Vol-87

EON2003 Evaluation of Ontology-based Tools

Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools held at the 2nd International Semantic Web Conference ISWC 2003
20th October 2003 (Workshop day)
Sundial Resort, Sanibel Island, Florida, USA

Edited by

York Sure (Contact Person) #
Oscar Corcho +

# Institute AIFB, University of Karlsruhe, Postfach, 76128 Karlsruhe, Germany
+ Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Madrid, Spain

Note: Additional information and material such as slides of the presentations and created experiment ontologies can be found at the EON2003 workshop website.

Table of Contents

Evaluation experiment of ontology tools' interoperability with the WebODE ontology engineering workbench
Óscar Corcho, Asunción Gómez-Pérez, Danilo José Guerrero-Rodríguez, David Pérez-Rey, Alberto Ruiz-Cristina, Teresa Sastre-Toral, M. Carmen Suárez-Figueroa

Case Study: Using Protege to Convert the Travel Ontology to UML and OWL
Holger Knublauch

Interoperability of Protege 2.0 beta and OilEd 3.5 in the Domain Knowledge of Osteoporosis
Franz Calvo and John Gennari

---

submitted by York Sure, 2003-11-04

Towards a benchmark for Semantic Web reasoners - an analysis of the DAML ontology library

Christoph Tempich and Raphael Volz

Institute AIFB, University of Karlsruhe, Germany
`http://www.aifb.uni-karlsruhe.de`
`(tempich,volz)@aifb.uni-karlsruhe.de`

1 Introduction

Benchmarks are one important aspect of performance evaluation. This paper concentrates on the development of a representative benchmark for Semantic Web-type[1] ontologies. To this extent we perform a statistical analysis of available Semantic Web ontologies, in our case the DAML ontology library, and derive parameters that can be used for the generation of synthetic ontologies. These synthetic ontologies can be used as workloads in benchmarks.

Naturally, performance evaluation can also be performed using a real workload, viz. a workload that is observed on a reasoner being used for normal operations. However, such workloads can usually not be applied repeatedly in a controlled manner.

Therefore synthetic workloads are typically used in performance evaluations. Synthetic workloads should be a representation or model of the real workload. Hence, it is necessary to measure and characterize the workload on existing reasoners to produce meaningful synthetic workloads.

This should allow us to systematically evaluate different reasoners and reasoning techniques using a benchmark to gain realistic practical comparisons of individual systems.

1.1 Related Work

The development of benchmarks for ontology-based systems is substantially different from the development of a test suite [3] for testing the correctness or ability of a reasoner in handling particular primitives of an ontology language. The latter is intended to give yes or no answers to questions like whether a system can make certain entailments or find particular inconsistencies. The former, however, is intended to come up with numbers for a set of performance criteria (metrics).

Within the Description Logic community benchmarking [8, 6] was performed repeatedly in the past for empirical system comparsion. However, these representativeness of the used benchmarkss [1, 7, 6] are questionable for practical cases

[1] Hence RDFS, DAML+OIL and OWL

due to several reasons. For example, [8] tested the performance of class satisfiability based on a sequence of classes which are (exponentially) increasingly difficult to compute. These class definitions are hardly representative for practical cases. The test for ABox reasoning was underdeveloped since most systems at the time of evaluation did not support any ABox reasoning capabilities.

[6] used both real and synthetically generated knowledge bases as one part of their evaluation of knowledge representation systems. The study was only concerned with the terminological part of knowledge representation systems and used a target representation language of limited expressivity for generating synthetic knowledge bases. The generated knowledge bases, however, are not realistic for Semantic Web-type knowledge bases as we will see from our analysis. Hence, their assumption for class formation [2] is not representative.

1.2 Contribution

In this paper, we provide a systematic approach for the creation of benchmarks for knowledge representation systems. The key characteristic of our approach is that we want to use generating functions to create synthetic ontologies ar, which are derived from structural properties of a given (representative) set of ontologies. If the set of analyzed ontologies is structurally inhomogeneous, clustering techniques are applied to come up with k homogeneous subsets, viz. types of ontologies, for which separate synthetic ontologies can be created. A particular benchmark then consists of several synthetic ontologies representing individual types. Instead of reducing language expressivity to the least common denominator (RDFS in the Semantic Web case), we consider the inability of a particular reasoner to support certain language primitives in our performance evaluation design.

1.3 Limitations

While our approach (with proper adaptation) might be reusable to evaluate other tools relevant to KR-based applications, e.g. editors and visualization tools, our primary focus is set on evaluation of the inference and data processing core of knowledge representation systems. Additionally, we do not consider the identification of a representative list of service requests, which nevertheless are an important aspect in a benchmark. The actual generation of synthetic ontologies is subject of our ongoing research, nevertheless the initial results of our analysis appear to be promising and worth to disseminate.

1.4 Structure of the paper

The paper is structured as follows. Section 2 describes our approach to performance evaluations using benchmarks. Section 3 presents our analysis of the DAML ontology collection, which motivates the necessity for a categorization into several types of ontologies. Section 4 describes the clustering and shows

[2] each class definition is a conjunction containing one or two class symbols (superclasses) zero or one cardinality restriction and zero, one or two value restrictions

how we come up with three categories of ontologies, which are more homogenous. We conclude in Section 5 summarizing our results and giving an outlook to ongoing and future work.

2 Performance Comparisons

We consider benchmarking as the process of performance comparison of two or more reasoners by measurements. A benchmark is the workload used in such measurements. Each performance comparison draws itself on a set of performance criteria or metrics. The choice of the metrics directly depends on the list of services offered by the reasoner.

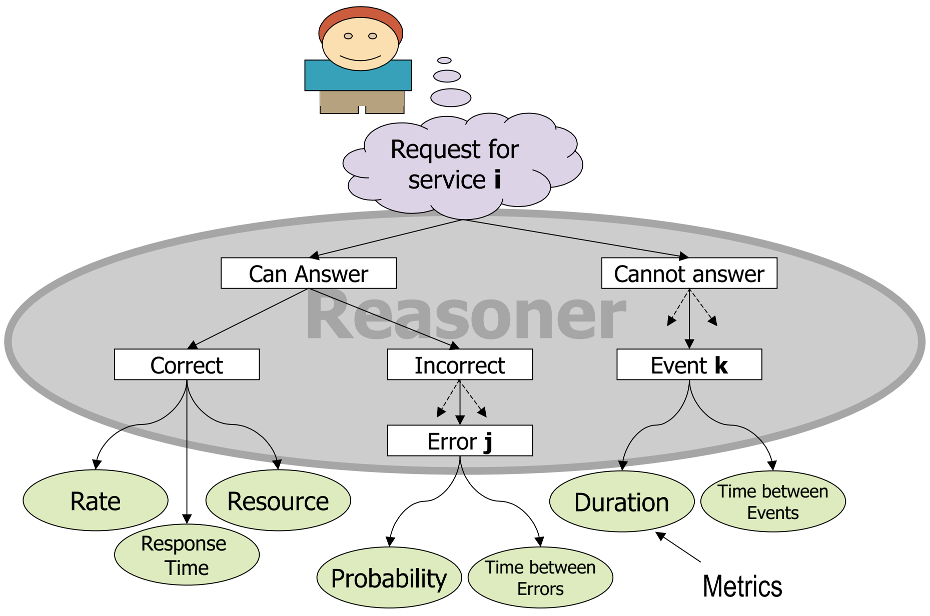


Fig. 1. Services and Metrics in Benchmarks

2.1 Reasoning Services

For each service request several possible outcomes exist (cf. Figure 1). Generally, we can assume that a particular system can either respond correctly, incorrectly or cannot answer the request. A reasoner usually offers query services to interface with the system, several systems also allow update services for manipulation of the knowledge base.

Unlike databases, a reasoner supporting DAML+OIL or OWL, will usually offer several different query services w.r.t. an ontology O. These query services primarily target queries about classes:

1. class-instance membership queries: given a class C,
   (a) ground: determine whether a given individual a is an instance of C;

   (b) open: determine all the individuals in O that are instances of C;
   (c) "all-classes": given an individual a, determine all the (named) classes in O that a is an instance of;
2. class subsumption queries: i.e., given classes C and D, determine if C is a subclass of D w.r.t. O;
3. class hierarchy queries: i.e., given a class C return all/most-specific (named) superclasses of C in the T-Box and/or all/most-general (named) subclasses of C in the T-Box;
4. class satisfiability queries, i.e., given a class C, determine if the definition of C is generally satisfiable (consistent).

There are similar queries about properties, viz. property-instance membership, property subsumption, property hierarchy, and property satisfiability, and also the possibility to check the consistency of the whole ontology / knowledge base.

A single service does not suffice One might want to argue that it is sufficient to measure the performance of the satisfiability, since it is well known, that all queries about classes can be reduced to satisfiability testing. It is important, however, to distinguish different types services, since optimizations can be made for particular services. Naturally the effect of those optimization should be measurable. For example, we might want to measure the performance of a classification service, which can be reduced to several class-subsumption queries (and in turn satisfiability), but reasoners may use different classification algorithms to minimize the number of issued subsumption queries.

2.2 Metrics

For each of the different service requests and their corresponding responses, we can observe a number of metrics. These metrics are later evaluated in the comparison of systems. We may measure successful performance by time-throughput-resource metrics, which measure the responsiveness and productivity and utilization (of system resources) of the reasoner. Notably it is not sufficient to consider response time as the only metric. Some reasoners may be able to respond to requests in parallel, which might lead to a higher throughput. Another reasoner may have a small memory footprint and therefore have a better utilization of system resources. Of course, individual evaluations might consider further metrics.

If the response is incorrect, errors should be returned by the reasoner. Such errors can be classified and it is interesting to determine probabilities for each class of errors and measure the time between such errors. Notably, it is not sufficient to only measure correct performance, since errors are common [6] (even if the reasoning procedures are supposed to be sound and complete).

Several reasons may exist that a reasoner fails to provide an answer at all. Similar to errors, it is sensible to classify failures and determine the probabilities and time between failures for each class. For example, a reasoner may be unavailable due to network errors or software errors or due to lack of support for certain language primitives.

2.3 Workloads

The workload of a reasoner consists of the knowledge base which is loaded by the reasoner and the list of service requests issued by users. We do not consider the identification of a representative list of service requests, which are an important aspect in a benchmark, but concentrate on creating a representative synthetic knowledge base that is subject to user queries.

3 Characterizing Semantic Web ontologies

In order to generate sensible synthetic ontologies, an analysis of available ontologies is necessary, this is the subject of this section.

3.1 Selecting a list of Semantic Web ontologies

For our experiment we chose the DAML.org list of ontologies [4], which contained 247 ontologies at the time of the analysis. Our selection of this set of ontologies is intentional and motivated by the following facts: Firstly, we are not related with the authors of the ontologies in any form. Secondly, most ontologies are created by different people with diverse technical backgrounds (ranging from students to researchers). In this sense, they can be understood as representative for the Semantic Web. Interestingly, many of the ontologies in the library turn out to be just conversions of ontologies, which were initially created in some other representation language. For example, the famous wine ontology is with us since 'Classic' times (for more than 15 years !). This also seems to be a valid assumption for the Semantic Web, which is for sure not created from scratch.

We processed these ontologies using the 1.6.1 version of JENA, we used the Jena DAML API to access the data. The collection contained 189 ontologies in DAML-ONT or DAML+OIL formats, which should be processable by the API in general.

Unfortunately more than 50% of the ontologies contained RDF errors or did not contain valid URIs or did not use RDF(S) namespaces correctly. We did not make any attempt to fix these problems, therefore we could only process 95 ontologies in practise. This results seems shocking, but underlines the need of software that can cope with such errors[3].

The correctness of namespace[4] and RDF usage is, however, not the only relevant property. For example, 21% of the parsable ontologies did not specify the type of properties, viz. whether they are in fact Datatype- or ObjectProperties. In practise, further heuristics need to be applied to make use of these ontologies in reasoners, i.e. deriving the missing type information (e.g. such as done in [2]). Again, we did not make any attempt to fix these problems.

[3] Analogous to HTML browsers, which can cope with all sorts of HTML errors !

[4] A good example for namespace confusion is the NASDAQ ontology (http://www.daml.org/ontologies/342). **Quiz question:** can you spot the inconsistency ?

We based our analysis on the structural properties of asserted information, hence no reasoning was applied. Detailed numbers and sources for the analysis package can be found online[5].

	Average	Std Dev	Median	Min	Max	C.O.V.
Primitive Classes	154,29	1.016,07	5	1	9.795	6,59
Class Expressions	175,20	1.016,39	19	1	9.795	5,80
Restrictions	19,13	44,52	7	-	327	2,33
Enumeration	0,33	1,72	-	-	16	5,26
Set Operation	1,45	8,62	-	-	78,00	5,94
Properties	28,41	43,47	13	-	269	1,53
Object Properties	8,34	31,26	2	-	269	3,75
Datatype Properties	12,57	23,15	4	-	145	1,84
Individuals[6]	29,48	222,32	-	-	2.157	7,54
EquivalentClass	0,73	3,15	-	-	20,00	4,34

Table 1. Average Usage of some language primitives (across all ontologies)

3.2 Average Characterizations

One part of our analysis was concerned with simply counting the usage of certain features. Table 1 summarizes the average usage of language primitives in the ontologies. One important aspect of the summary given in table 1 is that the coefficient of variation (C.O.V.), viz. the ratio of standard deviation and the mean, is high. This shows that the particular ontologies vary tremendously, that is the distribution is highly skewed, hence the median is a more representative characterization of the different numbers than the average.

As we can see primitive classes[7] are the predominant form of class expressions. Different sorts of restrictions are the second most important form of class expressions, interestingly only one ontology actually made use of a single cardinality restriction that would not be expressible in OWL Lite. Seldom enumerations are used, even considering hasValue[8]. Actually, only seven ontologies used set operations to define classes, which are also not available in OWL Lite[9]. Another interesting aspect is that equality is rarely used to define classes (and also rarely used to make properties and individuals synonymous). Also ontologies typically do not contain any individuals. We assume that the pool of individuals will be distributed through the web and is consequently rarely specified together with the ontology.

3.3 Ratios of Primitives

The second part of our analysis concerned the ratio of different primitives in ontologies. The variability of these ratios is smaller than the average counts (cf.

[5] http://kaon.semanticweb.org/owl/evaluation/

[7] By primitive class we denote the atomic named classes that occur on the left-hand sides of subClassOf statements

[8] which can be understood as a syntactically convenient form to express value restrictions with a nominal value

[9] viz. complementOf, disjointUnionOf or unionOf

Ratios	Average	Std Dev	Median	Min	Max	C.O.V.
Primitive/Class Expr.	50%	0,34	39%	6%	100%	0,67
Obj. Prop. / Prop.	24%	0,27	16%	0%	100%	1,12
Dat. Prop. / Prop .	52%	0,38	67%	0%	100%	0,72
Prop. / Prim. Class	3,54	3,89	2,00	-	21,00	1,10
Trans. Prop. / Prop.	0,05	0,10	-	-	0,40	2,30
Obj. Prop./ Prim. Class	0,61	0,83	0,50	-	5,57	1,35
Dat. Prop./ Prim. Class	2,25	3,33	1,00	-	15,75	1,48
Ex. Rest. / Rest.	1%	0,08	0%	0%	65%	5,79
Univ. Rest. / Rest.	48%	0,44	52%	0%	100%	0,91
Card. Rest./ Rest.	34%	0,38	20%	0%	100%	1,11
Rest./Primitive	2,32	2,70	1,50	-	16,00	1,17
Asserted Ind. / Primitive	0,60	4,18	-	-	40,50	6,97

Table 2. Ratio between some language primitives (across all ontologies)

Table 2) since we aggregated relativized numbers. Another effect is of course that numbers do not necessarily add up anymore.

One aspect that can be observed is that the DAML.org library typically contains ontologies and not schemas, since the ratio between Data Properties and Primitive Classes is very low. However, datatype properties are the predominant type of properties. Also some of the ontologies, particularly those with high numbers of classes do not contain any properties, hence the average number of properties per primitive class is very low.

Some 5% of the defined object properties were declared to be transitive. None of the analyzed ontologies contained any functional or inverse functional properties.

Among the restrictions, universal restrictions are predominant, in fact they almost half of all restrictions on average. Typically, a primitive class is further defined by more than two restrictions. Not surprisingly, if we recall our argument for the low number of individuals, at least half of all primitive classes have no direct asserted individuals.

3.4 Distributions of elements in class definitions

The third part of analysis was concerned with determining distributions of the elements contained in class definitions, viz. trying to get answers on questions like: 'How many super-classes does a class typically have ?', 'How many subclasses ?', and 'How many classes are defined using property restrictions ?'. We did not consider determining distributions for EquivalentClass-statements, as these statements occurred too rarely to come up with a statistically sound, viz. significant, argument.

