive search on a small subset of test cases. The resulting combinations were then applied systematically to the rest of the ontology pairs. Whenever the results were unsatisfactory, exhaustive search was applied to the neighborhood of the best scoring typical combinations. Here we provide some details on the combinations that were mined out by OLA as well as a brief comment for every single test indicating the combination of parameters that led to the best scoring alignment. A summary of the results obtained with equal weights for all factors in a category is provided at the end as well.

[8] That is, exists at least one y such that $(x, y) \in \mathcal{F}$ or at least one y' such that $(x', y') \in \mathcal{F}$.

2.1 Preliminary tests

The optimal weight searching engine of OLA was run on a small subset of ontologies that seemed to represent the extreme cases. The resulting matchings were compared to the respective expected alignments according to the contest guidelines. The underlying weight combinations and their respective alignment scores were then analyzed to discover possible trends. For this preliminary experiment, the step in the variation of the specific weight values was set to 0.2 while the total of all weights in a category was set to 1. This value provided a good trade-off between the range of variation for each single weight (i.e., a five-grade scale) and the number of combinations to be tested. Actually, there are 8 categories with 3, 4 or 5 weights. To bring down the resulting combinatorial explosion, we used the same weight combination for entity categories sharing the same set of similarity factors, e.g., datatype and object properties.

The results of this step suggested that there were three weight combinations that can lead to the best scoring alignment for a test case:

- equal or nearly equal weights for all factors,
- one factor is assigned the total weight of 1 while the other weights are set to 0,
- the total weight of 1 is divided into two non-zero parts assigned to two factors, the remaining factors are given zero weights.

In what follows, we indicate for each test the weight combination that led to the best alignment with respect to precision and the lexical similarity used. To provide an idea about the average performances of OLA, we include also a summary of the scores obtained with perfectly equal weights in every entity category (i.e., 0.2-step constraint relaxed). It is noteworthy that the overwhelming majority of the results where precision is below 1.0 are mere lower bounds and may well be improved through an exhaustive search in the weight combination space. Moreover, in each test, our tool aligned all the named entities of the basic ontology to the most similar entity of the compared ontology. Therefore, the recall scores, which depend on the size of the proposed alignment, are relatively low.

2.2 Concept

In this group of tests, the string distance was systematically used for lexical comparisons.

2.2.101 Identity

The best alignment was obtained with unit weight to lexical similarity and zero weights to the remaining factors for all categories.

Precision	Recall	Fallout
1.0	0.611	0.0

2.2.102 Irrelevant ontology

OLA used equal weights to obtain the following result that proved best.

Precision	Recall	Fallout
1.0	N/A	0.0

2.2.103 Language generalization

The best combination assigns for each category 0.4 weight to the lexical similarity and 0.6 to the factor representing the links to more general entities (e.g., the super classes of a class, the class for an individual, the relation for a property restriction, etc.).

Precision	Recall	Fallout
1.0	0.611	0.0

2.2.104 Language restriction

The combination that scored best is identical to the one described in the previous paragraph.

Precision	Recall	Fallout
1.0	0.611	0.0

2.3 Systematic

2.3.201 No names

Equal weights were used together with string distance.

Precision	Recall	Fallout
0.714	0.436	0.286

2.3.202 No names, no comment

The same settings as in test 201 were used.

Precision	Recall	Fallout
0.626	0.383	0.374

2.3.204 Naming conventions

are used for labels.

The same settings as in test 201 were used.

Precision	Recall	Fallout
0.901	0.550	0.099

2.3.205 Synonyms

Equal weights and WordNet led to the best precision alignment.

Precision	Recall	Fallout
0.802	0.490	0.198

2.3.206 Foreign names

The settings used were identical to those of test 205.

Precision	Recall	Fallout
0.761	0.450	0.239

2.3.221 No hierarchy

The settings used were identical to those of test 205.

Precision	Recall	Fallout
1.0	0.611	0.0

2.3.222 Flattened hierarchy

The best combination is equal to the one for test 201, except for the class category where the 0.6 weight was assigned to the instance factor. String distance was used as well.

Precision	Recall	Fallout
0.945	0.577	0.055

2.3.223 Expanded hierarchy

The same settings as in test 222 were used, with the exception of the 0.6 weight in the class category which was assigned to the datatype property factor.

Precision	Recall	Fallout
0.989	0.604	0.011

2.3.224 No instances

The same settings as in test 205.

Precision	Recall	Fallout
1.0	0.968	0.0

2.3.225 No restrictions

The same settings as in the test 222 were used.

Precision	Recall	Fallout
1.0	0.611	0.0

2.3.228 No properties

Once again, the winning combination had equal weights for all factors with string distance.

Precision	Recall	Fallout
1.0	0.375	0.0

2.3.230 Flattening entities

The winning weight combination was the one of test 222 but with WordNet-based similarity.

Precision	Recall	Fallout
0.946	0.476	0.054

2.4 Real ontologies

2.4.301 BibTeX/MIT

The exclusively lexical comparison, e.g., weight of 1.0 to the lexical similarity factors in all categories, which was supported by WordNet produced the best alignment in this case.

Precision	Recall	Fallout
0.623	0.513	0.377

2.4.302 BibTeX/UMBC

Same settings as in the 301 test.

Precision	Recall	Fallout
0.542	0.245	0.458

2.4.303 Karlsruhe

The same settings as in test 205.

Precision	Recall	Fallout
0.5	0.311	0.5

2.4.304 INRIA

Same settings as in the 301 test.

Precision	Recall	Fallout
0.671	0.315	0.329

2.5 Summary of equal-weight results

Figure 1 summarizes the results obtained by OLA with equal weight combinations.

Test Nbr	Name	Lex. Sim.	Precision	Recall	Fallout
101	Id	SD	0.97	0.59	0.03
102	Irrelevant	SD	1.0	N/A	0.0
103	Language Generalisation	SD	0.901	0.550	0.099
104	Language Restriction	SD	0.912	0.557	0.088
201	No Names	SD	0.714	0.436	0.286
202	No Names, No Comments	SD	0.626	0.383	0.374
204	Naming Conventions	SD	0.901	0.550	0.099
205	Synonyms	WN	0.802	0.490	0.198
206	Foreign Names	WN	0.761	0.450	0.239
221	No Hierarchy	WN	1.0	0.611	0.0
222	Flattened Hierarchy	WN	0.901	0.550	0.099
223	Expanded Hierarchy	WN	0.967	0.590	0.033
224	No Instances	WN	1.0	0.968	0.0
225	No Restrictions	WN	0.967	0.590	0.033
228	No Properties	SD	1.0	0.375	0.0
230	Flattening Entities	WN	0.92	0.463	0.08
301	BibTeX/MIT	WN	0.607	0.493	0.393
302	BibTeX/UMBC	WN	0.5	0.226	0.5
303	Karlsruhe	WN	0.5	0.311	0.5
304	INRIA	WN	0.618	0.439	0.382

Figure 1. Results of the alignment with equal weights. Lexical similarity codes: WN stands for similarity based on WordNet and SD for (inverted) string distance.

3 General comments

3.1 Comments on the results (strength and weaknesses)

According to experimental results, our algorithm performs well when the structure of the compared ontologies are closed or identical (tests 10X and 22X).

3.2 Discussions on the way to improve the proposed system

Many of the initial assumption and constraints have proven to be a hamper for the establishment of precise alignments. Here is a discussion of points that could not be corrected during the test period but that we shall look at in the aftermath.

3.2.1 Limited inference in OL-Graph construction

In the construction process of our inner representation of the ontology we plan to expand class, relation and property nodes with the description knowledge they inherit from the super entities. Similarly, in the case where class restrictions fix property values, these values will be brought down to the descriptions of class instances.

3.2.2 Inter-category comparisons

An extended version of the similarity measure should allow the comparison of entities from different categories:

- classes with data types,
- object properties with datatype ones,
- objects with values.

3.3 Comments on the test cases

We fond that the proposed testbed cases cover a large portion of the situations that may arise in ontology alignment practice. However, the targeted variation of the test cases, i.e., on one specific dimension at once, is a challenge for our algorithm. Indeed, it was designed to be robust

on all features, hence no single feature is favored by the collection of weights. This does not seem to be a winning strategy with a test set that alters systematically a single feature: Whereas an algorithm that puts the emphasis on a particular feature will be negatively affected only by the test that puts noise on that feature, our own system experiences a systematic, albeit much lighter, negative impact.

3.4 Comments on measures

The proposed measures are definitely a first step in the right direction. Applying information retrieval metrics such as recall and precision seems to be a good approximation for the expectations of an alignment tool user. And the underlying model is simple enough to be understood by an average user. However, there is a price to pay for the simplicity, in particular with similarity-based alignment tool that grades the strength of an alignment cell. In fact, the counting of "hits" and "misses" ignores completely the actual strength values which may vary in large ranges.

3.5 Proposed new measures

A possibility would be to integrate the strength of the cells in the precision computation, something that could be done at low cost (e.g., by adding up strength values instead of counting).

4 Raw results

4.1 Links to the set of provided alignments (in align format)

A .zip archive of all the mentioned results together with indication of the OLA settings used for their extraction is provided at:
`http://www.iro.umontreal.ca/~owlola/align_files.html`.

4.2 Matrix format

See section 2.5.

5 Conclusions

It is still too early to draw final conclusions on the capacity of our system and our similarity-based approach in general to produce meaningful alignments on real ontologies. Indeed, the results on artificially altered ontologies only suggest the tool is robust to a single, albeit often powerful, source of noise. Further experiments will be necessary to gain deeper insight into the behavior of our alignment mechanisms. It is also noteworthy that the time allocated to the preparation of the contest was clearly insufficient to deal with all the challenging issues it has revealed. We are nevertheless very obliged to the contest organizers for their excellent initiative. Indeed, our participation effort yielded a long list of exciting problems to look at.

Despite the partiality of the picture we could draw about the performances of OLA, we would advocate for similarity as a mechanism for supporting alignment construction. It represents a good trade-off between several criteria that need to be taken into account in the design of effective alignment tools: precision of the final results, computational efficiency, good level of automation.

REFERENCES

[1] Sean Bechhofer, Raphael Voltz, and Phillip Lord. Cooking the semantic web with the OWL API. In *Proc. 2nd International Semantic Web Conference (ISWC), Sanibel Island (FL US)*, 2003.

[2] Gilles Bisson. Learning in FOL with similarity measure. In *Proc. 10th AAAI, San-Jose (CA US)*, pages 82–87, 1992.

[3] John Didion. The Java WordNet Library, http://jwordnet.sourceforge.net/, 2004.

[4] Jérôme Euzenat. An API for ontology alignment. In *Proc. 3rd ISWC*, pages 698–712, Hiroshima (JP), 2004.

[5] Jérôme Euzenat and Petko Valtchev. An integrative proximity measure for ontology alignment. In *Proc. ISWC-2003 workshop on semantic information integration, Sanibel Island (FL US)*, pages 33–38, 2003.

[6] Jérôme Euzenat and Petko Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proc. 15th ECAI*, pages 333–337, Valencia (ES), 2004.

[7] A.G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[8] Mohamed Touzani. Alignement d'ontologies dans OWL. Master's thesis, University of Montréal, in preparation.

[9] Petko Valtchev. *Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets*. Thèse d'informatique, Université Grenoble 1, 1999.

A Semantic Category Matching Approach to Ontology Alignment

Tadashi Hoshiai[1], Yasuo Yamane[1], Daisuke Nakamura[2], and Hiroshi Tsuda[1]

[1] Fujitsu Laboratories Ltd., I. T. Media Laboratories,
211-8588 Kawasaki-shi, Kanagawa, Japan
{hoshiai, yamane.yasuo, htsuda}@jp.fujitsu.com
[2] Kyoto University, Graduate School of Informatics,
606-8501 Sakyo-ku, Kyoto, Japan
daisuke@lab7.kuis.kyoto-u.ac.jp

Abstract. We applied our semantic category matching (SCM) approach to the EON ontology alignment contest problems. Our approach found pairs of semantically corresponding categories from two different classification hierarchies such as Yahoo, based on natural language processing, similarity searching of huge vector spaces, and structural consistency analysis. The EON Contest's random name problems (#201, #202) could not be solved using conventional character string resemblance techniques. However, when we applied SCM to these problems, the results showed that SCM had improved the accuracy as compared to the conventional method (F-measure: 0.021=>0.949, 0.021=>0.580). Moreover, SCM exceeded the accuracy average in all problem areas by over 10 % as compared to conventional methods.

1 Semantic Category Matching

1.1 Outline

We applied semantic category matching (SCM) technology to the ontology alignment problem. Our method found pairs of semantically corresponding categories between two different classification systems. In the integration and interoperation of classification systems, this kind of technology is important. However, there are problems that cannot be solved using only the character string resemblance method because of the difference in the category names, category granularity, and the classification hierarchy formation principles.

Related works of SCM technology are the enhanced Bayes classification method by Agrawal [1] and the Identity test method by Ichise [2]. Agrawal's work is a content-oriented statistical approach, as much as ours. However, his work does not look at the entire hierarchical structure, and so therefore it is not suitable for large hierarchy classification systems. Ichise's work is not content-oriented approach but an extension-oriented approach, based on URL identification in web directories. However, we think that content-oriented approach is necessary for semantic analysis of text information. Furthermore, their works did not treat structural consistencies between the results of the category correspondence and hierarchical structures. Since these points are important for large system services and semantic approaches, we incorporated them into our method.

Semantic category matching is based on a statistical approach that takes sample documents from each category and hierarchical structure description data, and outputs all category pairs that semantically correspond with the two classification systems. Ontology alignment is a problem designed to find couples of corresponding classes.

While the purposes of SCM and ontology alignment are different, the problem structures of both are similar to each other, from the perspective of alignment between the hierarchical structures. Therefore, we applied our new SCM technology to problems that could not be solved by usual methods.

1.2 Elemental technology

We used the following elemental techniques, which we will explain sequentially, in SCM. An outline diagram of SCM is shown in Figure 1.

1) Hierarchical version of keyword extraction,
2) Similarity search category similarities, based on oblique coordinates and,
3) Structural consistency analysis.

1.2.1 Generating category feature vectors by hierarchical keyword extraction

A keyword extraction technique statistically analyzes documents classified by category. It finds keywords, which are words that occur frequently in the documents in a specific category, but exclude common frequently occurring words that appear regularly in other categories but that have a weak relationship to the category.

In keyword extraction technology, the following premises are given:

- High statistical correlation between word occurrences in the document and their classification categories.
- Sufficient classified documents to do a statistical analysis.
- Highly correlated nouns within a category.
- Subcategory word occurrence characteristics are succeeded by super categories along the classification hierarchy.