Figure 2 displays the distribution of sub-classes, super-classes and restrictions per class expression. The y-Axis displays the percentage of class expressions, which have a certain number of subclasses, superclasses or restrictions. The x-Axis represents this number. The last value (15) aggregates all greater numbers, hence the percentage of class expressions is also aggregated. As we can see the distributions are highly inhomogeneous. As our analysis was performed on the syntactic declarations, the semantic properties of description logics, namely

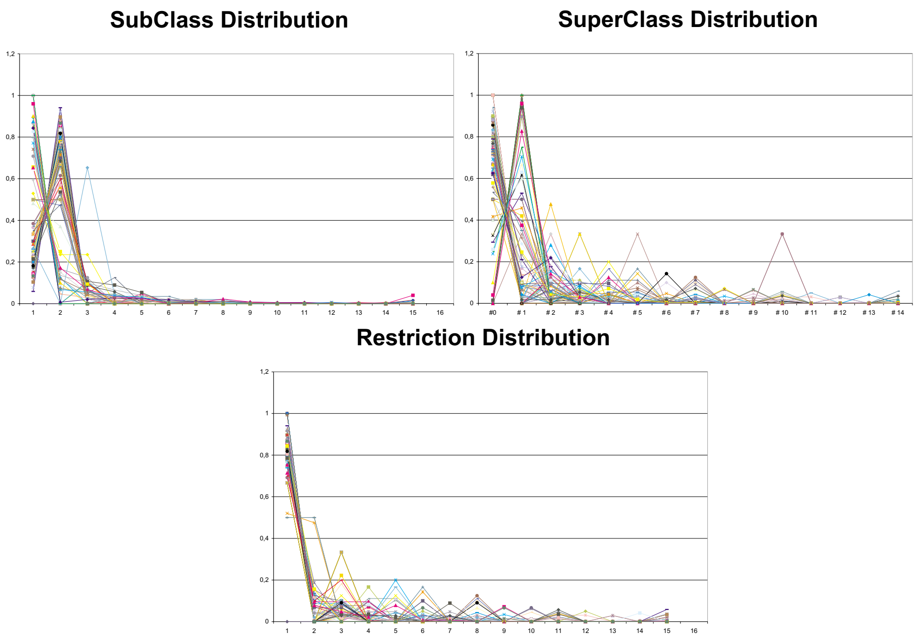


Fig. 2. Distribution of SubClass, SuperClass and Restrictions per Class Expression

that each class is a subclass of daml:Thing are not considered in the distributions, if this were done every class but daml:Thing would have one super-class. Actually, the found ontologies were inconsistent in this respect. Several ontologies redeclared daml:Thing in another namespace (usually the namespace of the ontology). Thing was explicitly assigned as the super-class of a class repeatedly (although this automatically sanctioned by the semantics of the language). Again, these effects were not considered in the analysis.

4 Categories of Ontologies

As discussed before, the distributions of different language primitives is inhomogeneous. However, a quick glimpse on the ontologies suggests that there are different classes of ontologies with a more homogeneous use of those primitives. Thus we applied a clustering algorithm to the data, and indeed found three different clusters.

4.1 Clustering

In order to apply the clustering, a normalization of the data was carried out by using the number of defined classes as denominator. Input values for the clustering algorithm were those language primitives, which were used at least 11 times across all ontologies[10]. The data set consisted of the 95 error-free on-

[10] We chose this number due to the consideration, that a primitive with lower usage, given the small number of ontologies, can only disturb the result.

tologies, which were characterized by 10 attributes, namely Class expressions (95)[11], Primitive Classes(95), Restrictions(69), All Restrictions (48), Cardinality Restrictions (52), Cardinality Restriction covered by OWL Lite (52), Properties (92), Datatype Properties (68), Object Properties (63), and Individuals (19).

We used the WEKA machine learning package to analyze the data, in particular the clustering packages. All attributes have a value range as real value. Hence, we expect unambiguous results from the clustering algorithm, since no transformations need to be applied.

The best results were identified using the k-means[5] clustering algorithm, which initially chooses k random seed points as cluster centroids. It then repeatedly aligns data points to the nearest seed point and calculates the new cluster centers by averaging the assigned data points. This procedure terminates when a certain terminating condition is reached, in our case that no data point is reassigned to another cluster anymore.

A critical decision with k-means is the number of cluster k. We did not evaluate measures like information loss or others in order to define the best number of clusters. We simply evaluated the attributes defined in table 3 for different k and found that $k = 3$ assigns the ontologies in a reasonable way, that is the the coefficient between the improvement of the homogenization (reduction of the COV measure) and the number of clusters k is maximal. More specifically, the clustering allowed us to decrease the c.o.v coefficient to almost half, namely an average of 2,4 in contrast to the 4,5 in 2 using $k = 3$ clusters.

	Average	Std Dev	Median	Min	Max	C.O.V.
Primitive Classes	414	1683	12	1	9795	4,0
Class Expressions	418	1710	15	1	9795	4,0
Restrictions	1,5	4,7	0	0	25	3,1
Properies	39	46	20	0	179	1,2
Object Properties	8	26	0	0	144	3,0
Datatyp Properties	13	25	0	0	108	1,9
Individuals	73	370	0	0	2157	5,1

Table 3. Average Usage of some language primitives (across clustered ontologies(C1))

4.2 Cluster contents

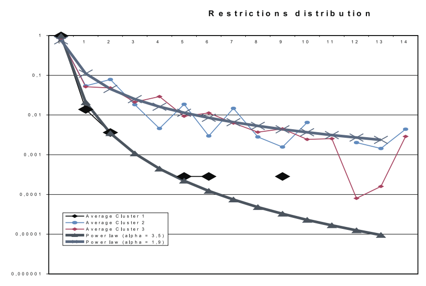
A closer examination of the ontologies assigned to the different clusters reveals that the clusters correspond more or less to three types of ontologies. The largest cluster of ontologies seems to contain ontologies of taxonomic or terminological nature. The ontologies are characterized by few properties and a large number of classes, cf. Table 3.

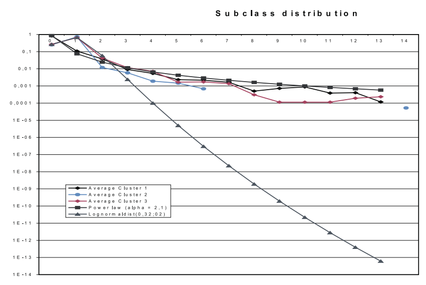
The second largest cluster contains description logic-style ontologies. This cluster is characterized by a high number of axioms per class and a low number of primitive classes. These ontologies also contain a very high number of restrictions and properties (especially datatype properties), however almost no individuals.

The third cluster contains database schema-like ontologies. The ontologies are medium size containing on average 65 class expressions and 25 properties.

[11] The value in brackets specifies the number of ontologies, where values occured

This cluster is more inhomogeneous as indicated by high standard deviations per primitive.





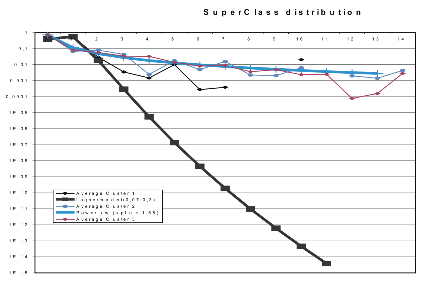


Fig. 3. Average Distribution of SubClass, SuperClass and Restrictions for the Clusters with the estimated distributions (y-axes is log scale)

4.3 Feature Distributions

Having clustered the ontologies into more consistent classes, we now have a look at the distributions of certain features in the taxonomic cluster and the database-like cluster. Again, due to lack of space, we will not look at all feature combinations but rather examine two representative features, namely restrictions per primitive class (database-like)[12] and the distribution of subclasses per class expression (cf. Figure 3 (taxonomic)).

Restrictions In particular we had a look at the distribution of restrictions across classes in the different clusters. In average 1,5[13] (C1), 26[14] (C2) and 30[15] (C3), 17 restrictions are defined in each ontology. Hence, 0.004 (C1), 0.6 (C2) and 0.63 (C3) per class. We compared the observed distributions with the expected values of parameterized distribution functions, in particular the exponential distribution and the power law distribution [9]. Intriguingly the distributions of restrictions closely corresponds to power law distributions with $\alpha = 3,5$ (C1),

[12] The absolute number of restrictions in the taxonomic case is to small to analyze the data expecting significant results.

[13] standard deviation of 4,8

[14] standard deviation of 59

[15] standard deviation of 48

$\alpha = 1,9$ (C2) and $\alpha = 1,8$ (C3). This argument is supported with a confidence value of 99,9 % (using the χ^2-Test). α was estimated to fit the average of the observed distribution. In this case the estimated standard deviation differs at most 16% (C1) from the actual standard deviation. In case of cluster C3 with the most restrictions the difference is just 2% which underlines the argument for a power law distribution.

Sub Classes per Class Considering the distribution of sub classes per class, we found that in the taxonomic-like cluster (C1) each class had 0.30 subclasses with a standard deviation of 0.86. At this point we want to recall, that we did not apply any reasoning to the data set. This would probably alter the figures a bit. We found that the distribution of sub classes per classes also follows a power law distribution with $\alpha = 2,2$. As in the case of restrictions, the argument is supported with a confidence value of 99%. However, the estimated standard deviation differs 40% from the actual observation. The other two clusters (C2, C3) seem to follow a lognormal distribution for the occurrence of sub classes for the first two classes, but than the distribution seems more like a power law distribution. However, a look at the distributions for Super Classes per Class shows an inverted picture, with clusters C2, C3 following a power law distribution and cluster C1 a lognormal one.

5 Conclusion

We provided a systematic approach for the creation of benchmarks for knowledge representation systems and presented the results of the first step in benchmark creation - the analysis of available data. Using our analysis of the DAML.org library, we can use generating functions, e.g. an exponential distribution with the calculated mean for the distribution of restrictions, to generate ontologies for benchmarking, that correspond structurally to real-life ontologies.

Our analysis shows, that benchmarks have to consist of several types of ontologies, since the set of analyzed ontologies would otherwise be too inhomogeneous to derive parameters. As our analysis showed, 3 types of ontologies can generally be identified. For each type of ontologies, high confidence values for the generator functions could be shown.

Our future work is concerned with implementation, viz. an online web service to generate synthetic ontologies, and with deriving realistic workloads for modelling user requests. To this extend we plan to monitor existing ontology-based applications, e.g. the OntoWeb portal and the portal of our institute.

References

1. P. Balsinger and A. Heuerding. Comparison of theorem provers for modal logics - introduction and summary. In *Proc. of Tableaux'98*, pages 25–26, 1998.
2. Sean Bechhofer, Raphael Volz, and Philip Lord. Cooking the Semantic Web with the OWL API. In *ISWC 2003*, Sanibel Island, Florida, USA, October 2003.
3. Jeremy Caroll and Jos De Roo. OWL Web Ontology Language Test Cases. Internet: http://www.w3.org/TR/owl-test/, May 2003.

4. DAML.org Ontology Library. Internet: www.daml.org/ontologies, As of July, 25th 2003.
5. J. A. Hartigan and M. A. Wong. Algorithm AS136. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.
6. J. Heinsohn, D. Kudenko, B. Nebel, and H.-J. Profitlich. An Empirical Analysis of Terminological Representation Systems. *Artificial Intelligence*, 68(2):367–397, August 1994. 1994.
7. A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics k, kt, s4. Technical Report IAM-96-015, University of Bern, Switzerland, October 1996.
8. I. Horrocks and P. F. Patel-Schneider. DL systems comparison. In E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL'98)*, pages 55–57. CEUR, May 1998.
9. G. K. Zipf. *Selective Studies and the Principle of Relative Frequency in Language*. Harvard University Press, 1932.

Results of Taxonomic Evaluation of RDF(S) and DAML+OIL Ontologies using RDF(S) and DAML+OIL Validation Tools and Ontology Platforms Import Services

Asunción Gómez-Pérez, M. Carmen Suárez-Figueroa

Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn.
Boadilla del Monte, 28660. Madrid, Spain
`asun@fi.upm.es`
`mcsuarez@delicias.dia.fi.upm.es`

Abstract. Before using RDF(S) and DAML+OIL ontologies in Semantic Web applications, its content should be evaluated from a knowledge representation point of view. In recent years, some RDF(S) and DAML+OIL 'checkers', 'validators', and 'parsers' have been created and several ontology platforms are able to import RDF(S) and DAML+OIL ontologies. Two are the experiments presented in this paper. The first one reveals that the majority of RDF(S) and DAML+OIL parsers (Validating RDF Parser, RDF Validation Service, DAML Validator, and DAML+OIL Ontology Checker) do not detect taxonomic mistakes in ontologies implemented in such languages. So, if such ontologies are imported by ontology platforms, are they able to detect such problems? The second experiment presented in this paper reveals that the majority of the ontology platforms (OilEd, OntoEdit, Protégé-2000, and WebODE) only detect a few of mistakes in concept taxonomies before importing them.

1 Introduction

In recent years, considerable progress has been made in developing the conceptual bases for building technology that allows reusing and sharing ontologies for the Semantic Web. As any other resource used in software applications, ontology content should be evaluated before (re)using it in other ontologies or applications. In that sense, we could say that it is unwise to publish an ontology or to implement software that relies on ontologies written by others (even by yourself) without first evaluating its content, that is, its concept definitions, its taxonomy and its formal axioms.

Ontology evaluation is an important activity to be carried out during the whole ontology life-cycle. Up to now, few domain-independent methodological approaches [6, 11, 15, 17] include an evaluation activity.

The first works on ontology content evaluation started in 1994 [9, 10], and in the last three years the interest of the Ontological Engineering community in this issue has grown. The main efforts were made by Gómez-Pérez [7, 8] and by Guarino and colleagues with the OntoClean method [12]. ODEClean [5] is a tool integrated into the WebODE environment that gives support to the OntoClean method.

With the increasing number of ontologies implemented in the ontology markup languages RDF(S) [3, 13] and DAML+OIL [18], many specialized ontology validation tools for these languages have been built: Validating RDF Parser[1], RDF Validation Service[2], DAML Validator[3], DAML+OIL Ontology Checker[4], etc. These tools are mainly focused on evaluating ontologies from a syntactic point of view, that is, checking whether the ontologies are compliant with the languages specification. However, they are not focused on detecting mistakes from a knowledge representation point of view, that is, if the ontologies have inconsistencies and redundancies.

We have performed experiments with 24 ontologies (7 on RDF(S) and 17 on DAML+OIL), which are well built from a syntactic point of view, according to the languages specifications, but have inconsistencies and redundancies. We have parsed them with the previous four tools and we have discovered that on the majority of the experiments, they do not detect the taxonomic mistakes identified in [7].

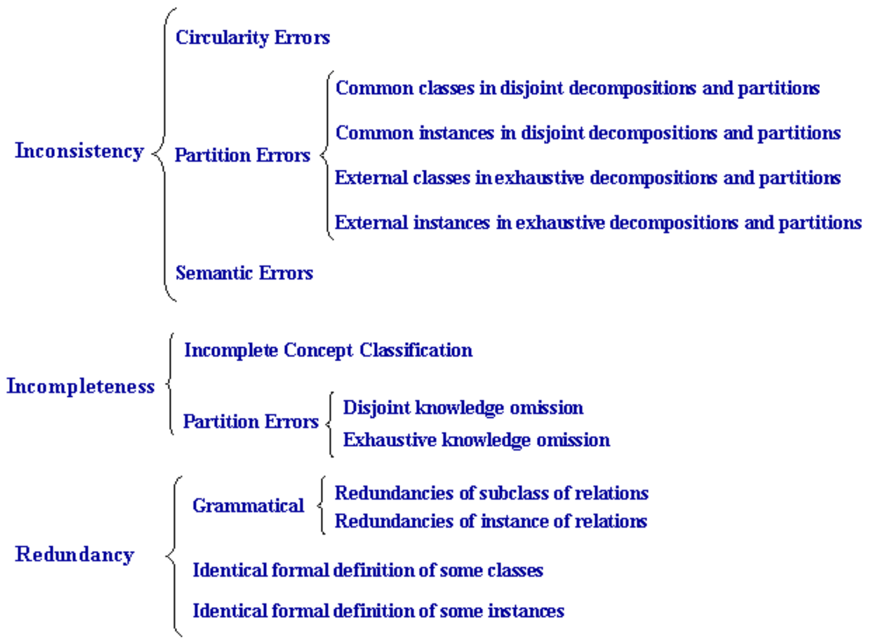
The key point is that RDF(S) and DAML+OIL ontologies are imported by ontology platforms. In fact, OilEd [2], OntoEdit [16], Protégé-2000 [14], and WebODE [4, 1] are able to import ontologies implemented in both languages, but there are not previous works analysing whether such platforms are able to detect wrong RDF(S) and DAML+OIL ontologies. In order to carry out this analysis, we have used the same 24 ontologies (7 on RDF(S) and 17 on DAML+OIL) and we have imported them within the previous ontology platforms. We have found out that on the majority of the experiments, these ontology platforms do not detect mistakes in concept taxonomies represented in RDF(S) and DAML+OIL.