Under the above premises, the keywords are extracted automatically, based on the statistical correlation between the document's topic category and the word occurrences. In this case, we can select criteria measures that highly evaluate only words that have a high correlation to a specific category. In our research, we used Kullback-Leibler's information as follows:

$$P(w|C)\cdot\log\frac{P(w|C)}{Q(w)}$$

w: word, C: category, P(w|C): word occurrence probability in category C
Q(w): average word occurrence probability of every brother-category of C

The keywords are extracted according to the following procedures.

input document => morphological analysis => remove unnecessary words
=> count words => total word occurrences in each category
=> inherit subcategory's statistical characteristics to super-category
=> select higher-ranking words in each category

Finally, the words whose value of the criteria measure is higher (for instance, the higher 30 words) in each category are selected as keywords. Then, the category feature vectors, based on the word occurrence characteristics of each category, were output. Moreover, because the subcategory feature is weighted and inherited to a super category, the neighborhood of the classification's hierarchical structure was reflected in the features of the keywords, and the distance in the vector space.

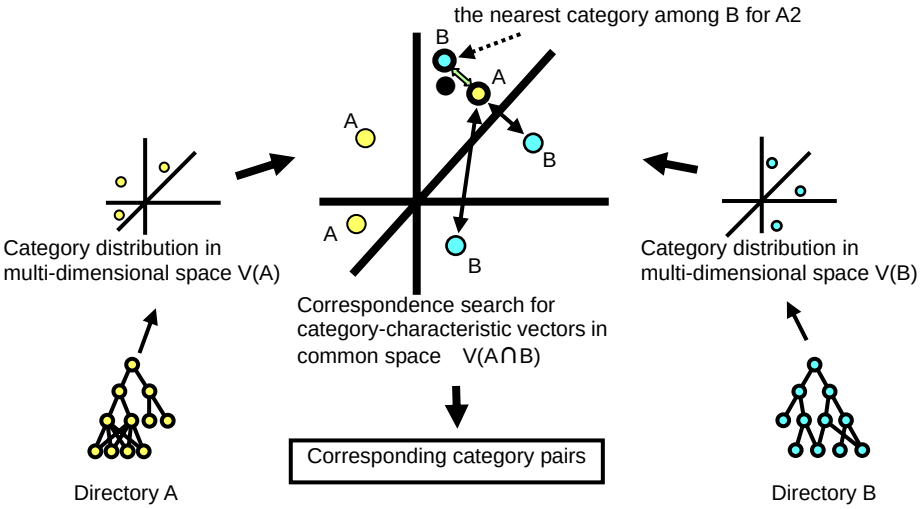


Figure 1. Outline of semantic category matching

1.2.2 Similarity category search based on oblique coordinates

Because each vector space (V(A) and V(B) in Fig. 1) formed by the two different classification systems generally has different coordinate systems (each axis coordinate corresponds to a keyword), we created and used a common feature vector space ($V(A \cap B)$ in Fig. 1) with vocabulary common to both systems, to compare the category vectors.

By using an oblique coordinate system [3] that reflects the correlationship between the keywords, the similarities in the word meanings can be woven into the coordinate system. As a result, the vector space becomes semantically more natural. For example, consider these keywords “sports”, “Olympics”, and “weather”. Because the first two words are more similar, the corresponding coordinate axes are set more closely than the last one. For each category in one classification system, the nearest neighboring categories of the other systems are output according to the order of their angle distances.

1.2.3 Structural consistency analysis

Structural consistency analysis focuses on whether the category couples correspond between the two classification systems and is formed naturally within their hierarchical structures, or not. After category couples are mutually and independently selected by a similar category search as the candidates of the results data of the entire system, it is necessary to decide which category couples are natural structural correspondences.

Figure 2 shows the naturalness of the correspondence between two hierarchical structures. When we call one of the category couples the reference couple (a solid-line arrow in Fig. 2), we can evaluate whether the neighboring couples (dotted-lines arrows in Fig. 2) can be placed well (or badly) in the hierarchical structure on both sides, by comparing them to the reference couple. Here, assuming that the categories of a reference couple are ‘a’ and ‘b’ in Fig.2, 'neighbor couple' indicates the category couple, whose category is near as link distance to another category. The link distance indicates the number of subcategory links that can be joined to ‘a’ or ‘b’.

The neighbor couple that contains subcategory ‘a1’ and ‘b1’ in Figure 2, is consistent with the reference couple (‘a’ and ‘b’) with respect to their hierarchical structures, because the category ‘a1’ is a subcategory of category ‘a’ and category ‘b1’ is subcategory of the category ‘b’. Conversely, the neighboring couple that contains subcategory ‘a2’ and ‘b2’ in Figure 2, the couple is not consistent with the reference couple with respect to their hierarchical structures, because category ‘a2’ is a super category of category ‘a’ and the category ‘b2’ is subcategory of the category ‘b’. If the degree of this consistency is provided according to a suitable measure, the structural consistency of the reference couple is obtained as the average consistency of all the neighboring couples. Finally, the structural consistency of the entire SCM is obtained as the structural consistency average of all the reference couples.

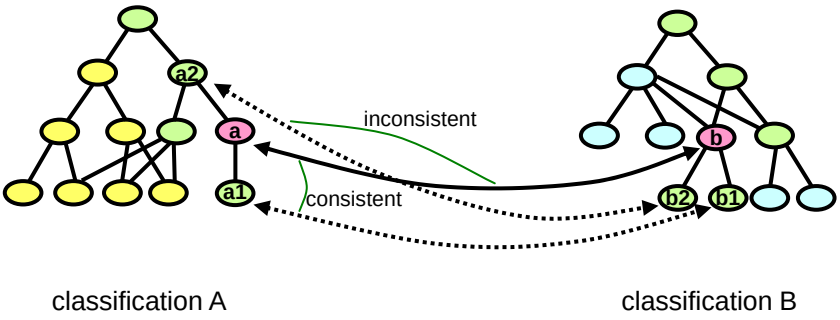


Figure 2. Structural consistency analysis

1.3 Adaptation for Semantic Category Matching

1.3.1 An SCM approach to the ontology alignment problem

Because both techniques have the same common structure from the point of view of correspondence between two hierarchical structures, we thought that we could apply SCM technique to ontology alignment (OA), even though the purpose of SCM was originally different from the purpose of OA.

Difference between ontology alignment and category matching:

Because the description unit for OA is class (or instance), and the description unit for an SCM is the category (object domain) of document topics, the granularity of OA is smaller than SCM. Furthermore, in OA, properties for both object's attributes and relations between objects, and also, the restricted condition of property can be described. On the other hand, we cannot describe any predefined logical relationships between any of the parts of a document in SCM, but XML tag's roles. Thus, the information described in OA is more detailed than the information described in SCM.

The idea of application of SCM to OA:

The class-instance relationship is common to both techniques: therefore if we interpret 'class' in ontology as 'category' in SCM, and interpret 'instance' as 'sample document', and the ontological description information is converted into a category name, the document ID, the category hierarchy relationship, and tag structure of XML documents in SCM, we can extract suitable keywords from the text of suitably selected tags in XML documents.

1.3.2 Outline of application of SCM to OA

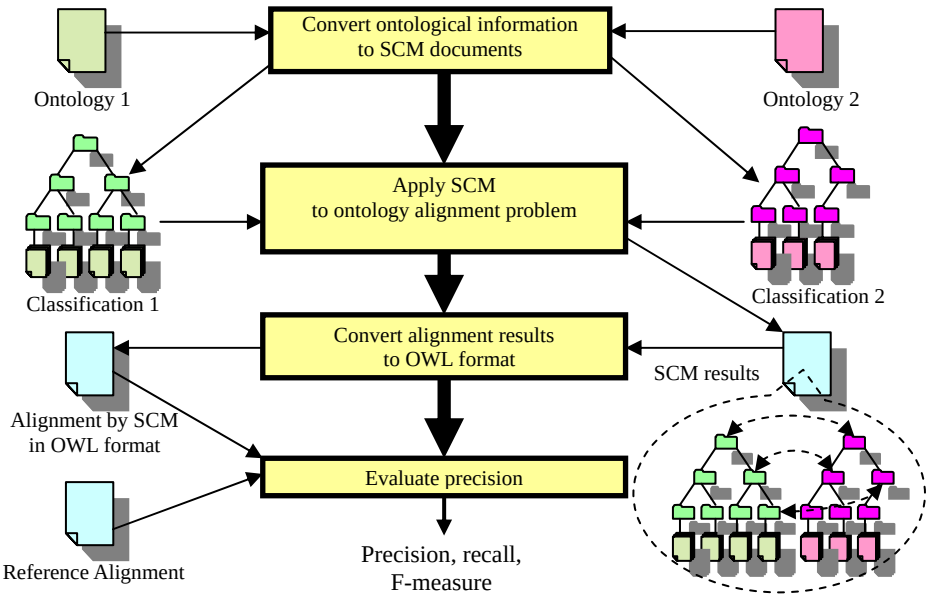


Figure 3. Application flow of SCM to OA

An outline of SCM application to the ontology alignment problem is as follows: (see Figure 3)

- The descriptions of two ontology sets (reference and target ontology) are converted into XML document sets in two classification systems and two category hierarchies in SCM format. (see Figure 4)
- SCM is applied to 2 sets of converted data to resolve the OA problem.

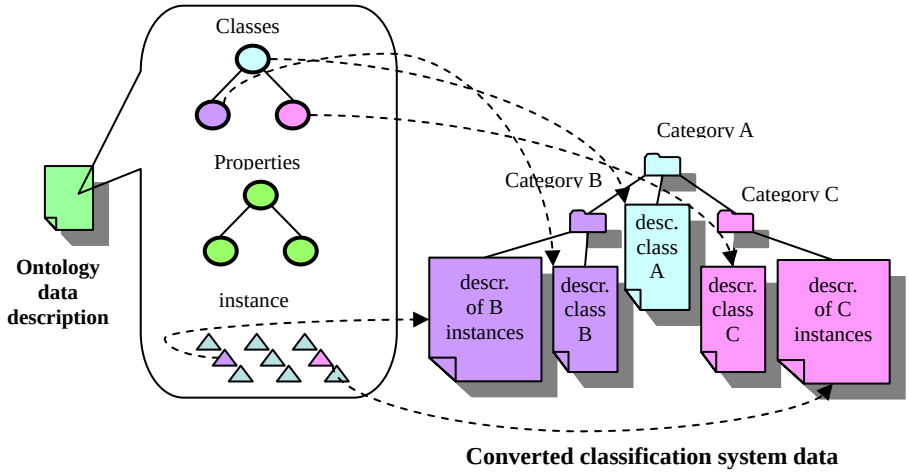


Figure 4. Conversion from ontology description to SCM input data

- SCM outputs a set of category-pairs that indicate the alignment between the two classification hierarchies. The results data are described in Ontolingua langage.
- The SCM results data are converted into the OWL alignment form of the EON contest.
- Finally, the alignment result accuracies (F-measure etc.) are calculated by ontoalign, the ontology alignment evaluation tool prepared by EON's promotion division.

2 Results

In first experiment, we applied SCM (version 1) to first version of contest test data, as much as string-based alignment method included in the ontoalign evaluation tool. Because these test data included bugs, we had to modify these data for enabling execution of programs, and so results data seem to be under a little influence of these modification. The accuracy data (F-measure) of results for applying SCM to each problem are listed in Table 1, along with the results of standard string-based alignment method.

Table 1. F-measures results of SCM and string-based methods in first experiment

test no.	101	102	103	104	201	202	204	205	206	221
String-based	.938	NaN	.948	.948	.021	.021	.753	.344	.423	.948
SCM v1	.990	NaN	.970	.980	.870	.500	.829	.579	.687	.909
Difference	.052	0	.022	.032	.849	.479	.076	.235	.264	-.039

test no.	222	223	224	225	228	230	301	302	303	304
String-based	.897	.897	.938	.948	.917	.854	.593	.411	.510	.804
SCM v1	.924	.916	.957	.978	.899	.890	.729	.468	.400	.820
Difference	.027	.019	.019	.030	-.018	.036	.136	.057	-.110	.016

Note: 'NaN' (not the answer) indicates 'division by zero' (the alignment number is zero) for calculating the F-measure. These are proper results, because test #102 has no proper alignment of classes or properties, therefore we can replace 'NaN' with 1.0 (that is, the results of the alignment are proper.).

After final revised version of test data was disclosed, we applied the little revised SCM (version 1.1) to the final version of test data in the second experiment. The results are listed in Table 2.

Table 2. F-measures results of SCM in second experiment

test no.	101	102	103	104	201	202	204	205	206	221
SCM v1.1	.995	NaN	.995	.995	.949	.580	.933	.699	.584	.925

test no.	222	223	224	225	228	230	301	302	303	304
SCM v1.1	.955	.908	.995	.995	.941	.953	.755	.468	.505	.886

For comparison between each method, the polygonal line graphs on results data of both experiments are shown in Figure 5.

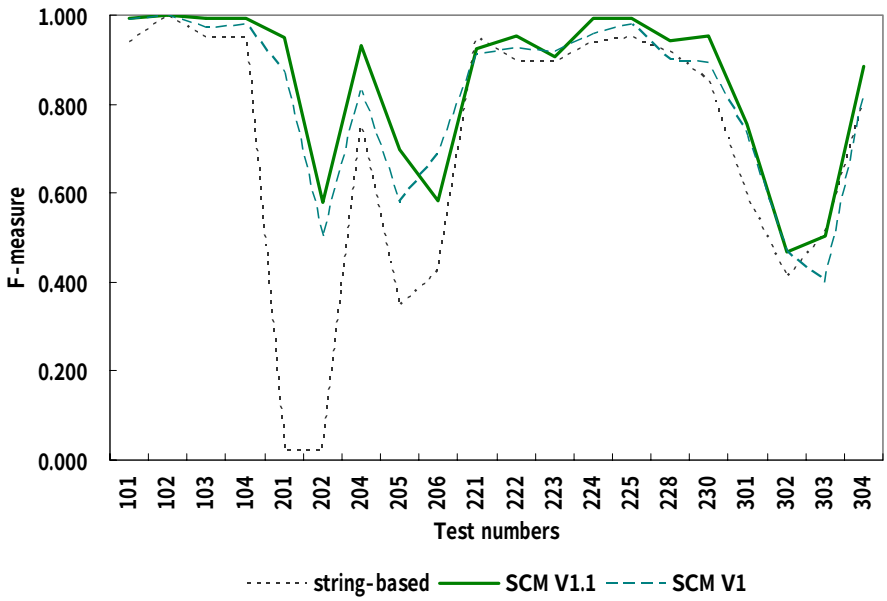


Figure 5. F-measure results of SCM and string-based method

2.1 Concept test (test no.: #101, #102, #103, and #104)

Test number 101 is a comparison test of the same ontology, and number 102 is a comparison test of quite different ontological domains (bibliography and food). The ontological structures of numbers 103 and 104 are close to that of number 101.

In test #102, results of both methods are exactly matched, that is zero alignment.

As the string-based method is based on the agreement/disagreement of the name character strings, whether or not the class names in the reference ontology are the

same as the class names in the target ontology, this method is suitable for this kind of tests. In the tests #101, #103, and #104, alignment result accuracies from the string-based methods attached to the ‘ontoalign’ evaluation tool were close to 100%.

Moreover, the SCM alignment results of other tests were generally superior to the results of the string-based methods.