This paper is organized as follows, section two presents briefly the method for evaluating taxonomic knowledge in ontologies. Section three presents a description of some ontology 'checkers', 'validators', and 'parsers'. Section four includes our first comparative study, including examples of the RDF(S) and DAML+OIL ontologies used on the testbed. Section five presents an overview of some ontology platforms. Section six presents the results of importing RDF(S) and DAML+OIL ontologies with taxonomic mistakes in the ontology platforms. Finally, we conclude with further work on evaluation.

2 Method for Evaluating Taxonomic Knowledge in Ontologies

Figure 1 presents a set of possible mistakes that can be made by ontologists when modeling taxonomic knowledge in an ontology under a frame-based approach [7]. In this paper we only focus on inconsistency mistakes (circularity and partition) and redundancy mistakes (grammatical), and we postpone the analysis of the others for further works. Below we explain briefly the studied mistakes.

1 http://139.91.183.30:9090/RDF/VRP/
2 http://www.w3.org/RDF/Validator/
3 http://www.daml.org/validator/
4 http://potato.cs.man.ac.uk/oil/Checker



Inconsistency
Circularity Errors
Partition Errors
Common classes in disjoint decompositions and partitions
Common instances in disjoint decompositions and partitions
External classes in exhaustive decompositions and partitions
External instances in exhaustive decompositions and partitions
Semantic Errors
Incompleteness
Incomplete Concept Classification
Partition Errors
Disjoint knowledge omission
Exhaustive knowledge omission
Redundancy
Grammatical
Redundancies of subclass of relations
Redundancies of instance of relations
Identical formal definition of some classes
Identical formal definition of some instances

Figure 1. Types of mistakes that might be made when developing taxonomies with frames

Inconsistency: Circularity Errors occur when a class is defined as a specialization or generalization of itself. Depending on the number of relations involved, circularity errors can be classified as circularity errors at distance zero (a class with itself), circularity errors at distance 1, and circularity errors at distance n.

Inconsistency: Partition errors. Concept classifications can be defined in a disjoint (disjoint decompositions), a complete (exhaustive decompositions), and a disjoint and complete manner (partitions). The following types of partition errors are identified:

- *Common classes in disjoint decompositions and partitions*. These occur when there is a disjoint decomposition or a partition *class-*p_1,…, *class-*p_n defined in a class *class-A*, and one or more classes *class-*B_1,..., *class-*B_k are subclasses of more than one *class-*p_i.
- *Common instances in disjoint decompositions and partitions*. These errors happen when one or several instances belong to more than one class of a disjoint decomposition or partition.
- *External classes in exhaustive decompositions and partitions*. They occur when having defined an exhaustive decomposition or a partition of the base class (*class-A*) into the set of classes *class-*p_1,..., *class-*p_n, and there are one or more classes that are subclasses of the *class-A*, instead of being subclasses of a class the set of classes *class-*p_1,..., *class-*p_n.
- *External instances in exhaustive decompositions and partitions*. These errors occur when we have defined an exhaustive decomposition or a partition of the base class (*class-A*) into the set of classes *class-*p_1,..., *class-*p_n, and there are one

or more instances of the *class-A* that do not belong to any class *class-*p_i of the exhaustive decomposition or partition.

Redundancy: Grammatical Errors.

- *Redundancies of 'subclass-of' relations* occur between classes they have more than one 'subclass-of' relation. We can distinguish direct and indirect repetition.
- *Redundancies of 'instance-of' relations*. As in the above case, we can distinguish between direct and indirect repetition.

3 Ontology 'Checkers', 'Validators' and 'Parsers'

At the moment, there exist various ontology 'checkers', 'validators', and 'parsers' which are intended to carry out some kind of validation and/or checking of ontologies on diverse web-based languages. In this paper, we focus on the most frequently used parsers that validate and/or check ontologies on RDF(S) and DAML+OIL: Validating RDF Parser and RDF Validation Service for RDF(S), and DAML Validator and DAML+OIL Ontology Checker for DAML+OIL. Other parsers not included in this paper are: Rapier RDF Parser[5], Thea RDF Parser[6], Chimaera[7], ConsVISor[8], etc.

The Validating RDF Parser. The ICS-FORTH RDFSuite[9] is a suite of tools for RDF metadata management. This RDFSuite consists of tools for parsing, validating, storing and querying RDF descriptions, namely the Validating RDF Parser (VRP), the RDF Schema Specific DataBase (RSSDB) and the RDF Query Language (RQL). The ICS-FORTH Validating RDF Parser (VRP v2.5)[10] analyzes, validates and processes RDF schemas and resource descriptions. This parser offers the following functions:

- *Syntactic Validation* for checking if the RDF/XML syntax of the input namespace conforms to the updated RDF/XML syntax proposed by W3C.
- *Semantic Validation* for verifying the selected constraints derived from RDF Schema Specification (RDFS). VRP allows to choose several semantic validation constraints: class hierarchy loops, property hierarchy loops, domain and range of subproperties, source and target resources of properties, and types of resources.

RDF Validation Service. The W3C RDF Validation Service[11] is based on HP-Labs Another RDF Parser (ARP[12]), which currenlty uses the version 2-alpha-1. This online service supports the Last Call Working Draft specifications issued by the RDF Core Working Group, including datatypes. This online service offers the following functions:

- *Syntactic Validation* for checking if the input namespace conforms to the updated RDF/XML Syntax Specification proposed by W3C.

[5] http://www.redland.opensource.ac.uk/raptor/
[6] http://www.semanticweb.gr/
[7] http://www.ksl.stanford.edu/software/chimaera/
[8] http://vis.home.mindspring.com/index.html
[9] Partially supported by EU projects C-Web (IST-1999-13479), MesMuses (IST-2001- 26074), and QUESTION-HOW (IST-2000-28767)
[10] http://139.91.183.30:9090/RDF/VRP/index.html
[11] http://www.w3.org/RDF/Validator/
[12] ARP was created and is maintained by Jeremy Carroll at HP-Labs in Bristol

- *Semantic Validation*. The service does not do any RDF Schema Specification validation.

DAML Validator. The DAML Validator[13] is available via either a WWW interface or download. The Validator uses the ARP parser from the Jena (1.6.1) toolkit to create an RDF triple model from the input code being validated. The DAML Validator checks DAML+OIL markup for problems beyond simple syntax errors. The Validator reads in a DAML file and examines it for a variety of potential errors. The output is a list of indications (errors, warnings, or information), a pointer to the errors in the file, and some guidance on the nature of the problems. It offers the following functions:

- *Syntactic Validation* for checking for namespace problems (outdated URIs, file extensions in URIs) during model creation. The validator tests RDF resources for existence: any subject, or object resource that is referenced must have a defined type.
- *Semantic Validation* for verifying the global domain and range constraints of the predicate. The subject and object of a statement should be instances of the predicate's domain and range classes. Each node (RDF Resource and it's accompanying statements) is validated based on the following types: Class, Property, Restriction, ObjectRestriction, DatatypeRestriction, or an Instance of one or more classes.

DAML+OIL Ontology Checker. The DAML+OIL Checker[14] was developed by University of Manchester (UK). The DAML+OIL Checker is a servlet that uses the OilEd codebase to check the syntax of DAML+OIL ontologies and returns a report on the classes and properties in the model. This checker is a web interface to check DAML+OIL ontologies and content using Jena. It offers the following functions:

- *Syntactic Validation* for checking missing definitions. The checker is fairly strict about the format of the input: in particular "rdf:ID attributes" must be conforming XML names, and unqualified attributes should not be used.
- *Semantic Validation* for verifying class hierarchy loops.

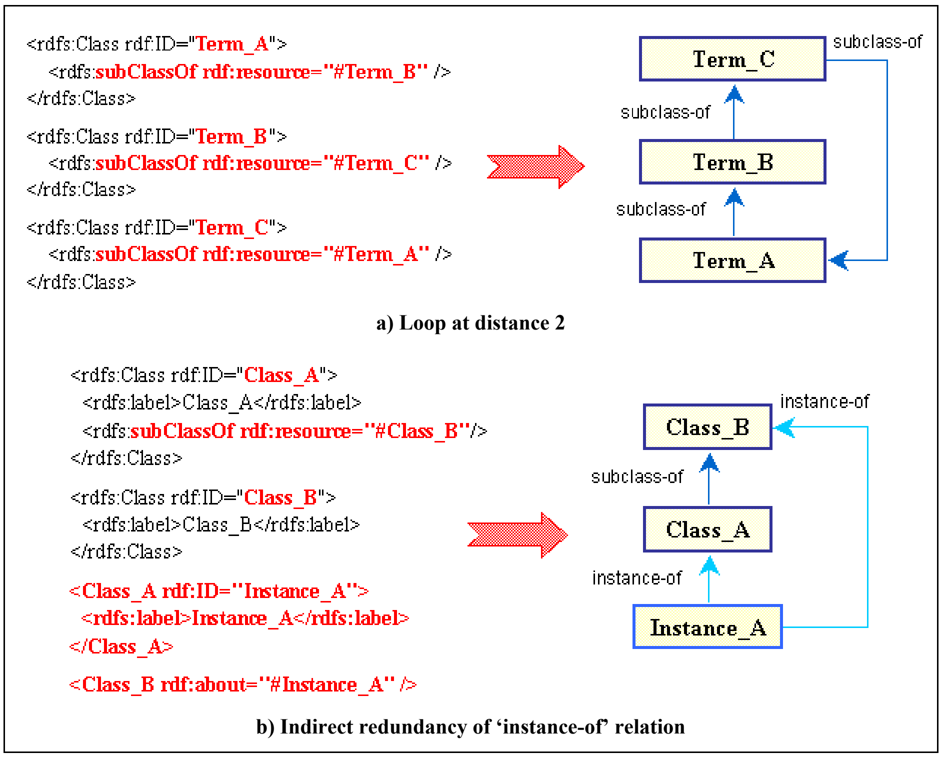
4 Comparative Study of RDF(S) and DAML+OIL 'Checkers', 'Validators' and 'Parsers'

As we said before, the first goal of this paper is to analyse whether RDF(S) and DAML+OIL parsers presented in section 3 detect the concept taxonomy mistakes presented in section 2. In order to achieve this goal, we have built a testbed of 24 ontologies (7 in RDF(S) and 17 in DAML+OIL), each of which implements one of the errors presented in section 2. And we have parsed them with the previous parsers. In the case of RDF(S) we have only 7 ontologies because partitions cannot be defined in this language.

These ontologies and the results of their evaluation can be found at http://minsky.dia.fi.upm.es/odeval/index.html.

[13] http://www.daml.org/validator/
[14] http://potato.cs.man.ac.uk/oil/Checker

In figure 2 we show the RDF(S) code and graphical notation of two of these ontologies: the one that implements the circularity error at distance 2, and the one that implements the mistake of indirect redundancy of 'instance-of' relation. Figure 3 shows the DAML+OIL code and graphical notation of three of these ontologies: the one that implements the circularity error at distance 1, the one that implements the mistake of common class in disjoint decomposition, and the last one that implements the mistake of external instance in partition.



```
<rdfs:Class rdf:ID="Term_A">
   <rdfs:subClassOf rdf:resource="#Term_B" />
</rdfs:Class>
<rdfs:Class rdf:ID="Term_B">
   <rdfs:subClassOf rdf:resource="#Term_C" />
</rdfs:Class>
<rdfs:Class rdf:ID="Term_C">
   <rdfs:subClassOf rdf:resource="#Term_A" />
</rdfs:Class>
```

a) Loop at distance 2

```
<rdfs:Class rdf:ID="Class_A">
  <rdfs:label>Class_A</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Class_B"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Class_B">
  <rdfs:label>Class_B</rdfs:label>
</rdfs:Class>
<Class_A rdf:ID="Instance_A">
  <rdfs:label>Instance_A</rdfs:label>
</Class_A>
<Class_B rdf:about="#Instance_A" />
```

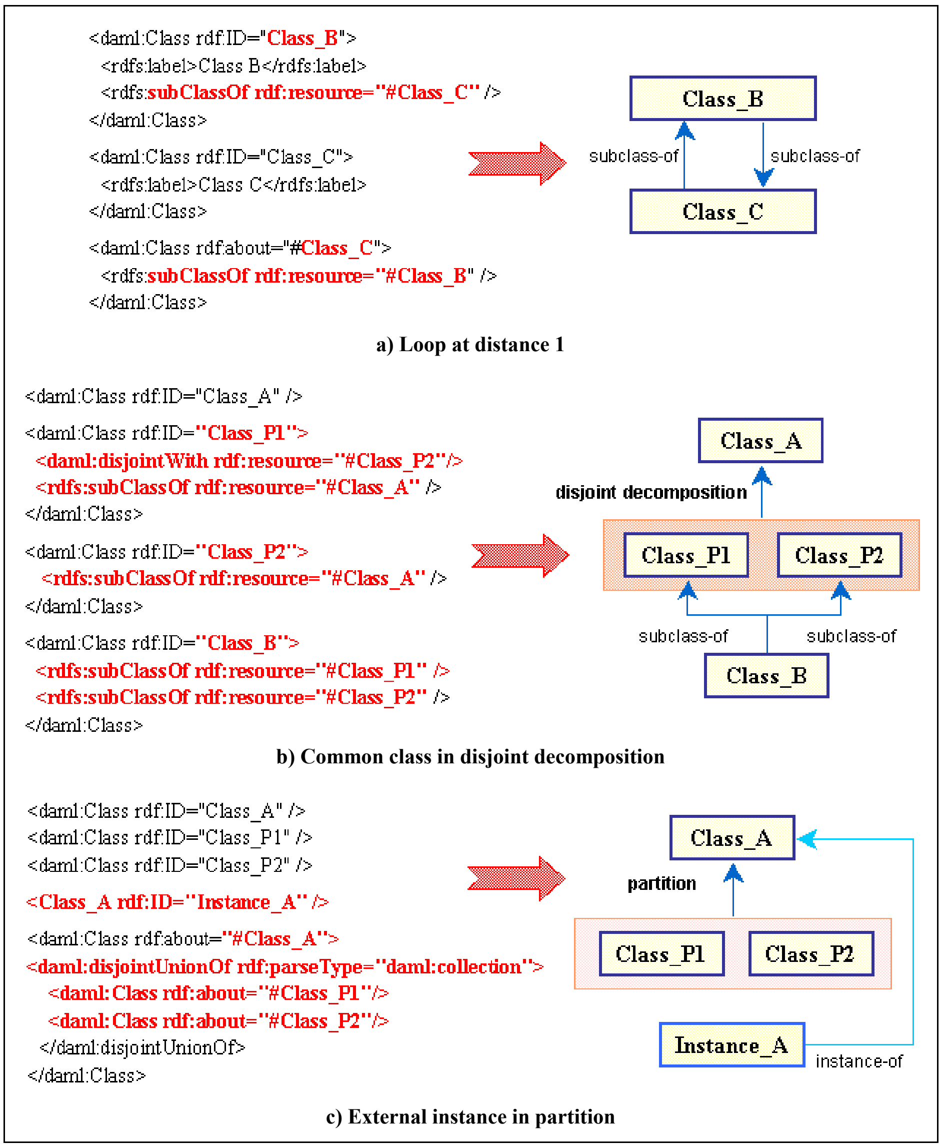
b) Indirect redundancy of 'instance-of' relation

Figure 2. Examples of RDF(S) ontologies

After parsing the ontologies on the testbed with the parsers, we found that all these parsers recognised the code as well formed code, but the majority had problems detecting most of the knowledge representation mistakes that these ontologies contained.