By the way, the revised SCM (v1.1) results (0.995) included the alignment couple between ‘language’ property in reference ontology and ‘language’ property in target ontology. Though we think that this result is proper and results become 1.0, but the final version of reference alignment file (refalign.rdf) does not include this alignment couple.

2.2 Name diversity test (#201, #202, #204, #205, and #206)

The problems we focused on in this paper are those that involve naming diversity. These cannot be solved by string-based methods. The other hand, SCM is a content-oriented approach, and can solve these problems using content similarity between the semantically same classes in different ontologies. That is, even if there is a disagreement in the class names between both ontologies, when the description data of the classes and the instances belonged to their classes were statistically and structurally similar, we could obtain an ontological alignment. Semantic similarity of properties can be discussed as well as semantic similarity of classes.

In random name tests (#201, #202), there was no similarity in the name character strings between reference ontology and target ontology, and so the string-based method results were almost 0%. In contrast to this, the results of SCM was 87% and SCM v1.1 improved to 94.9% in the test #201 where comment sentences were available, and was 50.0% and improved to 58.0% in the test #202 where no comment sentences were available.

In test #204, #205, and #206, SCM v1.1 exceeded the string-based method by over 10%.

2.3 Hierarchy variation test (#221, #222, and #223)

In the no hierarchy test (#221), the SCM results fell below the string-based method results by a few percentage points. Conversely, in the flattened hierarchy test (#222) and the expanded hierarchy test (#223), the SCM results exceeded the string-based method by a few percent.

In SCM, because the hierarchical relationship of ‘subClassOf' is reflected in the calculation of the category feature vectors of both a super class and a subclass, the feature vectors are distributed close to each other in the vector space, even if the names in these classifications have no common character strings. In test number 221, we think the accuracy fell because of lost information in this hierarchical relationship.

2.4 Other systematic tests (#224, #225, #228, and #230)

In the no instances (#224), no restrictions (#225), no properties (#228), and flattened

entities (#230) tests, the SCM v1.1 results exceeded all of results of the string-based method by several percent.

2.5 Real ontology test (#301, #302, #303, and #304)

In BibTex/MIT test (#301), the SCM results exceeded the string-based method results by 10% or more. In the BibTex/UMBC test (#302) and BibTex/INRIA test (#304), the SCM results exceeded our expectations by several percent. In the BibTex/Karlsruhe test (#303) the SCM results fell by only 0.5%.

3 General comments

3.1 Results (strength and weaknesses)

Strength:
When there are semantically similar classes between both ontologies, even if the name of the class in one kind of ontology is different from that in another, SCM can find correspondences of these classes in both ontologies.

Weakness:
When there is little common vocabulary between the ontologies, there is a possibility that the system cannot identify the semantically similar category vectors in the feature vector space. (For example, the case there is no similarities in the instance description data.)

3.2 Improving the proposed system

Stemming:
Because we don’t process English stemming now, SCM cannot absorb inflection variations of English words (-s, -es, -ing, -ed, -er, -est, etc.). It is true that this changes the original word into a different one, thus decreasing the accuracy rate, but this influence is reduced by effects of correlation between semantically similar words in oblique coordinate vector space. Consequently we will use the stemming function in our system in the future. In our experiments, we performed Japanese morphological analysis and stemming.

3.3 New measures proposed

Path-weighted accuracy (P-measure):
Currently, we obtain an incorrect answer (accuracy 0) if the intended class is not described in results data. If there is correspondence between two classes that are semantically unrelated to the intended class, and the correspondence between two classes that are closely related to the intended class, it is clear that the latter

performance will be better than the former. Therefore, if we use the number of links between two classes of ‘subClassOf’ and define the semantic closeness between classes r (0 =< r =< 1) as accuracy, then the overall accuracy can be calculated as the average of all of accuracies of the correct answers).

4 Raw results

4.1 Links to the set of provided alignments

Currently, our company does not permit public access to URLs containing the alignment results data files.

5 Conclusions

We showed that there were large improvements in the accuracy during experiments when our category matching technology was applied to difficult ontological alignment problems, such as naming diversity. The EON contest’s random name problems (#201, #202) were difficult to solve using conventional techniques, based on character string resemblance. However, when we applied our category matching method, the SCM accuracy results showed some improvement over conventional methods (F-measure: 0.021=>0.949, 0.021=>0.580). Moreover, in all tests, the accuracy average surpassed that obtained in conventional tests by over 10 % on average.

In the future, I want to work on other ontology alignment problems and improve the accuracy of category matching technology much more.

Acknowledgements
I would like to thank Tomoya Iwakura for his suggestions and support on our program development.

References

1) Agrawal, R. and Srikant, R.: On Integrating Catalogs, in Proceedings of the Tenth International World Wide Web Conference (WWW-10), (2001) 603-612.
2) Ichise, R., Takeda, H. and Hon'iden, S.: Learning on the adjustment rules between hierarchical knowledge, Journal of JSAI, vol.17, no.3-F (2002) 230-238.
3) Yamane, Y., Hoshiai, T., Tsuda, H., Katayama, K., Ohta, M., Ishikawa, H.: Multi-Vector Feature Space Based on Pseudo-Euclidean Space and Oblique Basis for Similarity Searches of Images, in Proceedings of the First International Workshop on Computer Vision meets Databases (CVDB 2004), (2004) 27-34.
4) Hoshiai, T., Yamane, Y., and Tsuda H.: Category-level Retrieval among Heterogeneous Information Sources based on Category Matching, in Proceedings of the 6th SANKEN (ISIR) International Symposium, Osaka (2003) 73-76.

Using PROMPT Ontology-Comparison Tools in the EON Ontology Alignment Contest

Natalya F. Noy, Mark A. Musen

Stanford Medical Informatics, Stanford University,
251 Campus Drive, x-215, Stanford, CA 94305, USA
{noy, musen}@smi.stanford.edu

Abstract. Objective evaluation and comparison of knowledge-based tools has so far been mostly an elusive goal for researchers and developers. Objective experiments are difficult to perform and require substantial resources. The EON Ontology Alignment Contest attempts to overcome these problems in inviting tool developers to perform a series of experiments in ontology alignment and compare their results to the reference alignments produced by experiment authors. We used our PROMPT suite of tools in the experiment. We briefly describe PROMPT in the paper and present our results. Based on this experience, we share our thoughts on the experiment design, its positive and negative aspects, and talk about lessons learned and ideas for future such experiments and contests.

1 Introduction

Objective evaluation and comparison of knowledge-based tools has so far been mostly an elusive goal for researchers and developers. First, such evaluations require resources, both financial and human. Second, for an evaluation to be objective, it must be designed and run by someone other than tool developers themselves. Otherwise, the evaluation setup and comparison parameters are inevitably skewed (often subconsciously) to benefit the designers' own tools. Third, it is often hard to come up with realistic tasks and gold standards because all tools are designed for somewhat different purposes and trying to pigeonhole the tools into a single set of tasks for an evaluation often puts the tool designers in the untenable position of comparing their tool in circumstances for which it was not designed. Fourth, many of the criteria are, by their very nature subjective. For example, when evaluating the quality of an ontology, we often don't have a single correct answer for how certain concepts should be represented. When evaluating the quality of ontology alignment, we often cannot agree on the precise relationships between concepts in source ontologies.

Therefore, any experiment or contest to compare ontology-based tools will almost inevitably draw criticism. Nonetheless, given the extreme dearth of any comparative evaluation of ontology-based tools, any good evaluation is a significant step forward. Hence, we believe that the community will learn many lessons from the EON Ontology Alignment Contest[1] that is run as part of the 3d Workshop on the Evaluation of

[1] http://co4.inrialpes.fr/align/Contest/

Ontology-based Tools at the International Semantic Web Conference.[2] In the experiment, developers of ontology-alignment tools used their tools to compare concepts in a set of ontologies. The resulting alignment was then compared to the reference alignment provided by the experiment authors. Hopefully, the lessons from this experiment will enable the community to produce new experiments that do not suffer from some of the problems the EON experiment has. Indeed, in some of the materials it is called a "contest" and in others, an "experiment" that will help us to understand how to run such evaluations. We subscribe to the latter goal since the experiment design, being the first one of its kind, has considerable deficiencies that make a contest premature.

In the rest of this paper, we share our thoughts on the experiment design, its positive and negative aspects, describe the set of PROMPT tools that we used in the experiment and our results. We then talk about lessons learned and ideas for future such experiments and contests.

2 Ontology Comparison with PROMPT

PROMPT is a suite of tools for managing multiple ontologies. It is a plugin to the Protégé ontology-editing and knowledge-acquisition environment.[3] The open architecture of Protégé allows developers to extend it easily with plugins for specific tasks. We implemented PROMPT as a set of such plugins.

2.1 Components of the PROMPT Suite

The PROMPT suite includes tools for many of the tasks in multiple-ontology management: interactive ontology merging [1], graph-based mapping [3], creating views of an ontology [2], ontology versioning [4], and ontology-library maintenance. It is through development of these tools that we came to realize that many of these directions are indeed related and started integrating the approaches into a common framework. The tools in the PROMPT suite share user-interface components, internal data structures, some of the algorithms, logging facilities, and so on.

IPROMPT is an interactive ontology-merging tool. It leads users through the ontology-merging process, suggesting what should be merged, identifying inconsistencies and potential problems and suggesting strategies to resolve them. IPROMPT uses the structure of concepts in an ontology and relations among them as well as the information it gets from user's actions. For example, if IPROMPT's analysis identified that two classes from different ontologies may be similar and then the user merged some of their respective subclasses, IPROMPT will be even more certain that those classes are similar.

ANCHORPROMPT—another component in the suite—analyzes a graph representing ontologies on a larger scale, producing additional suggestions. It takes as input a set of pairs of matching terms in the source ontologies and produces new pairs of matching terms. ANCHORPROMPT's results could then be used in IPROMPT to present new suggestions to the user.

[2] http://km.aifb.uni-karlsruhe.de/ws/eon2004/
[3] http://protege.stanford.edu

Merging source ontologies to create a new one is not always what the user needs. If the user prefers to keep the source ontologies separately and to record only the alignment, iPROMPT saves the alignment as a side-effect of the merging process. The user can then discard the merged ontology and simply use the produced alignment.

PROMPTDIFF is a tool for comparing ontology versions. We observed that many of the heuristics we used in iPROMPT would be very useful in finding what has changed from one version of an ontology to another. These heuristics include analysis and comparison of concept names, properties that are attached to concepts, domains and ranges of properties and so on. At the same time, our level of confidence in the analysis could be much higher than in the case of ontology merging: If two concepts that came from versions of the same ontology look the same (e.g., have the same name and type), they probably *are* the same, whereas if two frames that came from independently developed ontologies look the same, they may or may not be the same. Consider a class `University` for example. In two different ontologies, this class may represent either a university campus, or a university as an organization, with its departments, faculty, and so on. If we encounter a class `University` in two versions of the same ontology, it is much more likely that it represents exactly the same concept.

The PROMPTDIFF algorithm for version comparison consists of two parts: (1) an extensible set of heuristic matchers and (2) a fixed-point algorithm to combine the results of the matchers to produce a structural diff between two versions. Each matcher employs a small number of *structural* and *lexical* properties of the ontologies to produce matches. The fixed-point step invokes the matchers repeatedly, feeding the results of one matcher into the others, until they produce no more changes in the diff.

One matcher, for example, looks for unmatched classes where all siblings of the class have been matched. If multiple siblings are unmatched, but their sets of properties differ, another matcher will pick up this case and try to match these classes to unmatched subclasses of the parent's image. Another matcher looks for unmatched properties of a class when all other properties of that class have been matched. There are matchers that look for lexical properties such as all unmatched siblings of a class acquiring the same suffix or prefix. The architecture is easily extensible to add new matchers. With the introduction of OWL, for example, we implemented additional matchers that compared anonymous classes.

2.2 Evaluating PROMPT in the EON Ontology-Alignment Contest

In the EON Ontology-Alignment Contest, the participants had to perform the following task: For each of the ontologies in the experiment, map its classes and properties to the classes and properties in a **reference ontology** and record the alignment. The alignment was then compared to the **reference alignment** provided by the authors of the experiment.

We originally planned to use iPROMPT and ANCHORPROMPT—our tools for ontology merging and alignment—in the experiment. However, most of the source ontologies in the experiment are not independently developed ontologies, but rather versions of the same ontology. Even in the three experiments where the reference ontology developed by the experiment authors was compared to ontologies developed elsewhere,

the comparison closely resembled comparison between ontology versions: All ontologies represented the structure of BibTex references and therefore were at the same time limited and driven by the BibTex data model and hence did not vary significantly in the concepts they represented. They varied only in coverage: some ontologies included for instance more detailed representation of people and projects but since these concepts were not part of the BibTex model, they were not mapped to concepts in the reference ontology anyway.

Given that the comparison was very close to version comparison, we decided to use only PROMPTDIFF, our version-comparison tool. PROMPTDIFF produces a mapping between concepts in one version and their corresponding concepts in another version. In addition, PROMPTDIFF produces a comparison between these corresponding concepts, presents the results to the user in an intuitive interface and enables users to accept and reject changes. For the experiment, we used only the first component of PROMPTDIFF, the one that produces correspondences between concepts (classes and properties) in two versions.

Adaptation of PROMPTDIFF **for the Experiment.** Figure 1 shows the process that we used to perform comparisons for the experiment. First, we ran PROMPTDIFF on the two ontologies to be mapped. PROMPTDIFF runs in a stand-alone completely automatic mode and produces a table of mappings between concepts (clases, properties, and instances) as one of its results. Second, we recorded the results in an RDF file following the format set in the experiment. The recording mechanism was our first modification of the tool for the experiment. Third, we used PROMPTDIFF *itself* again to compare the alignment to the benchmark alignment provided in the experiment and to compile the results. Because PROMPTDIFF compares not only classes and properties, but also individuals (by comparing their values for corresponding properties), we could easily perform this comparison of RDF data automatically. We present our comparison results in Section 3.

We have performed minimal adaptation of the tool itself. One of the key features of PROMPTDIFF is that it is an *extensible* collection of matchers. Hence, it is easy to add new matchers. In the process of the experiment, we found several interesting heuristics that we have added to the tool. These heuristics were conservative enough that they did not lower the resulting precision for all the tests, but they did help us increase the recall. For instance, we already had a heuristic that matched two properties P_1 and P_2, each of which had a single range, R_1 and R_2 respectively, if we already new that R_1 and R_2 matched. We did not have a corresponding heuristic for domains, however, and therefore added it. We found these heuristics to be generally useful for version comparison and plan to keep them in the tool. Overall, we added three new matchers to the 17 matchers that we already had.

Other Considerations. We would like to point out several other features of PROMPTDIFF that are relevant for the experiment.