The results of analysing and comparing these parsers are shown in table 1. The symbols used in this table are the following:

⃠: The parser does not accept files written in this language
✓: The parser detects the mistake in this language
×: The parser does not detect the mistake in this language
--: The mistake cannot be represented in this language



```
<daml:Class rdf:ID="Class_B">
  <rdfs:label>Class B</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Class_C" />
</daml:Class>
<daml:Class rdf:ID="Class_C">
  <rdfs:label>Class C</rdfs:label>
</daml:Class>
<daml:Class rdf:about="#Class_C">
  <rdfs:subClassOf rdf:resource="#Class_B" />
</daml:Class>
```

a) Loop at distance 1

```
<daml:Class rdf:ID="Class_A" />
<daml:Class rdf:ID="Class_P1">
 <daml:disjointWith rdf:resource="#Class_P2"/>
 <rdfs:subClassOf rdf:resource="#Class_A" />
</daml:Class>
<daml:Class rdf:ID="Class_P2">
  <rdfs:subClassOf rdf:resource="#Class_A" />
</daml:Class>
<daml:Class rdf:ID="Class_B">
 <rdfs:subClassOf rdf:resource="#Class_P1" />
 <rdfs:subClassOf rdf:resource="#Class_P2" />
</daml:Class>
```

b) Common class in disjoint decomposition

```
<daml:Class rdf:ID="Class_A" />
<daml:Class rdf:ID="Class_P1" />
<daml:Class rdf:ID="Class_P2" />
<Class_A rdf:ID="Instance_A" />
<daml:Class rdf:about="#Class_A">
<daml:disjointUnionOf rdf:parseType="daml:collection">
   <daml: Class rdf:about="#Class_P1"/>
   <daml: Class rdf:about="#Class_P2"/>
  </daml:disjointUnionOf>
</daml:Class>
```

c) External instance in partition

Figure 3. Examples of DAML+OIL ontologies

As we can see in table 1, we have checked whether RDF(S) tools (VRP and RDF Validation Service) were able to evaluate DAML+OIL files, and whether DAML+OIL tools (DAML Validator and DAML+OIL Ontology Checker) were able to evaluate RDF(S) files. In the case of RDF(S) tools, the experiments showed that RDF Validation Service can read DAML+OIL ontologies, although it does not detect the mistakes, but VRP cannot read them. In the case of DAML+OIL tools, the experiments showed that both of them are able to recognize RDF(S) files. Although the DAML+OIL Ontology Checker is not a RDF(S) validation tool, it was able to detect circularity errors in that language.

Before going in detail with circularity errors, we have an important comment to make. The RDF(S) and DAML+OIL specifications allow cycles in concept taxonomies. However, we consider that this is a mistake from the knowledge representation point of view, that is, we would not recommend designing ontologies with cycles in their concept taxonomies. So here we want to stress the distinction between checking an ontology from a syntactic point of view (checking whether the ontology is compliant with the language specification) and checking an ontology from a knowledge representation point of view (checking whether the ontology does not have the mistakes presented in section 2).

Circularity errors are the only ones detected by some of the parsers studied in this experiment. VRP is able to detect circularity errors at any distance in RDF(S) ontologies, indicating that there is a semantic error ("loop detected"). The DAML+OIL Ontology Checker detects circularity errors at any distance in RDF(S) and DAML+OIL ontologies, throwing a warning about it ("cycles in class hierarchy").

Regarding *partition errors*, they have only been studied for DAML+OIL, since they cannot be represented in RDF(S). None of the DAML+OIL validators, neither the RDF Validation Service, have detected partition errors with the 10 ontologies from the testbed.

The same occurs with the *grammatical redundancy errors*, which are not detected by any of the RDF(S) and DAML+OIL parsers studied.

5 Ontology Platforms

In this paper we focus on the most representative ontology platforms that can be used for importing ontologies: OilEd, OntoEdit, Protégé-2000, and WebODE. In this section, we provide a broad overview of these ontology platforms.

OilEd[15] [2] was initially developed as an ontology editor for OIL ontologies, in the context of the European IST OntoKnowledge project. However, OilEd has evolved and now is an editor of DAML+OIL and OWL ontologies. OilEd can import ontologies implemented in RDF(S), OIL, DAML+OIL, and the SHIQ XML format. Besides exporting ontologies to DAML+OIL, OilEd ontologies can be exported to the RDF(S) and OWL ontology languages and to the XML formats SHIQ and DIG.

OntoEdit[16] [16] has been developed by AIFB in Karlsruhe University. It is an extensible and flexible environment, based on a plugin architecture, which provides functionality to browse and edit ontologies. It includes plugins for reasoning using Ontobroker, plugins for exporting and importing ontologies in different formats (FLogic, OXML, RDF(S), DAML+OIL), etc. Two versions of OntoEdit are available: OntoEdit Free and OntoEdit Professional.

[15] http://oiled.man.ac.uk
[16] http://www.ontoprise.de/com/start_downlo.htm

			ICS-FORTH Validating RDF Parser		RDF Validation Service		DAML Validator		DAML+OIL Ontology Checker	
			RDF(S)	DAML+OIL	RDF(S)	DAML+OIL	RDF(S)	DAML+OIL	RDF(S)	DAML+OIL
Inconsistency: Circularity Errors	At distance zero		✓	⊘	×	×	×	×	✓	✓
	At distance one		✓	⊘	×	×	×	×	✓	✓
	At distance n		✓	⊘	×	×	×	×	✓	✓
Inconsistency: Partition Errors	Common classes in disjoint decompositions	Direct	--	⊘	--	×	--	×	--	×
		Indirect	--	⊘	--	×	--	×	--	×
	Common classes in partitions		--	⊘	--	×	--	×	--	×
	Common instances in disjoint decompositions	Direct	--	⊘	--	×	--	×	--	×
		Indirect	--	⊘		×	--	×	--	×
	Common instances in partitions		--	⊘	--	×	--	×	--	×
	External classes in exhaustive decompositions		--	⊘	--	×	--	×	--	×
	External classes in partitions		--	⊘	--	×	--	×	--	×
	External instances in exhaustive decompositions		--	⊘	--	×	--	×	--	×
	External instances in partitions		--	⊘	--	×	--	×	--	×
Redundancy: Grammatical Errros	Redundancies of 'subclass-of' relations	Direct	×	⊘	×	×	×	×	×	×
		Indirect	×	⊘	×	×	×	×	×	×
	Redundancies of 'instance-of' relations	Direct	×	⊘	×	×	×	×	×	×
		Indirect	×	⊘	×	×	×	×	×	×

Table 1. Results of the analysis of the RDF(S) and DAML+OIL parsers

Protégé-2000[17] [14] has been developed by the Stanford Medical Informatics (SMI) at Stanford University, and is the latest version of the Protégé line of tools. It is an open source, standalone application with an extensible architecture. The core of this environment is the ontology editor, and it holds a library of plugins that add more functionality to the environment (ontology language importation and exportation, OKBC access, constraints creation and execution, etc.). Protégé-2000 ontologies can be exported and imported with some of the backends provided in the standard release or as plugins: RDF(S), DAML+OIL, OWL, XML, XML Schema, and XMI.

WebODE[18] [4, 1] has been developed by the Ontology Engineering Group at Universidad Politécnica de Madrid (UPM). It is an ontology-engineering suite created with an extensible architecture. WebODE is not used as a standalone application, but as a Web application. There are several services for ontology language import and export (XML, RDF(S), DAML+OIL, OIL, OWL, CARIN, FLogic, Jess, Prolog), axiom edition with WAB (WebODE Axiom Builder), ontology documentation, ontology evaluation, and ontology merge.

6 Comparative Study of Ontology Platforms Import Services

As we said before, the second main goal of this paper is to analyse whether ontology platforms presented in section 5, are able to detect taxonomic mistakes in RDF(S) and DAML+OIL ontologies before importing them.

In order to carry out this experiment, we have reused the same 24 ontologies (7 in RDF(S) and 17 in DAML+OIL with inconsistency and redundancy mistakes) used in the previous experiment. In the case of RDF(S) we have only 7 ontologies because partitions cannot be defined in this language. We have imported these ontologies using the import facilities of the ontology platforms presented in section 5. Table 2 presents the results of the experiment using the following symbols:

- 😐 : The ontology platform does not allow representing this type of mistake
- ✓ : The ontology platform detects the mistake during ontology import
- × : The ontology platform does not detect the mistake during ontology import
- -- : The mistake cannot be represented in this language

The main conclusions of the RDF(S) and DAML+OIL ontology import are:

Circularity errors at any distance are the only ones detected by most of ontology platforms analyzed in this experiment. However, OntoEdit Free does not detect circularity errors at distance zero, but it ignores them.

Regarding *partition errors*, we have only studied DAML+OIL ontologies because this type of knowledge cannot be represented in RDF(S). Most of ontology platforms used in this study do not detect partition errors in DAML+OIL ontologies. Furthermore, some partition errors (common instance in partitions, external instance

[17] http://protege.stanford.edu/plugins.html
[18] http://webode.dia.fi.upm.es/

in exhaustive decompositions, etc.) cannot be represented in the ontology platforms studied. Only WebODE detects some partition errors using the ODEval[19] service.

Grammatical redundancy errors are not detected by most of ontology platforms used in this work. However some ontology platforms ignore direct redundancies of 'subclass-of' or 'instance-of' relations. As the previous case, only WebODE detects indirect redundancies of 'subclass-of' relations in RDF(S) and DAML+OIL ontologies using the ODEval service.

7 Conclusions and Further Work

In this paper we have shown that, in general, current RDF(S) and DAML+OIL 'checkers', 'validators', and 'parsers' are not able to detect mistakes from a knowledge representation point of view, but they mainly focus on the syntactic validation of the RDF(S) and DAML+OIL ontologies that they parser.
We have also shown that only a few taxonomic mistakes in RDF(S) and DAML+OIL ontologies are detected by ontology platforms which are able to import ontologies in such languages.

Taking into account that only a few parsers are able to detect loops in RDF(S) and DAML+OIL taxonomies, we considered that it is necessary to create more advanced evaluators than those already existing for evaluating RDF(S) and DAML+OIL from a knowledge representation point of view.

We also consider that it is necessary to create more advanced ontology import services in ontology platforms.

We think that much work must be made to integrate ontology evaluation functions in ontology development tools, and to create an integrated ontology evaluation tool suite that will permit analyzing ontologies in different languages and KR formalisms.

Acknowledgements

This work has been supported by the Esperonto project (IST-2001-34373), by the Spanish project 'Plataforma Tecnológica para la web semántica: Ontologías, análisis de lenguaje natural y comercio electrónico' (TIC-2001-2745), and by a research grant from UPM ("Beca asociada a proyectos modalidad B").

We thanks for his comments and revisions to Óscar Corcho.

[19] http://minsky.dia.fi.upm.es/odeval

			OilEd		OntoEdit Free		Protégé-2000		WebODE	
			RDF(S)	DAML+OIL	RDF(S)	DAML+OIL	RDF(S)	DAML+OIL	RDF(S)	DAML+OIL
Inconsistency: Circularity Errors	At distance zero		✓	✓	×	×	✓	✓	✓	✓
	At distance one		✓	✓	✓	✓	✓	✓	✓	✓
	At distance n		✓	✓	✓	✓	✓	✓	✓	✓
Inconsistency: Partition Errors	Common classes in disjoint decompositions	Direct	--	×	--	×	--	×	--	✓
		Indirect	--	×	--	×	--	×	--	✓
	Common classes in partitions		--	×	--	😐	--	×	--	✓
	Common instances in disjoint decompositions	Direct	--	×	--	×	--	😐	--	😐
		Indirect	--	×	--	×	--	😐	--	😐
	Common instances in partitions		--	×	--	😐	--	😐	--	😐
	External classes in exhaustive decompositions		--	×	--	😐	--	×	--	😐
	External classes in partitions		--	×	--	😐	--	×	--	✓
	External instances in exhaustive decompositions		--	×	--	😐	--	×	--	😐
	External instances in partitions		--	×	--	😐	--	×	--	✓
Redundancy: Grammatical Errors	Redundancies of subclass-of relations	Direct	×	×	×	×	×	×	×	×
		Indirect	×	×	×	×	×	×	✓	✓
	Redundancies of instance-of relations	Direct	×	×	×	×	×	😐	😐	😐
		Indirect	×	×	×	×	😐	😐	😐	😐

Table 2. Results of the RDF(S) and DAML+OIL ontology import

References

1. Arpírez JC, Corcho O, Fernández-López M, Gómez-Pérez A (2003) *WebODE in a nutshell*. AI Magazine To be published in 2003
2. Bechhofer S, Horrocks I, Goble C, Stevens R (2001) *OilEd: a reason-able ontology editor for the Semantic Web*. In: Baader F, Brewka G, Eiter T (eds) Joint German/Austrian conference on Artificial Intelligence (KI'01). Vienna, Austria. (Lecture Notes in Artificial Intelligence LNAI 2174) Springer-Verlag, Berlin, Germany, pp 396–408
3. Brickley D, Guha RV (2003) *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Working Draft. http://www.w3.org/TR/PR-rdf-schema
4. Corcho O, Fernández-López M, Gómez-Pérez A, Vicente O (2002) *WebODE: an Integrated Workbench for Ontology Representation, Reasoning and Exchange*. In: Gómez-Pérez A, Benjamins VR (eds) 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02). Sigüenza, Spain. (Lecture Notes in Artificial Intelligence LNAI 2473) Springer-Verlag, Berlin, Germany, pp 138–153
5. Fernández-López M, Gómez-Pérez A (2002) *The Integration of OntoClean in WebODE*. In: Angele J, Sure Y (eds) EKAW02 Workshop on Evaluation of Ontology-based Tools (EON2002), Sigüenza, Spain, pp 38-52.
6. Fernández-López M, Gómez-Pérez A, Pazos-Sierra A, Pazos-Sierra J (1999) *Building a Chemical Ontology Using METHONTOLOGY and the Ontology Design Environment*. IEEE Intelligent Systems & their applications 4(1) (1999) 37-46.
7. Gómez-Pérez A (2001) *Evaluating ontologies: Cases of Study. IEEE Intelligent Systems and their Applications*. Special Issue on Verification and Validation of ontologies. Marzo 2001, Vol 16, Nº 3. Pag. 391 – 409.
8. Gómez-Pérez A (1996) *A Framework to Verify Knowledge Sharing Technology*. Expert Systems with Application. Vol. 11, N. 4. PP: 519-529.
9. Gómez-Pérez A (1994) *Some ideas and Examples to Evaluate Ontologies*. Technical Report KSL-94-65. Knowledge System Laboratory. Stanford University. Also in Proceedings of the 11th Conference on Artificial Intelligence for Applications. CAIA94.
10. Gómez-Pérez A (1994) *From Knowledge Based Systems to Knowledge Sharing Technology: Evaluation and Assessment*. Technical Report. KSL-94-73. Knowledge Systems Laboratory. Stanford University. December.
11. Grüninger M, Fox MS (1995) *Methodology for the design and evaluation of ontologies*. In Workshop on Basic Ontological Issues in Knowledge Sharing (Montreal, 1995).
12. Guarino N, Welty C (2000) *A Formal Ontology of Properties* In R. Dieng and O. Corby (eds.), Knowledge Engineering and Knowledge Management: Methods, Models and Tools. 12th International Conference, EKAW2000, LNAI 1937. Springer Verlag: 97-112. 2000.
13. Lassila O, Swick R (1999) *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. http://www.w3.org/TR/REC-rdf-syntax/
14. Noy NF, Fergerson RW, Musen MA (2000) *The knowledge model of Protege-2000: Combining interoperability and flexibility*. In: Dieng R, Corby O (eds) 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00). Juan-Les-Pins, France. (Lecture Notes in Artificial Intelligence LNAI 1937) Springer-Verlag, Berlin, Germany, pp 17–32
15. Staab S, Schnurr HP, Studer R, Sure Y (2001) *Knowledge Processes and Ontologies*, IEEE Intelligent Systems, 16(1). 2001.
16. Sure Y, Erdmann M, Angele J, Staab S, Studer R, Wenke D (2002) *OntoEdit: Collaborative Ontology Engineering for the Semantic Web*. In: Horrocks I, Hendler JA (eds) First International Semantic Web Conference (ISWC'02). Sardinia, Italy. (Lecture Notes in Computer Science LNCS 2342) Springer-Verlag, Berlin, Germany, pp 221–235

17. Uschold M, Grüninger M (1996) *ONTOLOGIES: Principles, Methods and Applications*. Knowledge Engineering Review. Vol. 11; N. 2; June 1996.
18. van Harmelen F, Patel-Schneider PF, Horrocks I (2001) *Annotated DAML+OIL (March 2001) Markup Language*. Technical Report. http://www.daml.org/2001/03/daml+oil-walkthru.html

Racer: A Core Inference Engine for the Semantic Web

Volker Haarslev[†] and Ralf Möller[‡]

[†]Concordia University, Montreal, Canada (haarslev@cs.concordia.ca)
[‡]Technical University Hamburg-Harburg, Germany (ra.moeller@tu-harburg.de)

Abstract. In this paper we describe Racer, which can be considered as a core inference engine for the semantic web. The Racer inference server offers two APIs that are already used by at least three different network clients, i.e., the ontology editor OilEd, the visualization tool RICE, and the ontology development environment Protege 2. The Racer server supports the standard DIG protocol via HTTP and a TCP based protocol with extensive query facilities. Racer currently supports the web ontology languages DAML+OIL, RDF, and OWL.