As a plugin to Protégé which, through its extensible knowledge model and architecture, supports many different backends and ontology formats, PROMPTDIFF works with ontologies in OWL, RDFS, OKBC, and other languages. Therefore, we did not need to

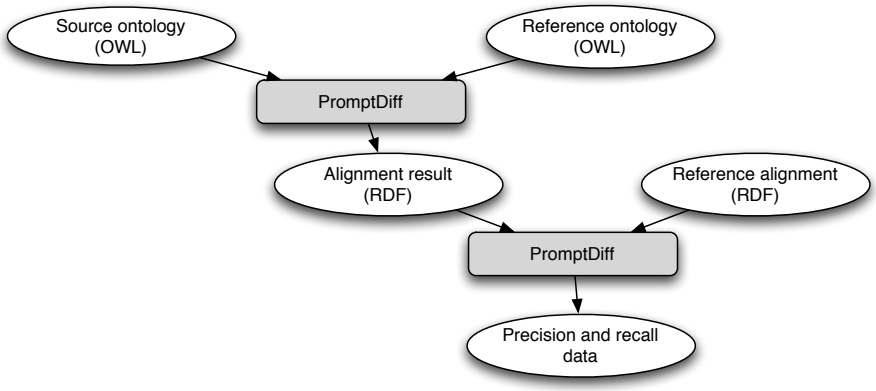


Fig. 1. Using PROMPTDIFF in the EON Ontology-alignment Contest. First, PROMPTDIFF automatically compares the source ontology to the reference ontology. Second, it produces the alignment results in the RDF Format specified in the contest rules. Third, it automatically compares the alignment result to the reference alignment to produce recall and precision data.

adapt it in any way either to compare the ontologies in the experiment (which were in OWL) or to compare the alignments that PROMPTDIFF produced to the reference alignment (which were in RDF).

PROMPTDIFF and other tools in the PROMPT suite produce only equivalence mappings. Some of the reference alignments in the experiment contained generalization and specialization mappings. In our results, we considered only the equivalence mappings from the reference alignment. In cases where PROMPTDIFF produced an equivalent mapping and the reference alignment contained only generalization or specialization, we considered it a positive result for PROMPTDIFF. In almost all of these cases, we believed that, from the common-sense point of view, the equivalence alignment was actually more correct (Section 3).

PROMPTDIFF does not use comments or instance information to align classes or properties (It uses instance information only to align between instances themselves). Therefore, tests that differed only in the presence of comments produced identical results.

3 Experiment Results

We performed 20 tests as part of the experiment, all the tests specified in the experiment. Table 1 contains the summary of the results. Note that in the discussion below we focus almost exclusively on recall. Because PROMPTDIFF uses a very conservative approach in creating matches, matches that it finds are almost always correct. Our previous experiments have put precision at 100%. in this experiment, it was above 99%.

3.1 Results of Specific Tests

In 10 of the 20 tests, the source ontology contained classes and properties with exactly the same names as those in the reference ontology. While in a general case of ontology alignment, when two classes from different ontologies have the same name, they may not necessarily match, in version alignment, they almost always do. In fact, in the experiment, they always matched. Furthermore, in these 10 cases, there were no other matches. Thus, any tool, which, just as PROMPTDIFF starts by comparing names of concepts (and, perhaps their types) would trivially produces the perfect alignment in these cases. These tests are: 101, 103, 104, 221, 222, 223, 224, 225, 228, 230. Test 102 had no matches, and PROMPTDIFF correctly found this result. We do not discuss these tests further in this section.

Tests 201 and 202: No names, no comments. In these tests, each class and property name was replaced by a random one. These tests were the most difficult of all tests because, except for the imported classes $foaf : Organisation$ and $foaf : Person$, there was no other name information for the tool to use. PROMPTDIFF was able to match 8 classes and 3 properties. Having the two imported classes was crucial to come up with any matches, since PROMPTDIFF could then use such clues as having single unmatched subclasses of a matched class; having a single property attached to matched classes; and so on.

We performed a variation of this experiment, where the class names all remained scrambled, but property names were all restored (tests 201a and 202a in Table 1). Having properties helped tremendously since now PROMPTDIFF was able to find correctly 90 of the 92 matches from the reference alignment. The only classes that it did not match were $MastersThesis$ and $PhDThesis$. In fact, after examining these classes in the ontology, we are convinced that these two classes are indistinguishable to any tool in the experiment: They are referenced in exactly identical ways in the reference ontology and having their names scrambled makes them indistinguishable.

Since PROMPTDIFF does not use comments in its analysis, results for the tests 201 and 202 (which differed only in the presence of comments) were identical.

Test 204: Naming conventions. Despite the use of different naming conventions in the source ontology, PROMPTDIFF found all matches.

Tests 205 and 206: Synonyms and Foreign Names. As the Table 1 shows, PROMPTDIFF matched approximately 50% of classes in each of these tests, and approximately 25% of properties. PROMPTDIFF does not have a specific matcher that uses a dictionary to look up word translations or a thesaurus to look up synonyms. Hence, it was left to using only structural clues to find the matches. Without the knowledge of synonyms, however, many of the classes and properties were hard or impossible to distinguish. Some of the classes that it did not match include $MastersThesis$ and $PhDThesis$ mentioned earlier, and $School$ and $Publisher$, which are structurally indistinguishable without instances (which PROMPTDIFF does not use to compare classes). Other classes could probably be distinguished structurally if we used less conservative heuristics, but this approach would have produced many false matches in other experiments. Clearly, tools

that use dictionaries and thesauri would perform better on these tests. PROMPTDIFF itself would also benefit from additional matchers that use these external sources.

Test 301, 302, 303: BibTex/MIT, BibTex/UMBC, BibTex/Karlsruhe. The comparisons in this series were the only ones involving ontologies developed at different institutions and, hence, were not true *version* comparisons. As we pointed out earlier, the ontologies were still sufficiently close and used a very similar terminology, and hence using PROMPTDIFF was still appropriate. However, this case was the first one where PROMPTDIFF produced some incorrect matches (PROMPTDIFF incorrectly matched the $Conference$ classes in the two ontologies, and two $address$ properties).

In the reference alignments for these tests, we believed that some of the equivalence matches were not correct. In test 301, the matches between properties $date$ and $hasYear$ and $book$ and $booktitle$ were not supported by the ontologies. There was a similar problem with the $book$–$booktitle$ match in the reference alignment for test 302. In tests 302 and 304, the reference alignment suggests that class $Chapter$ in the reference ontology matches class $InBook$ in the source ontology and the class $InBook$ in the reference ontology is actually a specialization of the class $InBook$ in the source ontology. We did not find any data in the ontologies to support this match and therefore considered the equivalence match between $Chapter$ and $InBook$ in the reference alignment to be incorrect and the equivalence match between the two classes $InBook$ to be correct. It was also unclear why the properties $proceedings$, $isPartOf$ and $booktitle$ match. We did not include these incorrect matches from the reference alignment in our results. We also did not consider matches to concepts in non-local namespaces and therefore discarded the match to the $dateTime$ XML Schema datatype from the reference alignment for test 302.

3.2 Overall Results

Table 1 presents a detailed look at the results of the experiments. We separate the results for classes and properties before finding the average value. We have included our modified tests 201a and 202a into consideration: In these tests, we kept the class names scrambled, but restored the property names. We show the average values for the original set of tests (the line marked "no mods"). The line marked "mods" shows the average values if we consider modified versions of tests 201 and 202 (with property names intact) instead of the original ones. Given that the recall number for tests 201 and 202 were clearly outliers (recall that because PROMPTDIFF does not look at comments, these two tests were identical), we also looked at the average values without these outliers (the last line in the table).

The average precision for both class and property matches was above 99%. The average recall is 62.4%. Note however, that his low recall figure is caused mainly by two tests, 201 and 202. If we replace these two tests with our modified tests, 201a and 202a (the "mods" line), the recall goes up to 94.1%. If we drop these two outlier tests altogether, the recall is 94.9%. In all cases, where there was any difference in recall between classes and properties, the recall for classes was higher than the recall for properties.