1 Motivation

The Semantic Web initiative defines important challenges for knowledge representation and inference systems. Recently, several standards for representation languages have been proposed (RDF, DAML+OIL, OWL). One of the standards for the Semantic Web is the Resource Description Framework (RDF [12]). Since RDF is based on XML it shares its document-oriented view of grouping sets of declarations or statements. With RDF's triple-oriented style of data modeling, it provides means for expressing graph-structured data over multiple documents (whereas XML can only express graph structures within a specific document). As a design decision, RDF can talk about everything. Hence, in principle, statements in documents can also be referred to as resources. In particular, conceptual domain models can be represented as RDF resources. Conceptual domain models are referred to as "vocabularies" in RDF. Specific languages are provided for defining vocabularies (or ontologies). An extension of RDF for defining ontologies is RDF Schema (RDFS [6]) which only can express conceptual modeling notions such as generalization between concepts (aka classes) and roles (aka properties). For properties, domain and range restrictions can be specified. Thus, the expressiveness of RDFS is very limited. Much more expressive representation languages are DAML+OIL [15] and OWL [14]. Although still in a very weak way, based on XML-Schema, OWL and DAML+OIL also provide for means of dealing with data types known from programming languages.

The representation languages mentioned above are defined with a model-theoretic semantics. In particular, for the language OWL, a semantics was defined such that very large fragments of the language can be directly expressed using so-called description logics (see [1]). The fragment is called OWL DL.

With some restrictions that are discussed below one can state that the logical basis of OWL (or DAML+OIL) can be characterized with the description logic $\mathcal{SHIQ}(\mathcal{D}_n)^-$ [3] (DAML+OIL documents are to be interpreted in the spirit of OWL DL). This means, with some restrictions, OWL documents can be automatically translated to $\mathcal{SHIQ}(\mathcal{D}_n)^-$ T-boxes. The RDF-Part of OWL documents can be translated to $\mathcal{SHIQ}(\mathcal{D}_n)^-$ A-boxes.

In the remainder of this paper Racer, its APIs, and its inference services are briefly described. The use of Racer as network server is illustrated by RICE offering an interactive visualization and query interface. The paper is concluded by reporting on Racer's constraint-based data types support whose functionally exceeds the current OWL standard.

2 Racer: A Description Logic Inference Engine

The logic $\mathcal{SHIQ}(\mathcal{D}_n)^-$ is interesting for practical applications because highly optimized inference systems are available (e.g., Racer [8]). Racer is freely available for research purposes and can be accessed by standard HTTP or TCP protocols (the Racer program is subsequently also called Racer server). Racer can read DAML+OIL and OWL knowledge bases either from local files or from remote Web servers (i.e., a Racer server is also a HTTP client). In turn, other client programs that need inference services can communicate with a Racer server via TCP-based protocols. OilEd [4] can be seen as a specific client that uses the DIG protocol [5] for communicating with a Racer server, whereas RICE [13] is another client that uses a more low-level TCP protocol providing extensive query facilities (see below).

The DIG protocol is a an XML- and HTTP-based standard for connecting client programs to description logic inference engines. DIG allows for the allocation of knowledge bases and enables clients to pose standard description logic queries. The main ideas behind DIG are described in detail in [5]. As a standard and a least common denominator it cannot encompass all possible forms of system-specific statements and queries. Let alone long term query processing instructions (e.g., exploitation of query subsumption, computation of indexes for certain kinds of queries etc., see [9]). Therefore, Racer provides an additional TCP-based interface in order to send instructions (statements) and queries. For interactive use, the language supported by Racer is not XML- or RDF-based but is largely based on the KRSS standard with some additions and restrictions. The advantage is that users can spontaneously type queries which can be directly sent to a Racer server. We will see below that RICE can be used as a shell for Racer. However, the Racer TCP interface can be very easily accessed from Java or C++ application programs as well. For both languages corresponding APIs are available.

The following code fragment demonstrates how to interact with a Racer server from a Java application using Racer's TCP-based API. The aim of the example is to demonstrate the relative ease of use that such an API provides.

```
public class KillerApplication {
   public static void main(String[] argv) {
      RacerClient client=new RacerClient("racer.cs.concordia.ca", 8088);
      try {
         client.openConnection();
         try {
            String kbName=
              client.send("(owl-read-document
              \"http://www.cs.concordia.ca/~faculty/haarslev/family.owl\")");
            String queryResult=
                client.send("(individual-direct-types |#CHARLES|)");
            System.out.println(racerResult);
            }
         catch (RacerException e) {
            ...
            }
         }
         client.closeConnection();
      } catch (IOException e) {
         ...
      }
   }
}
```

The connection to the Racer server is represented with a client object (of class `RacerClient`). The client sends messages to a Racer server running on the machine with name `"racer.cs.concordia.ca"` on port 8088. The Java program can be run on another computer, of course. The program instructs the Racer server to load an OWL document from a remote server. In addition, the Java client program executes a query and prints the result set.

3 A Selection of Supported Inference Services

In description logic terminology, a tuple consisting of a T-box and an A-box is referred to as a knowledge base. An individual is a specific named object. OWL also allows for individuals in concepts (and T-box axioms). For example, expressing the fact that all humans stem from a single human called ADAM requires to refer to an individual in a concept (and a T-box). Only part of the expressivity of individuals mentioned in concepts can be captured with A-boxes. However, a straightforward approximation exists (see [10]) such that in practice suitable $\mathcal{SHIQ}(\mathcal{D}_n)^-$ ontologies can be generated from an OWL document. Racer can directly read OWL documents and represent them as description logic knowledge bases (aka ontologies). In the following a selection of supported queries is briefly introduced.

- Concept consistency w.r.t. a T-box: Is the set of objects described by a concept empty?
- Concept subsumption w.r.t. a T-box: Is there a subset relationship between the set of objects described by two concepts?

– Find all inconsistent concepts mentioned in a T-box. Inconsistent concepts might be the result of modeling errors.
– Determine the parents and children of a concept w.r.t. a T-box: The parents of a concept are the most specific concept names mentioned in a T-box which subsume the concept. The children of a concept are the most general concept names mentioned in a T-box that the concept subsumes. Considering all concept names in a T-box the parent (or children) relation defines a graph structure which is often referred to as taxonomy. Note that some authors use the name taxonomy as a synonym for ontology.

Whenever a concept is needed as an argument for a query, not only predefined names are possible. If also an A-box is given, among others, the following types of queries are possible:

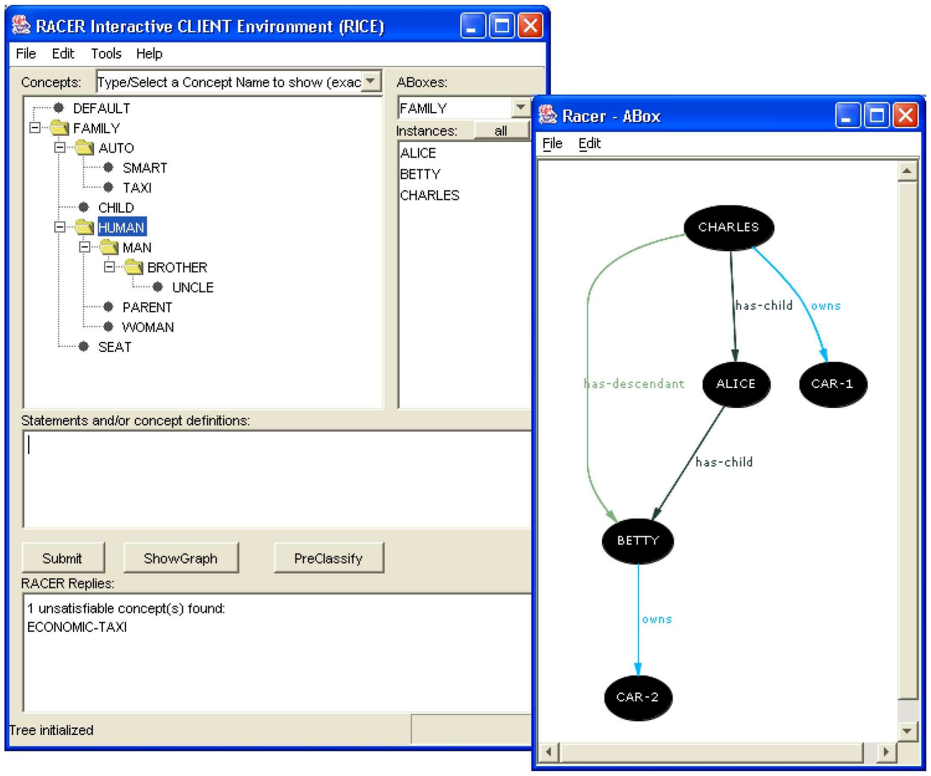
– Check the consistency of an A-box w.r.t. a T-box: Are the restrictions given in an A-box w.r.t. a T-box too strong, i.e., do they contradict each other? Other queries are only possible w.r.t. consistent A-boxes.
– Instance testing w.r.t. an A-box and a T-box: Is the object for which an individual stands a member of the set of objects described by a certain query concept? The individual is then called an instance of the query concept.
– Instance retrieval w.r.t. an A-box and a T-box: Find all individuals from an A-box such that the objects they stand for can be proven to be a member of a set of objects described by a certain query concept.
– Computation of the direct types of an individual w.r.t. an A-box and a T-box: Find the most specific concept names from a T-box of which a given individual is an instance.
– Computation of the fillers of a role with reference to an individual.

Given the background of description logics, many application papers demonstrate how these inference services can be used to solve actual problems with DAML+OIL or OWL knowledge bases. The query interface is extensively used by RICE, which is briefly described in the next section.

4 RICE: Racer Interactive Client Environment

RICE [13] is a tool for Racer that visualizes taxonomies and A-box structures and enables users to interactively define queries using these visualizations. RICE is started as an application program and can be configured to connect to a Racer server by giving a host name and a port. When RICE connects to a Racer server it retrieves all T-boxes and displays them in an unfoldable tree view (in a similar way as OilEd [4] does).

In Figure 1 the taxonomy induced by the T-box specified above is presented (left window, unfoldable tree display). The taxonomy is accompanied by the pane for displaying A-box individuals (to the right of the tree display). Selecting a concept name in the taxonomy corresponds to posing an instance retrieval query with that concept name as a query concept. The result set is displayed in



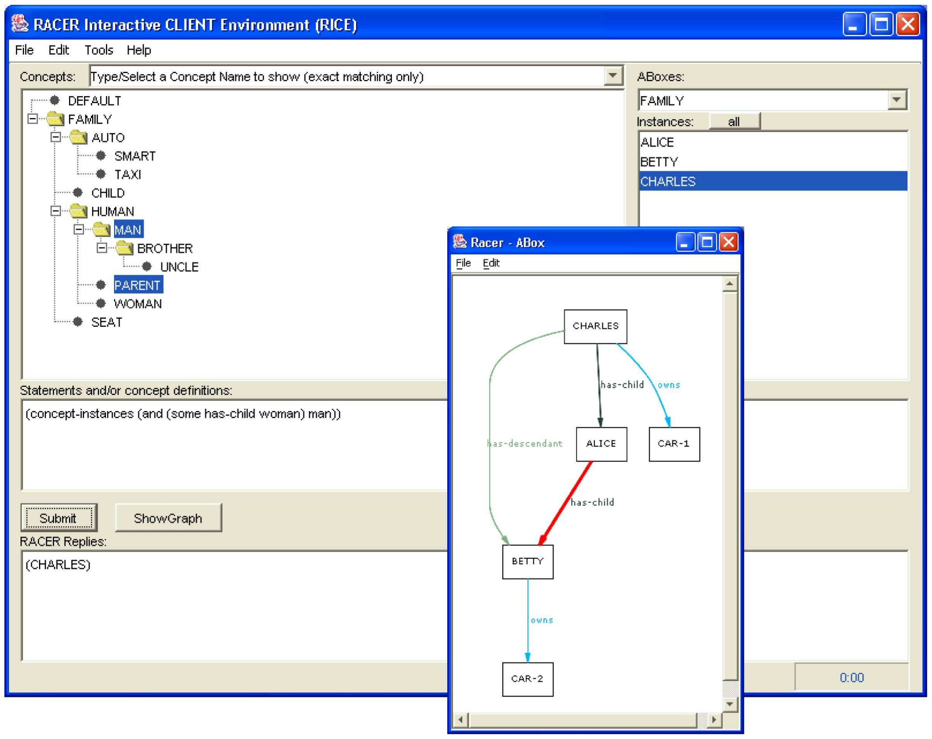
```
1 unsatisfiable concept(s) found:
ECONOMIC-TAXI
```

Fig. 1. Snapshot of RICE displaying an example knowledge base (T-box and A-box).

the instances pane. In the example in Figure 1 all humans are displayed. The structure of the whole A-box can be displayed by pressing the button "Show Graph". The graph window to the right appears. Clicking on individuals is interpreted as posing queries for the direct types of the individuals. In Figure 2 the individual *CHARLES* is selected. The taxonomy is automatically unfolded such that all concept names which are direct types can be seen as highlighted nodes. Figure 2 also demonstrates that graphical attributes (e.g., color, shape) for displaying A-boxes can be (interactively) specified as appropriate.

RICE users can interactively type instructions and queries into the interaction pane in the middle. In Figure 2 we see an instance retrieval query. Query results are printed into the lower pane. Since Racer supports multiple A-boxes, users can interactively select the A-box subsequent queries should refer to (see the main window in Figure 2, top-right selection box). The current T-box can also be easily set by clicking on a T-box name.

If a user specifies a knowledge base with OilEd, Racer can be used to verify and classify it with a single click. The knowledge base is then known to the Racer server. If RICE connects to the Racer server, the knowledge base is visible. Note that OilEd and RICE can access a Racer server in parallel without any problems



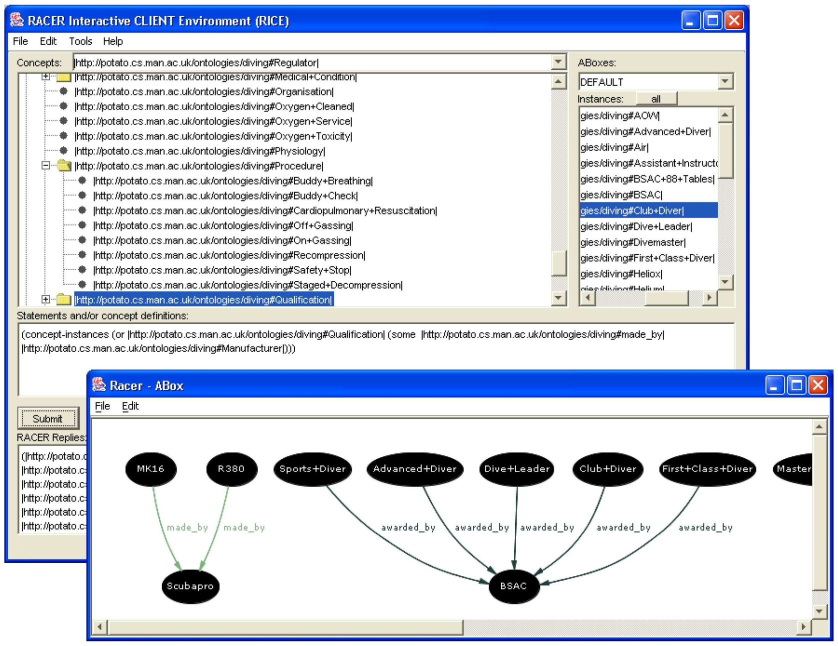
```
(concept-instances (and (some has-child woman) man))
```

```
(CHARLES)
```

Fig. 2. Snapshot of RICE showing the results of a direct types query and an instance retrieval query.

if the Racer Proxy is installed appropriately (see [10]). If the RICE user selects the knowledge base stemming from OilEd (in Figure 3 we used one of the OilEd example files) and presses "Show Graph", the A-box part is shown in a graph display (see Figure 3).