Test	Class matches in PROMPT	Property matches in PROMPT	Precision for class matches (%)	Precision for property matches (%)	Overall PROMPT precision (%)	Recall for class matches (%)	Recall for property matches (%)	Overall PROMPT recall (%)
101	33	59	100	100	100	100	100	100
102	0	0	100	100	100	100	100	100
103	33	59	100	100	100	100	100	100
104	33	59	100	100	100	100	100	100
201	8	3	100	100	100	24.2	5.1	14.7
201a	31	59	100	100	100	93.9	100.0	97.0
202	8	3	100	100	100	24.2	5.1	14.7
202a	31	59	100	100	100	93.9	100	97.0
204	3	59	100	100	100	100	100	100
205	20	21	100	100	100	60.6	35.6	48.1
206	23	21	100	100	100	69.7	35.6	52.6
221	33	59	100	100	100	100	100	100
222	33	59	100	100	100	100	100	100
223	33	59	100	100	100	100	100	100
224	33	59	100	100	100	100	100	100
225	33	59	100	100	100	100	100	100
228	33	59	100	100	100	100	100	100
230	33	59	100	100	100	100	100	100
301	14	15	93.3	100	96.7	93.3	41.7	67.5
302	12	19	100	90.5	95.2	100	86.4	93.2
303	16	28	100	93.3	96.7	88.9	100.0	94.4
304	30	46	100	100	100	100	100	100
no mods			99.8	99.6	**99.7**	67.6	57.3	**62.4**
mods			99.9	99.7	**99.8**	94.5	93.8	**94.1**
no outliers			99.8	99.4	**99.6**	96.9	92.8	**94.9**

Table 1. Experiment results for PROMPTDIFF. Test numbers correspond to the tests in the experiment description. Tests 201a and 202a are the same as tests 201 and 202 respectively, but with property names retained and only class names scrambled. The first line in the results ("no mods") represents the average result for all the original tests in the set; the line for "mods" result represent the average for all the tests when tests 201a and 202a (our own test modifications) are considered instead of 201 and 202; results for "no outliers" represent the average after dropping the two (identical from PROMPT's point of view) worst tests, 201 and 202.

3.3 Discussion of PROMPT Results

Generally, we were very satisfied with the performance results. The precision was almost perfect, averaging above 99.5%. In general, if there were some matches between class or property names, there was enough information in the structure of the ontology to find the rest of the matches.

PROMPT did not perform as well with finding all property matches. Indeed, after examining the ontologies, we found that many properties were hard to distinguish purely by the structure of the ontology. Many properties had the same domains and ranges, there were very few restrictions to differentiate them, and so on. We felt that any heuristics that would have allowed us to find these property matches, would have provided false positives in other tests, improving recall but lowering the precision.

In summary, PROMPTDIFF works very well, both in terms recall and precision, if at least some of the class and property names match. It is notable that it was able to find 25% of the class matches even when all property and class names were scrambled.

As we noted earlier, PROMPTDIFF provides only equivalence matches, and therefore if generalization or specialization matches are required, this tool will not be appropriate.

There is some information that PROMPTDIFF does not use but that could provide additional matches. We plan to consider using this information in the future, such as using dictionaries for comparing ontologies in different languages; using thesauri for synonym lookup; using instance information and property values to match classes and properties.

4 Discussion of the Experiment

Perhaps the most important result of the experiment are the insights that we gained in how these experiments can be conducted in the future. Many things about this experiment were very positive:

- It was a controlled experiment, with test cases and reference results published well in advance and in computable form.
- The ontologies were of reasonable size.
- Some of the alignments were to real ontologies produced by different groups independent from one another (although they all were modeled from the BibTex data model, which made this independence slightly less valuable than it could have been).

Many things can be improved and must be considered for future experiments, however.

Use ontologies developed independently. As we have noted earlier, this experiment was really about comparing ontology *versions* rather than about comparing independently developed ontologies. This experiment, while a perfect experiment for comparing ontology-versioning tools, does not provide any data on how well the same tools

will perform in the semantic-integration scenario when they must align truly independent ontologies. A more interesting and challenging experiment would be to use ontologies that are truly independent from one another—developed by different groups and in the domain that is less "pre-modeled" than BibTex references.

Provide consensus alignments. The more independent source ontologies are the more likely it is that alignments themselves would be a point of disagreement. Even in this experiment, we disagreed with some of the reference alignments provided by the experiment authors. Some of the alignments were completely unsubstantiated by any information in the ontologies or even by common sense. In the future, a panel of experts from different groups (ideally, the authors of the ontologies themselves) should produce a consensus alignment reference to minimize disagreement.

Include interactive tools. Many researchers working in the area of ontology alignment and mapping firmly believe that a fully automatic ontology alignment is not possible. Therefore, many tools include an interactive element in them. The current experiment assumed a fully automatic alignment. For more realistic ontologies and usage scenarios, we have to incorporate the interactive component as well. It is challenging to introduce such component in a controlled way. One possibility would be to specify the set of user-provided alignments as input. This approach would simulate tools that allow users to provide alignment information to the tool. It may be however that we can come up with a controlled test for interactive tools only after we perform a pilot experiment similar to the current one to understand better what types of inputs the tools require and at which stages.

Freeze systems before the test set is published. The results of the current experiment will inevitably be skewed because the tool developers had the opportunity to adjust their tools after the test data were published. This situation opens up a possibility that developers will produce versions of the tools optimized for this particular task and set of ontologies, but not for another set of ontologies and tasks. In our case, as noted earlier, we have added three new matchers to PROMPTDIFF after analyzing the ontologies. These matchers are not specific to the test, they are simply heuristics we have not thought of before and will remain part of the tool. However, to make the results completely fair, future experiments (and, even more so, "contests") should require that a version of the tool is frozen before the test data are published. We can follow the traditional TREC model where training data are published first, tools fine-tuned and instrumented to collect all the statistics, and then frozen before the test data are made available. Then the period to perform the tests can be very short (2 weeks or so) since no tool instrumentation will need to be done at this point.

Provide more automated tools to collect and compare results. One of the great positive points about the current experiment was the requirement to use the standard format to record the alignments. There is no reason (except time and effort, which should not be significant) not to go further and provide a suite of tools that will perform the analysis on the alignment files, producing the same statistics and in the same format for all the participants.

Require that tools be made available to the community. Ideally, the tools participating in the experiment should be available to the community under some open-source or free-use license. In this case, others will be able to reproduce the results and to use the tools on their own ontologies. If making tools available for free is not feasible, they must at least be available to the experiment organizers so that they can verify the results. We believe that this availability to the organizers should be a hard requirement for anyone participating in the experiment.

Finally, we believe that such experiments are extremely important in not only comparing different semantic-integration tools, but also in improving these tools and their performance.

Acknowledgments

This work was conducted using the Protégé resource, which is supported by grant LM007885 from the United States National Library of Medicine. This work was supported in part by a contract from the US National Cancer Institute. The Prompt suite is available as a plugin to the Protégé ontology-development environment at `http://protege.stanford.edu/plugins/prompt/prompt.html` The results of the experiment are available at `http://protege.stanford.edu/plugins/prompt/oacontest`.

References

1. N. Noy and M. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX, 2000.
2. N. Noy and M. A. Musen. Specifying ontology views by traversal. In *3rd International Semantic Web Conference (ISWC2004)*, Hiroshima:Japan, 2004.
3. N. F. Noy and M. A. Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, WA, 2001.
4. N. F. Noy and M. A. Musen. PromptDiff: A fixed-point algorithm for comparing ontology versions. In *Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, Edmonton, Alberta, 2002.