As a summary, we compare OilEd and RICE. OilEd supports DIG, which makes it useful for more reasoners, but is limited to what DIG supports. Furthermore, OilEd provides a graphical means for displaying definitions of concepts and instances. This makes it easy to see what properties are defined and which ones are inherited. OilEd presents unsatisfiable concepts in the taxonomy, whereas they are not shown in RICE. RICE can connect to a Racer server that has already loaded a model, and retrieve its taxonomy (this is not supported by DIG). RICE can add individual DL statements to Racer (although this currently requires full classification of the model involved). RICE can be used to pose queries on Racer (either interactively or with a textual specification), and shows a graphical representation of relations in an A-box. RICE can also deal with multiple T-boxes and associated A-boxes. In particular, it can show instances of a concept and concepts (direct types) of instances.



```
Concepts: |http://potato.cs.man.ac.uk/ontologies/diving#Regulator|
|http://potato.cs.man.ac.uk/ontologies/diving#Medical+Condition|
|http://potato.cs.man.ac.uk/ontologies/diving#Organisation|
|http://potato.cs.man.ac.uk/ontologies/diving#Oxygen+Cleaned|
|http://potato.cs.man.ac.uk/ontologies/diving#Oxygen+Service|
|http://potato.cs.man.ac.uk/ontologies/diving#Oxygen+Toxicity|
|http://potato.cs.man.ac.uk/ontologies/diving#Physiology|
|http://potato.cs.man.ac.uk/ontologies/diving#Procedure|
  |http://potato.cs.man.ac.uk/ontologies/diving#Buddy+Breathing|
  |http://potato.cs.man.ac.uk/ontologies/diving#Buddy+Check|
  |http://potato.cs.man.ac.uk/ontologies/diving#Cardiopulmonary+Resuscitation|
  |http://potato.cs.man.ac.uk/ontologies/diving#Off+Gassing|
  |http://potato.cs.man.ac.uk/ontologies/diving#On+Gassing|
  |http://potato.cs.man.ac.uk/ontologies/diving#Recompression|
  |http://potato.cs.man.ac.uk/ontologies/diving#Safety+Stop|
  |http://potato.cs.man.ac.uk/ontologies/diving#Staged+Decompression|
|http://potato.cs.man.ac.uk/ontologies/diving#Qualification|

ABoxes: DEFAULT
Instances:
gies/diving#AOW|
gies/diving#Advanced+Diver|
gies/diving#Air|
gies/diving#Assistant+Instructo
gies/diving#BSAC+88+Tables|
gies/diving#BSAC|
gies/diving#Club+Diver|
gies/diving#Dive+Leader|
gies/diving#Divemaster|
gies/diving#First+Class+Diver|
gies/diving#Heliox|

Statements and/or concept definitions:
(concept-instances (or |http://potato.cs.man.ac.uk/ontologies/diving#Qualification| (some  |http://potato.cs.man.ac.uk/ontologies/diving#made_by|
|http://potato.cs.man.ac.uk/ontologies/diving#Manufacturer|)))

RACER Replies:
(|http://potato.c
|http://potato.c
|http://potato.c
|http://potato.c
|http://potato.c
|http://potato.c
```

Fig. 3. Using RICE to visualize a RDF document interactively defined with OilEd.

5 Reasoning Beyond OWL: Constraints on Data Types

For various practical reasons OWL also includes so-called data types based on XML-Schema. Data types in XML-Schema are inspired by a storage-oriented characterization of values and are taken from programming languages. For instance, data types encompass `integer`, `short`, `long`, `boolean`, `string` as well as various kinds of specializations for strings.

For an ontology representation language, a semantic characterization for data types might have been more appropriate in our opinion. Thus natural numbers, integers, reals, or complex numbers might have been selected as data types rather than `long` or `short` etc. because for knowledge representation languages the storage format should not be of top-most concern.

Based on XML-Schema in DAML+OIL or OWL it is possible to specify subtypes of, for instance, `integer` by defining a minimum or maximum value [15]. However, OWL does not support so-called constraints between data values. In many practical applications, for instance, linear polynomial inequations with order relations are appropriate. In description logics and databases, these kinds of constraints have a long tradition (see [1, 11]). In the following we will adopt the description logic perspective: concrete domains [2].

Racer supports concrete domain reasoning over natural numbers ($\mathbb{N}$), integers ($\mathbb{Z}$), reals ($\mathbb{R}$), complex numbers ($\mathbb{C}$), and strings. For different sets, different kinds of predicates are supported:

– ℕ: linear inequations with order constraints and integer coefficients
– ℤ: interval constraints
– ℝ: linear inequations with order constraints and rational coefficients
– ℂ: nonlinear multivariate inequations with integer coefficients
– Strings: equality and inequality

For convenience, rational coefficients can be specified in floating point notation. They are automatically transformed into their rational equivalents (e.g., 0.75 is transformed into $\frac{3}{4}$). In the following we will use the names on the left-hand side of the table to refer to the corresponding concrete domains.

The following example uses the concrete domains ℤ and ℝ. For sake of brevity, we use Racer's Lisp syntax [10].

```
(in-tbox family)
(signature
   :atomic-concepts (... teenager)
   :roles (...)
   :attributes ((integer age)))
...
(equivalent teenager (and human (min age 16)))
(equivalent old-teenager (and human (min age 18)))
```

Asking for the children of teenager reveals that `old-teenager` is a `teenager`. A further extensions demonstrates the usage of reals as concrete domain.

```
(signature
   :atomic-concepts (... teenager)
   :roles (...)
   :attributes ((integer age)
                (real temperature-celsius)
                (real temperature-fahrenheit)))
...
(equivalent teenager (and human (min age 16)))
(equivalent old-teenager (and human (min age 18)))
(equivalent human-with-feaver (and human (>= temperature-celsius 38.5))
(equivalent seriously-ill-human (and human (>= temperature-celsius 42.0)))
```

Obviously, Racer determines that the concept `seriously-ill-human` is subsumed by `human-with-fever`. For the Reals, Racer supports linear equations and inequations. Thus, we could add the following statement to the knowledge base in order to ensure the proper relationship between the two attributes `temperature-fahrenheit` and `temperature-celsius`.

```
(implies top (= temperature-fahrenheit
                (+ (* 1.8 temperature-celsius) 32)))
```

If a concept `seriously-ill-human-1` is defined as

```
(equivalent seriously-ill-human-1
            (and human (>= temperature-fahrenheit 107.6)))
```

Racer recognizes the subsumption relationship with `human-with-fever` and the synonym relationship with `seriously-ill-human`.

In an A-box, it is possible to set up constraints between single individuals. This is illustrated with the following examples.

```
(signature
   :atomic-concepts (... teenager)
   :roles (...)
   :attributes (...)
   :individuals (eve doris)
   :objects (temp-eve temp-doris))
...
(constrained eve temp-eve temperature-fahrenheit)
(constrained doris temp-doris temperature-celsius)
(constraints
   (= temp-eve 102.56)
   (= temp-doris 39.5))
```

For instance, this states that the individual `eve` is related via the attribute `temperature-fahrenheit` to the object `temp-eve`. The constraint `(= temp-eve 102.56)` specifies that the object `temp-eve` is equal to 102.56.

Now, asking for the direct types of eve and doris reveals that both individuals are instances of `human-with-fever`. In the following A-box there is an inconsistency since the temperature of 102.56 Fahrenheit is identical with 39.5 Celsius.

```
(constrained eve temp-eve temperature-fahrenheit)
(constrained doris temp-doris temperature-celsius)
(constraints
   (= temp-eve 102.56)
   (= temp-doris 39.5)
   (> temp-eve temp-doris))
```

An additional kind of query is possible for concrete domains: Check if certain concrete domain constraints are entailed by an A-box and a T-box. For instance, in the above-mentioned example, the following query returns true.

```
(constraint-entailed? (= temp-eve temp-doris))
```

6 Conclusion

This paper briefly described Racer and demonstrated that Racer can cooperate with various kinds of ontology editors and visualization tools. Racer can be considered as one of the fastest OWL DL reasoners based on sound and complete algorithms that is currently freely available. It is still unique in its highly optimized reasoning support for A-boxes and constraint-based data types (as demonstrated in the previous sections). Racer also includes optimization techniques supporting the classification of very large knowledge bases (KBs). For instance, a set of KBs could be classified in a few hours [7] that were derived from the Unified Medical Language System (UMLS) and contain up to 200,000 concept introduction axioms (OWL classes) and up to 50,000 hierarchical roles (OWL object properties).

Acknowledgements

We gratefully acknowledge the work of Ronald Cornet, who developed RICE.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook.* Cambridge University Press, 2002. In print.
2. F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Twelfth International Joint Conference on Artificial Intelligence, Darling Harbour, Sydney, Australia, Aug. 24-30, 1991*, pages 452–457, August 1991.
3. F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In D. Hutter and W. Stephan, editors, *Festschrift in honor of Jörg Siekmann.* LNAI. Springer-Verlag, 2003.
4. S. Bechhofer, I. Horrocks, and C. Goble. OilEd: a reason-able ontology editor for the semantic web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna.* LNAI Vol. 2174, Springer-Verlag, 2001.
5. S. Bechhofer, R. Möller, and P. Crowther. The DIG description interface. In *Proc. International Workshop on Description Logics – DL'03*, 2003.
6. D. Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF Schema, http://www.w3.org/tr/2002/wd-rdf-schema-20020430/, 2002.
7. V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-01, August 4-10, 2001, Seattle, Washington, USA*, pages 161–166, August 2001.
8. V. Haarslev and R. Möller. Racer system description. In *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy.*, 2001.
9. V. Haarslev and R. Möller. Optimization stategies for instance retrieval. In *Proc. International Workshop on Description Logics – DL'02*, 2002.
10. V. Haarslev and R. Möller. The Racer user's guide and reference manual, 2003.
11. G. Kuper, L. Libkin, and J. Paredaens (Eds.). *Constraint Databases.* Springer-Verlag, 1998.
12. O. Lassila and R.R. Swick. Resource description framework (RDF) model and syntax specification. recommendation, W3C, february 1999. http://www.w3.org/tr/1999/rec-rdf-syntax-19990222, 1999.
13. R. Möller, R. Cornet, and V. Haarslev. Graphical interfaces for Racer: querying DAML+OIL and RDF documents. In *Proc. International Workshop on Description Logics – DL'03*, 2003.
14. F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference, 2003.
15. F. van Harmelen, P.F. Patel-Schneider, and I. Horrocks (Editors). Reference description of the DAML+OIL (march 2001) ontology markup language, 2001.

DL-workbench: a meta-modeling approach to ontology manipulation

Mikhail KAZAKOV[1, 2] Habib ABDULRAB[2]

[1] Research division, Open Cascade S.A. 91400, Saclay, France
mikhail.kazakov@opencascade.com
[2] PSI laboratoire, INSA de Rouen, BP 8, 76131, Mont Saint Aignan, France
abdulrab@insa-rouen.fr

Nowadays many ontological editors are available. However, several specific requirements forced us to develop a new ontology edition platform which we call DL-workbench. DL-workbench contains three main modules. First module defines a meta-model for description of ontological formalisms. It provides an API that allows management of ontological entities, containers of entities, and many other features that are useful when one needs to use an ontological model within its project. A second module of DL-workbench is a formalism-independent processing module with integrated GUI for edition of elements based on the meta-model. This module uses the meta-model and is implemented as a plug-in to the IBM Eclipse platform. A third module defines the SHIQ description logic formalism using the meta-model. The third module customizes also the user interface (images, etc.). DAML+OIL is used within this module as a persistent format. DL-workbench pays much attention to edition of logical equations and to management of a group of ontologies within a project. DL-workbench allows easy integration of ontological model with other data inside one standalone or distributed application. DL-workbench can be used both as the ontological editor and as an ontology manipulation platform integrated with other tools and environments. This paper describes our motivation for creation of DL-workbench, implemented features and lessons learned from implementation of ontological editor.

1. Introduction

Ontologies have become an increasingly important research topic. This is a result of their usefulness in many application domains (software engineering, databases, medical domain, conceptual modeling, etc.) and the role they will play in the development of emerging Semantic Web activity. It is clear that development and manipulation of ontologies have to be supported by corresponding software tools (annotation tools, editors, reasoners, etc.) and standards (OWL [9], etc.).

Nowadays many application domains are looking for use of ontologies. Each domain, that uses ontologies, sets its own requirements for tools. One of such domains is software integration. The main challenge of software integration is: how to make working together software entities that were not initially created to work with each other. We are currently working on the topic of semi-automated integration of various numerical simulation multi-physics solvers [2]. Here the use of ontologies (as formal description models) helps to share a common view on specification of solvers that come from different vendors. Integration topic is out of the scope of this article and will be published in a separate paper. Some preliminary results are given in [16].

Since we use logical models in our specific domain, we need a formalism that will be used for creation of these models and tools that support the formalism (creation of ontologies and reasoning). Also we need an API that allows us to integrate all these tools with software, specific to the integration domain. Taking into account the research origin of our work, we have formulated a set of requirements for an ontological tool that would be convenient for us (the same requirements arise often within the variety of other domains):

- Full support of at least one of ontological formalisms. Several formalisms are preferred for research purpose. During the work on our research project dedicated to semi-automated integration we experimented with several ontological formalism and at the moment of DL-workbench creation we were not sure, whether description logics are appropriate or not for our research.
- Convenient user interface to work with complex logical expressions (with ability to modify the structure of expressions via the graphical user interface). While trying several ontological editors, we found that most of the ontological editors don't implement expression editors. Those who implement them were not convenient from our point of view.
- Presence of reasoner connection for implemented formalisms. Within our research we have to work with complex logical axioms and ontological models. These require the presence of reasoner both for validation of ontologies and for performing of additional reasoning tasks that are dedicated to semi-automated integration of software components.
- Ability to integrate that tool into a specific domain environment. Our research is dedicated to integration environments that already exist as prerequisite. The next requirement is coming from the same origin.
- Ability to manipulate ontologies and their elements from specific domain environment. We need that in order to merge several ontological formalisms within the same environment. For example the ontological data is merged with specifications of Java interfaces.
- Ability to work with structured "ontological projects" but not with "files". Following our methodology of semi-automated integration we need to work with several separated ontological files at the same time. Managing of these files within the "project"-like environment seems to be an acceptable solution. Most of the people in software engineering who will use our products are familiar with this concept.
- Fast and portable user interface and presence of extension points

Studying the state of the art, we did not find any tool that could comply with our requirements. Thus the decision to create own platform was taken. The name of the platform is DL-workbench (short of Description Logic Workbench, since SHIQ description logic [7] is the main formalism that is used). DL-workbench is published now under open source license (http://www.opencascade.org/dl-workbench).

Semantic Web activity is an important world-wide effort that combines many techniques and touches many application domains. We hope that our experience in creating of software tools for manipulation of ontologies (and DL-workbench itself) will be useful for the semantic web community. This paper describes a meta-model approach that was used for creation of Dl-workbench. First we give the concept of DL-workbench and our motivations. Further, we describe the meta-model kernel and other modules. At the end we share some of lessons learned during creation of this tool.

2. DL-workbench conception

The next list of principles and it constitutes the main concept of DL-workbench:

- DL-workbench is based on a meta-model that is capable to describe the structure of ontological formalisms and data that is used within a specific domain.
- DL-workbench has a modular (plug-in based) architecture with clearly specified dependencies among modules.
- Main processing module of DL-workbench is based only on the meta-model and does not depend on any of specific formalisms. This module implements features such as persistence skeleton, tracking of changes, transactions[1], lifecycle of instances and many others.
- Each generic module provides a set of documented extension points that can be used by other modules / software to customize the platform.
- The work with ontologies is performed using a notion of a project. A project here is a structured set of ontological files (ontological resources) and other domain specific data if needed.
- DL-workbench defines an internal data model and "UI-ready" data model. That allows organization of different views on the same data (i.e. project view, namespace view, taxonomy view)
- DL-workbench supports manipulation/edition of complex expressions and axioms. In many application domains, the complete and "reasoning-able" ontologies require the use of logical expressions and axioms.
- DL-workbench provides the user interface of an ontological editor.

We strongly believe that an ontological manipulation platform shall be based on these principles to be interoperable and useful for researchers, software architects and end users. DL-workbench source code is public and we hope it will be useful for vendors of ontology edition tools.

DL-workbench can be viewed both as a meta-model based platform for creating ontology-manipulation tools and as an ontological editor that supports SHIQ description logic (i.e. the meta-model is used to define SHIQ model). DL-workbench

[1] Transaction support is not implemented in the current version of DL-workbench

uses DAML+OIL[2] [8] as persistent format and Racer [11] as DL reasoner. DL-workbench is implemented as a set of plug-ins to IBM Eclipse platform [10]. Eclipse is an emerging open source environment for creation of project-based tools.

3. DL-workbench structure

Meta-model

The major advantage of the DL-workbench is its meta-model based architecture. It allows easy-to-use definition of entities and relations of an ontological formalism that has to be used within the workbench. The meta-model is implemented as a set of Java interfaces for the convenience of use from programming environments. Java programming is involved for instantiation of the meta-model. This is an explicit design choice. We benefit of existing type checking system and absence of another compiler. That makes the objects instantiated from meta-objects working very fast. The power of Java language can be used for definition of behavioral parts of the meta-model. Moreover most of the people working in integration area are familiar with Java.

A meta-model is a language that is used for description of ontological formalisms by specifying their elements, structure and invariants (invariants have to be satisfied when instances of these elements are created or modified). For example: one needs to work with a simple propositional logic that includes notions of “atomic proposition” and “composite proposition” expressed via logical expression with conjunction, disjunction and negation operands.

Our meta-model is implemented as a light module and is independent from Eclipse or any other tool. The main goal of the module is to achieve the maximal level of independence from specific formalism and to enable implementing generic software features (object lifecycle, transactions, etc.). The module also helps achieving the interoperability among different tools by sharing the same meta-model.

The meta-model is basically intended for specifying structural models. Semantics of the meta-model were inspired by Description Logics [1].We tried to keep the meta-model as simple as possible, but powerful enough for many possible needs.

The main element of meta-model is a meta-concept (`IMetaConcept` Java interface). It represents any typed element with a set of properties. For example: “Expression”, “proposition”, “atomic proposition”, “logical operand” are the instances of meta-concept. Each meta-concept has a meta-name that is represented in a form of java interface. The taxonomy of these Java interfaces defines the taxonomy (i.e. subsumption relationships) of corresponded meta-concept instances. For example: an “atomic proposition” is a “proposition” and an “expression” with atomic propositions is also a “proposition”.

[2] We consider using the OWL language [9] instead of DAML+OIL as soon as OWL parsers will appear on the market.

Another important element of a meta-model is a meta-property (`IMetaProperty` Java interface). A meta-property represents a property that can potentially restrict a meta-concept (similar to description logic semantics of "property"). For example, the "atomic proposition" concept has "name" property; an "expression" concept has "operand", "left part" and "right part" properties. Meta-property also has a meta-name that is represented in the same way as for meta-concepts (i.e. in the form of Java interfaces). Each meta-concept instance may have a set of meta-property instances as its definition. An instance of meta-concept that inherits other meta-concepts also takes all their properties. Generally, the most specific meta-concept is restricted by the transitive closure of all the properties

Any meta-property has the notion of domain restriction, range restriction, inverse property restriction and transitivity. Semantics of all these notions (meta-concept, meta-property, domain, range, etc.) are very close to semantics of corresponded elements from description logics [7].

In addition to these semantics, each meta-property can be augmented with a pre-post processing handler for checking the invariant conditions of given property values. For example: the "name" property of the "proposition" concept has a "String" type. It must be complaint with the URI specification (i.e. no spaces, quotes, etc). An invariant handler that does such check can be written for the "name" property.

The meta-model includes a simple type system. Types are used to define meta-property range restrictions and to type instances of meta-model elements. Type system includes singular types and collection types. A singular type can be primitive (String, Boolean, Integer, Float, and Enumeration) or a meta-concept instance. Collection type is defined by the type of its elements (any other type).

In order to specify some logical model (formalism), meta-concept and meta-property classes must be instantiated. A special meta-model factory is implemented within DL-workbench to facilitate this task. It is worth to mention, that many formalisms can exist at the same time, can share their definitions and can be searched for specific meta-concepts and meta-properties.

Here it's necessary to mention, that three meta-modeling levels exist. `IMetaConcept` interface carries the notion of "meta-concept", its Java instance is a specific instance of the "meta-concept" (model concept) and `IModelInstance` interface specify an instance of the model concept (i.e. model instance). In order to instantiate the formalism, the notion of value is defined (`IValue` interface). Every primitive type and collection has its value object (`IPrimitiveValue`, `IModelCollection`). Every instance of meta-concept within some formalism can have many model instances represented by instances of `IModelInstance` interface.

Each model instance has its type represented by a meta-concept instance and a set of values according to properties of the model concept. For example: a model instance of "expression" meta-concept can be created with three values of their properties:

- "left part": model instance of type : "atomic prop." :"name" = "MARY" : String
- "operand" : model instance of type : "conjunction"
- "right part" : model instance : type "atomic prop." : "name" = "PETER": String

As we can see the instances of the above mentioned simple propositional formalism are expressed in terms of meta-model. The structure of the ontological formalism can be easily traversed by any application. Consequently, an ontological editor that supports only the notion of (concept – property – value) triple can be easily implemented. That is extremely important for specific application domains, where the ontological information must be included within some other application.

The last element of meta-model is a “Container”. Container is a collection of model instances or other containers that supports basic operations of addition, removal, iteration, checking of containment and size request. Creation of dynamic groups of elements is useful when sending information to reasoner, saving sub-ontology, classifying elements, etc.

We use the same meta-model for description of Java interfaces within the software integration domain. The meta-model is generic enough while having rich semantics. Many structural data formalisms can be easily expressed in the presented meta-model.

The meta-model kernel has a small abstract model expressed by means of meta-model. It contains the most useful semantic such as named object, namespace, generic expression and many others. We believe that this model is useful for expressing different formalisms. The DL-workbench meta-model documentation [5] may be consulted for further information.

The meta-model kernel publishes an API that allows defining and manipulating the meta-model instances (models), instantiation of these models, manipulation of instances of models (for example ontological elements) and many other features.

Processing module

A generic model processing module implements manipulation and edition of ontologies. This module is integrated with Eclipse workspace and depends only on the meta-model module. Eclipse framework provides us with the project-oriented workspace. The framework enables transparent connection to many environments of software integration domain (IBM Web Sphere, some UML tools, etc.).

The processing module implements the “view” and “controller” concepts according to the Model-View-Controller paradigm. Here the “model” concept is represented by a formalism that is an instance of the meta-model. “Controller” can be viewer as the UI operations. “View” data model is built on top of meta-model and enables many representations for specified formalism. An ontological project can be viewed by default as:

- a structured set of persisted ontologies (files)
- a set of namespaces (in case if formalism supports namespace)
- each element of some formalism is presented by a tree including properties of corresponded meta-elements and their typed values

The processing module defines all the generic UI operations for lifecycle of model instances independently of the formalism used. Processing module generates user interface controls following the structure of a given model instance. For example: the “expression” meta-concept instance has two properties of type “proposition” and one property of type enumeration. In this case, when a user asks for edition of an instance of “expression”, three groups of controls will be generated independently on the end-user semantics of “expression”. This principle works for any operation within the module. User interface elements are created only once for each type of elements and cashed to reduce unnecessary OS interactions. Each module, dependent on this one, must specify the formalism itself and its UI resources (icons, names, order, etc.). Several concurrent/joint formalisms can be also supported.

The processing module publishes an API for manipulation and configuration of the user interface. It has an UI ready API that encapsulates instances coming from meta-modeling kernel allowing their visual presentation. Further details of implementation, extension points and API can be found in [5].

SHIQ module

SHIQ module implements a model of SHIQDn_ description logic formalism [7] that is based on the meta-model, implements DAML+OIL reader and writer and defines DIG interface [4] connection for the solver. The implementation of SHIQ formalism can be found in [5] and is not repeated in this paper. The module implements an additional view – taxonomical view: Selecting of any SHIQ "concept" or "object property" concepts within the Eclipse workspace causes the dynamic building of the taxonomy tree for this concept/property. The view is shown by default on the right edge of the Eclipse window within the DL-workbench perspective. The taxonomy is dynamically rebuilt using "subClassOf" and "sameAs" properties of the SHIQ "concept"/"properties". SHIQ module (as any formalism specific module) also specifies a set of Eclipse extensions: association of *.daml files with the plug-in, association of specific icons with menu items and others UI features. Integration with Eclipse is described in DL-workbench documentation [5].

As one can observe, the inclusion of specific formalism and customization of the user interface represent a very small part of the ontological editor code. SHIQ logic formalism was described via the meta-model within one working day. GUI customization was done within one week. The F-logic formalism was prototyped as an example during several hours using the same meta-model and benefits all the editor features (F-logic module is not published with DL-workbench due to incompleteness of GUI customization and absence of persistent format connection.).

Racer reasoner [11] is used via DIG interface. We choose Racer due to its support of ABox reasoning. However, the DL-workbench itself uses a reasoner only for satisfability and subsumption checks, thus FaCT [12] or any other DIG-complaint reasoner can be used. DAML+OIL reading support is done with the help of Jena DAML parser [17]. DAML+OIL writing was done via Xerces XML parser. Since DAML+OIL is a superset of RDFS, RDFS files can be also read by DL-workbench.

More details on the user interface and use of DL-workbench as ontological editor can be taken from [5]. The product itself and its source code can be downloaded following link in [5].

4. Lessons learned

In this section we'd like to indicate some positive experience and observations that were received during the creation of DL-workbench and use of ontologies for a specific application domain.

Axioms and logical expression are extremely important for creating of complete and reasoning-ready ontologies. However, it's extremely difficult to have a convenient user interface (GUI) for their edition. We've implemented our own GUI of expression editor; however we strongly believe that some deep research must be conducted on ergonomics of expression editor. It can be something between text input with dynamic compilation and GUI-based editor that chooses elements from lists. Current implementation of the expression editor is based on the same principle as other editors of DL-workbench: editor of "Expression" entity with a set of properties such as "left part", "right part" and "operand". Expression entities can be nested (i.e. "left part" can be also an expression). The rules of expression building are defined in the form of invariants and pre-post conditions in a generic model from meta-modeling module. We came from considerations that the given structure of expressions is common for most of the possible ontological formalisms. End user has always the possibility not to use the presented model or replace it according to its needs. The correctness of expressions is also supported by invariants and all structurally inconsistent equations are highlighted for the user. The processing module recognize "equation" as a special type and provide light edition mechanism using drag and drop and dynamic management of lists of possible elements. Same mechanism tries to assure that nested equations are not lost when containing structure is modified within the top level editor (loss happens sometimes in OilEd equation editor).

From our point of view, working with ontologies must follow the project-oriented paradigm. It's hard to imagine a real industrial ontology that is saved in one file and has no references to other files. The use of URI as a physical location of imported ontology is not always suited for industrial use due to possible unavailability of some URI at some time. Here the notion of project as a complete set of needed ontological resources can facilitate manipulation of ontologies. It can clearly separate a physical structure of files from logical structure of ontologies (i.e. namespaces, taxonomies). However it is worth to mention that needs of Semantic Web can be different.

Presence of meta-model for implementation of ontological formalism and connection with other data structures is very important. Above we said many things about these benefits.

The ability to have many different views (by namespaces, by taxonomies, by files, graphical view[3] and many others) on the same ontological structure helps a lot in many real cases.

Support of several formalisms and several GUI views is extremely important since it allows creation of different views for different groups of users on the same domain data and its ontological semantics. For example the same ontology may be presented by two formalisms with different expressivity to different groups of people.

We found it useful to introduce several macro-semantics into the SHIQ editor that are computed from basic SHIQ semantics[4]:

- XOR, IMPLIES and EQUALS logical operators can be easily introduced into any expression. These operands are easily convertible into AND/OR/NOT sequences and vice versa. That adds more high level semantics to the user.

[3] Graphical view is not provided in current version of the DL-workbench
[4] These macro-commands are excluded from the first open source version of DL-workbench

- In the same way: “class or equivalents” or “class of disjoints” elements can be defined on top of basic SHIQ axioms. When writing/reading to DAML+OIL corresponded transformations are performed.

Easiness of integration of ontological model / ontological edition user interface is crucial when ontologies are used within some application domains. There is an evident help from modern frameworks such as Eclipse and use of component technologies. Especially this is important, when research projects with sharp time frames are conducted.

By developing DL-workbench we have achieved all the requirements that were described at the beginning of this paper. Our current research for software integration is based on DL-workbench. We use described concepts for creation of extensions of DL-workbench that facilitate our experiments with integration of numerical solvers and creation of “good enough” ontologies verified by reasoner.

5. Other editors and APIs

Many ideas of user interface were inspired by OilEd [6] ontological editor. OilEd is the first editor that implements most of the features of SHIQ description logic, reasoner connection and expression edition. That was an ontological editor we used before creating DL-workbench. However, despite of all its benefits, some elements of user interface, such as choice among “subClassOf” and “sameClassAs”, semantic of some axioms and some others elements are not always clear for the end user. We tried to resolve these issues in DL-workbench. OilEd is an open source project, but OilEd API seemed to us difficult to integrate with other tools. Presence of meta-model level within the DL-workbench gives more flexibility and ease of use together with other tools.

WebODE project [3] is an ontological workbench that provides various ontological services. The project has a highly flexible architecture and provides many viewpoints on data. The meta-modeling approach chosen by DL-workbench allows on-fly changing of ontological formalism and easy integration with non-ontological data structures. In addition, DL-workbench is an open source project.

Protege [18] ontological editor has a convenient plug-ins API for its extension. However our intention was to integrate an ontological editor as a plug-in into software development tools but not vice-versa.

KAON API and a set of related KAON tools [14] define a distributed ontology manipulation infrastructure that is based on client-server architecture and provides many useful features. KAON has a hard coded API for its ontological formalism, that is mostly RDFS based and doesn’t support extended semantics of equations nor very expressive description logics (such as SHIQ). We found the OI-modeler ontological editor of KAON to be difficult for the end user.

Many other ontological editors are present nowadays. Due to the absence of space in the paper we can’t compare many tools. [13] is a good survey of ontological tools. The deliverable 1.3 [19] of OntoWeb project gives a comprehensive comparison of tools.

6. Conclusions and future plans

We have presented DL-workbench, both an Eclipse-based ontological editor for SHIQ logic and a meta-model based platform for manipulation of ontologies in conjunction with other tools. We've shown the benefits to use meta-model for creating of ontology-based products especially when working within specific application domains.

Today the DL-workbench is a research prototype and it lacks the stability that is needed for industrial development of ontologies. It lacks the functionality of merging and aligning of ontologies and more extensive support of reasoners features on the user interface level. All of that is planned to be improved soon. The user interface ergonomics and usability study is required.

We use DL-workbench for development of our domain specific extensions and integration with other tools. That assures the constant evolution and support of the DL-workbench. In the future we plan to introduce a transaction mechanism with undo-redo operations, merging/alignment of ontologies, graphical representation of ontological information and make many other improvements.

References

1. F. Baader et all, "*The Description Logic Handbook: theory, implementation and applications*", Cambridge University Press, 2003 ISBN 0-521-78176-0
2. The SALOME project, Online: http://www.opencascade.org/salome
3. WebODE project, Online : http://delicias.dia.fi.upm.es/webODE
4. S. Bechhofer, "The DIG description logic interface: DIG/1.0", 2002, Online: http://www.fh-wedel.de/~mo/racer/interface1.0.pdf
5. DL-workbench project web site. Online: http://www.opencascade.org/dl-workbench
6. S. Bechhofer et all, "*OilEd: a Reason-able Ontology Editor for the Semantic Web*", Springfied-Verlag, LNCS, 2001
7. I. Horrocks, lU. Sattler, and S. Tobies. "*Reasoning with individuals for the description logic SHIQ"*. LNAI number 1831 pp. 482-496. Springer-Verlag, 2000
8. DAML+OIL language, Online: http://www.w3.org/TR/daml+oil-reference
9. OWL language, Online: http://www.w3.org/TR/owl-absyn
10. IBM Eclipse 2.1 platform, project page, Online: http://www.eclipse.org
11. Racer reasoner. http://www.fh-wedel.de/~mo/racer
12. FaCT reasoner, http://www.cs.man.ac.uk/~horrocks/FaCT
13. M. Denny, "*Table 1. Ontology editor survey results*", 2002, Online: http://www.xml.com/2002/11/06/Ontology_Editor_Survey.html
14. KAON API, Online: http://km.aifb.uni-karlsruhe.de/kaon/Members/rvo/kaon_api
15. OMG MOF repository specification, Online: http://www.omg.org/technology/documents/formal/mof.htm
16. M. Kazakov, H. Abdulrab, E. Babkin, "*Intelligent integration of distributed components: Ontology Fusion approach*", In proceedings of CIMCA 2003 conference, 2003, ISBN 1-740-88069-2
17. Jena DAML and RDF parser, Online: http://www.hpl.hp.com/semweb/index.html
18. Prot•g• environment, Online: http://protege.standord.edu
19. OntoWeb project, "Deliverable 1.3 report", Online: http://www.ontoweb.org

OntoTrack: Fast Browsing and Easy Editing of Large Ontologies

Thorsten Liebig[1] and Olaf Noppens[1]

Dept. of Artificial Intelligence
University of Ulm
D-89069 Ulm
{liebig|olaf.noppens}@informatik.uni-ulm.de

Abstract. OntoTrack is a new browsing and editing "in-one-view" ontology authoring tool. It combines a sophisticated graphical layout with mouse enabled editing features optimized for efficient navigation and manipulation of large ontologies. The system is based on SpaceTree [PGB02] and implemented in Java2D. OntoTrack provides animated expansion and de-expansion of class descendants, zooming, paning and uses elaborated layout techniques like click-able miniature branches or selective detail views. At the same time OntoTrack allows for quite a number of editing features using mouse-over anchor buttons and graphical selections without switching into a special editing layout. In addition, every single editing step is synchronized with an external reasoner in order to provide instant feedback about relevant modeling consequences.

1 Introduction

The availability of adequate tools for end users is a pivotal element in order to push Semantic Web techniques from academia to commercial environments. Simple, flexible, and intuitive user interfaces play an important role within this context. In contrast to current tool evaluations which concentrate mainly on language specific issues (e. g. language conformity) and technical criteria (e. g. turn around ability for interoperability) [AS02] we will focus on adequate visualization, navigation and simple editing of large ontologies in the remainder of this paper.

Currently, many ontology editors use two functionally disjunct interfaces for either editing or browsing ontologies. Editing interfaces are commonly based on vertical expand and contract lists representing the class hierarchy. When selecting a particular class in the list one can inspect and manipulate its corresponding definition using predefined forms in an additional display area. Our experiences with expand and contract style interfaces identified a number of conceptual drawbacks:

– The number of visible classes is limited by the screen height. Even middle sized ontologies very likely require scrolling after some level expansions.

– The larger the ontology the harder it will get to identify the inheritance path from a particular class up to the root of the ontology. This is due to the fact of exclusively two level states. An ontology level is either completely expanded or contracted and is not allowed to display a selection of context relevant classes.
– Depending on the branching factor of an ontology a list representation makes it difficult to compare two different expansion paths concerning level depth or common ancestors.
– Because of the tree based nature of expansion lists multiple inheritance is difficult to represent in general. Multiple ancestors of a class are usually displayed with help of an auxiliary display area. Inversely, this class will appear as "cloned" class in the list of descendants of every super class. This, however, will result in a proportional growth of redundant classes with the number of multiple inheritance statements.
– When defining a class one commonly needs to access and select other classes. This temporally requires additional expand and contract style selection lists for a class hierarchy already on screen.

An extreme example for which the list representation will be inherently unsuitable is the task of showing the complete inheritance path of a class in a large ontology having multiple ancestors.

In order to better support those tasks most tools incorporate an additional graphical browsing interface using tree like, tree map, ven or hyperbolic layout techniques more or less suitable for navigating large ontologies. However, those interfaces do not allow for substantial editing and are designed as view-only plugins in most cases.

Our novel ontology editor, called *OntoTrack*, combines hierarchical layout technologies with context sensitive zooming features and mouse enabled editing abilities optimized for navigation and manipulation of large ontologies. OntoTrack is based on the linked tree diagram approach of SpaceTree [PGB02] which dynamically zooms and lays out tree branches to best fit the available screen space. OntoTrack's ontology layout is driven by an animated "expansion on user demand" strategy making use of elaborated minimization techniques for alternative inheritance paths or descendants. At the same time OntoTrack allows for quite a number of editing features from mouse-over anchor buttons to context sensitive choose lists without switching into a special editing layout.

The remainder of this paper is organized as follows. In the next section we present OntoTrack our new graphical authoring tool for ontologies. In particular, we explain OntoTrack's browsing, editing, and searching abilities as well as its inference features via link-up to an external reasoner. In section 3 we describe the current implementation status and discuss current and future work. Section 4 contains preliminary benchmarking results concerning to some qualitative navigation criteria. We will end with a short summary and some notes about possible enhancements.

2 A New Graphical Authoring Tool For Ontologies

2.1 Browsing Features

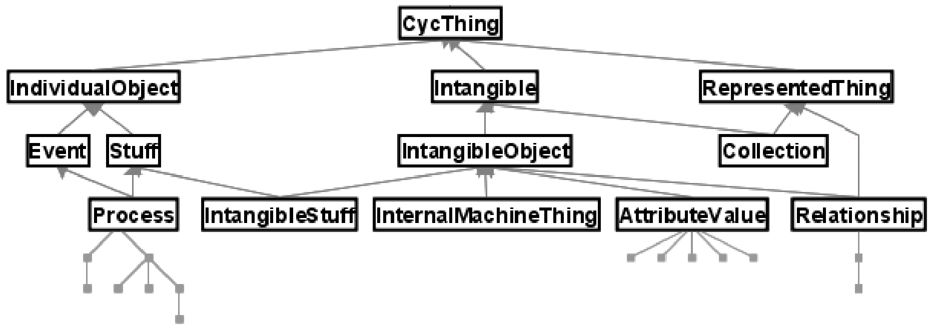


Fig. 1. Partially expanded ontology in top-down layout showing miniature tree thumbnails summarizing not expanded sub-branches.

OntoTrack aims at integrating optimized layout techniques for hierarchies with graphical editing features. Currently, the system is based on SpaceTree [PGB02] an interactive tree browser with dynamic rescaling of branches, optimized camera movement, and preview icons for non expanded sub-branches. Within SpaceTree the expansion of a new tree level is animated and may result in trimming of branches of previous levels when needed. SpaceTree allows for changing the overall orientation of the layout and for explicit de-expansion blocking of user selected branches.

Our attempt was to adapt SpaceTree for browsing and editing ontologies by extending its inheritance centered layout algorithm in a first step. The primary structuring element of ontologies and trees is the inheritance relation. Consequently, SpaceTree's as well as OntoTrack's layout algorithm dynamically adapt their graphs in order to be able to display the complete inheritance path either in a top-down or left-right orientation. As an option, the path from the last expanded class to the root will be outlined. Depth, width and the number of descendants of not expanded sub-branches are symbolized by triangles of varying length, width, and shading or as an iconified branch in order to provide information about deeper levels. In addition, the whole ontology layout can continously be zoomed or paned simply by mouse-down movements. Figure 1 shows an OntoTrack screen capture of an ontology[1] after some level expansions. Here, a classical top-down orientation and a miniature tree thumbnail style has been chosen. In comparison, figure 2 shows the same ontology in a less expanded state using a left-to-right orientation and a triangle thumbnail style. Here, the

[1] Showing the top-level classes in an early version of Cyc.

inheritance path for the last expanded class "IndividualObject" is outlined by a darker node background.

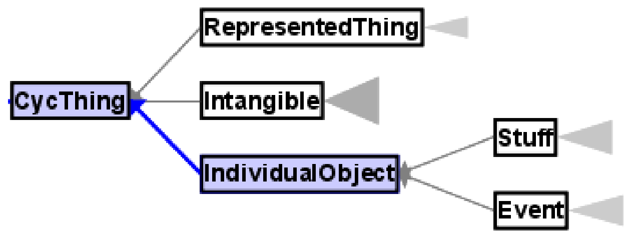


Fig. 2. Ontology of figure 1 in left-right layout and triangle thumbnails.

In the case of expanding a level containing classes having multiple ancestors in currently not expanded branches, those ancestors are drawn as click-able icons. As an example, figure 3 shows the ontology of figure 2 after expansion of class Stuff via middle mouse click. Both descendants of Stuff have multiple ancestors. A further ancestor of Process is an already expanded class Event. In contrast, one ancestor of IntangibleStuff (namely IntangibleObject as can be seen in figure 1) is within the currently not expanded sub-branch of class Intangible and therefore drawn as a click-able icon.

When moving over an iconified ancestor with the mouse pointer a tool-tip message with the corresponding class name appears. Clicking on such an ancestor icon results in an expansion of this class. This strategy guarantees that all ancestors of all expanded classes are displayed either expanded or abbreviated as click-able icons. Having all inheritance paths visible up to the root helps a user to keep orientated concerning to the primal structuring principle of ontologies.

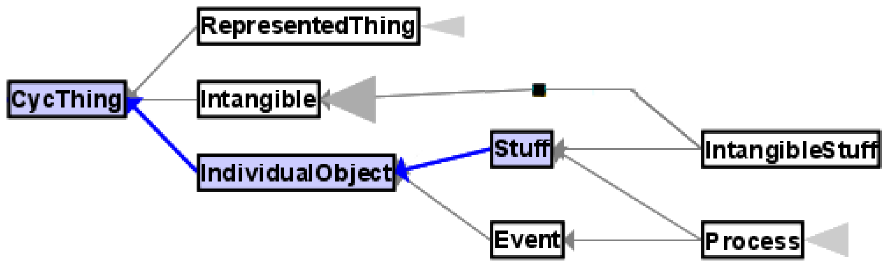


Fig. 3. Ontology of figure 2 after expansion of class Stuff.

2.2 Editing Features

As mentioned before OntoTrack is a browsing and editing in-one-view authoring tool. This allows to re-use already available navigation principles for the task of building and manipulating ontology definitions. The most primitive manipulation feature consists of the direct editable class name field of every class node. Beyond that, OntoTrack's click-and-drop editing features are enabled by switching into the "anchor button" mode. Within this mode anchor buttons appear when moving the mouse over editable entities. Figure 4 shows the anchor buttons of a class IndividualObject displayed in top-down orientation (in left-right orientation the button layout is rotated 90° anti-clockwise). The triangle symbol on top of the class box represents the superclass relationship. With a click on this button one can specify this class to be an descendant of another class selectable with a click on that class. A new sub class can be created with a click onto the bottom triangle. In correspondence with the RDFS and OWL specification the semantics of multiple subclass statements for one class is that of a conjunction in OntoTrack.

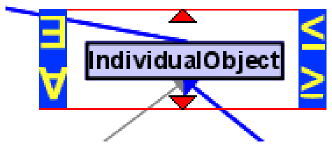


Fig. 4. Anchor buttons of a class in top-down layout.

In addition, OntoTrack offers further editing functions while in its "detailed view" mode. The detailed view mode is activated or deactivated for each class separately using the mouse-wheel up- resp. down-wards while being over the class with the mouse pointer. When activated, OntoTrack uses a slightly adapted UML style class diagram syntax. In contrast to the UML specification our class diagram is divided into two (instead of three) compartments. The top compartment contains the name of the class. The bottom compartment contains a list of property restrictions of this class. In case of OWL Lite each row of this list contains (implicit conjuncted) one existential or universal quantification or unqualified cardinality restrictions displayed in abstract Description Logic (DL) syntax (see [Baa03] for the abstract DL terminology). Figure 5 shows a class with one minimal and one exact cardinality restriction. An existing restriction can be deleted by clicking on the red dot on the right side of the corresponding row. A new restriction is added to a class by using the green dot at the bottom of the class box (see figure 5). The cells of each row are editable via choose lists. An unqualified cardinality restriction provides three choose lists, one for the cardinality operator ($\geq, \leq, =$) one for the value (0 or 1 in OWL Lite) and one for the currently available properties. Quantifications also require three choose lists

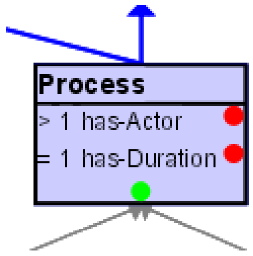


Fig. 5. Class in detailed view mode.

(one for the quantifier ∃ or ∀, one for the property, and one for the qualifying class). Additional editing features like switching between complete and partial definitions are accessible via a right mouse button context menu. As an alternative short-cut we plan to add click-and-drop quantifier and cardinality symbols for specifying properties statements between classes as shown in figure 4 in the near future.

2.3 Inference Feedback

OntoTrack is equipped with an interface to an DL reasoner called RACER [HM01]. All changes after each editing step (e. g. list selection, subclass statement) are send to the RACER system via the TCP-based client interface JRacer. RACER will then make all modeling consequences explicitly available for OntoTrack. Of special interest within our ontology layout is the subsumption relationship which may implicitly be influenced by an editing step. As soon as RACER recognizes a change in the class hierarchy OntoTrack updates the corresponding graphical representation (showing only direct subsumers/subsumees of each class). Those updates are also animated in order not to confuse the user with a new hierarchy layout in one step. Other graphical inference services (which are special cases of the subsumption relationship in fact) cover unsatisfiable class definitions or equivalence between different classes. In OntoTrack an unsatisfiable class will be drawn in red and equivalent classes are outlined with a colored background.

2.4 Searching

OntoTrack also adapts SpaceTree's search features. When looking for a specific class name, even in the selection phase during editing, one can use a string based ontology search. When start typing a search string all matching classes or sub-branch icons are highlighted. Each additional character or deletion in the search string directly results in an updated highlighting of matching parts of the ontology. OntoTrack currently supports three matching mode: exact match, substring match from string beginning, and full substring match. As an option, the user can then fan out the ontology by expansion of all currently matching classes via one button click.

3 Implementation Status and Current Work

Our ontology authoring tool OntoTrack is still under development. The features described in section 2 are those of the first implementation phase. Some may change in future versions if they don't prove to be useful. It is our considered opinion that performance and scalability are very important properties of user friendly tools and a key for user acceptance. We therefore have chosen Piccolo as our graphical library. Piccolo is an optimized subset of Jazz [BMG00], a fast zoomable interface toolkit based on Java2D.

A first prototype of OntoTrack has been implemented by extending SpaceTree's layout algorithm, which itself uses the Piccolo libraries. Within this version all mentioned browsing features of subsection 2.1 are implemented in full detail. Some of the editing features of subsection 2.2 however are still under development (click-and-drop qualifiers and cardinality statements).

Current work is focused on refining and optimizing the layout algorithms for ancestor thumbnails. Miniature tree layouts for ancestors with multiple inheritance turned out to be difficult in general. Imagine the problem of thumb placing for a short expanded inheritance path together with a long alternative path via a thumbnail miniature tree (or vice versa). Inheritance links between thumbnail classes and already expanded classes are another factor of complexity for placing and cross minimizing layout algorithms. As an additional constraint we want to re-arrange the layout of expanded classes in each possible expansion step as less as possible. Therefore, OntoTrack implements a local optimization layout algorithm triggered by the class the user currently wants to expand.

OntoTrack's file import as well as export uses the RDF parser Jena2[McB01]. Jena2's internal ontology model for classes and properties also serves as OntoTrack's central representation model. Currently, OntoTrack is able to read in and write out OWL Lite ontologies. However, properties as well as global property constraints (domain and range statements) are not editable in OntoTrack at the moment.

Conceptually, we plan to cover a notable fragment of OWL Lite's language constructs while adopting UML's class diagram representation. In a first step we concentrated on OWL Lite's class axioms and restriction statements (see section 2.3.1.1. and 2.3.1.2. of OWL Abstract Syntax and Semantics document [PSHH03]). Next, we want to extend the editor with a parallel representation of properties and property hierarchies. Our goal is an mixed graphical representation based on the hierarchical class layout described above together with editable property edges in combination or as alternative to the list representation of OntoTrack's detailed view mode.[2]

[2] The ezOWL plugin [OC03] for Protégé is an example of a likewise mixed class and property representation to some extend.

4 Preliminary Evaluation

It was not the goal of our preliminary evaluation tests to determine an overall ranking of different ontology editors. Other tools like Protégé [GMF$^+$03], OntoEdit [SSA01] or OilEd [BHGS01] are obviously in a more sophisticated state of development and in some cases tailored to different tasks or users. Our aim was to evaluate our graphical browsing and editing interface against other user interfaces with respect to certain navigation criteria.

First we compared the maximum number of classes to display for a given screen size. Using a screen size of 1280×1024 we counted a number of 50 to 60 displayable classes in expand and contract style ontology browsers using full screen hight (here, the screen width has no effect on the maximum of displayable classes). Using a comparable font size in OntoTrack we were able to expand more than 100 classes using full screen mode.

In contrast, the length for an inheritance path for a branch with classes having a name with an average length of 12 characters has a depth of 13 levels in OntoTrack. In an expand and contract style interface the same number of level expansions approximately take up 30 % of the screen width.

However, in order to have some qualitative results concerning average navigation or editing performance a controlled experiment has to be conducted. A set of experiments comparing three tree-based browsing tools (MS Explorer, a Hyperbolic tree browser, and SpaceTree) showed some performance advantages for the SpaceTree approach concerning tasks like first-time node finding, listing all ancestors of a node, or differentiate between branches with varying numbers of nodes [PGB02]. These results may serve as an indicator with respect to navigation and editing performance of OntoTrack in comparison with expand and contract style interfaces.

5 Summary and Outlook

Expand and contract style interfaces for ontologies inherently have substantial drawbacks concerning search and navigation speed, user orientation, and editing flexibility in our opinion. Our new authoring tool for ontologies combines an animated graphical layout with mouse enabled editing features within one view. We are still in an early development phase, but first experiences with our SpaceTree [PGB02] based prototype are encouraging. We therefore see OntoTrack as an easy-to-use interactive ontology editor especially for non-experienced users and even for large ontologies.

Current work focuses on finalizing the layout algorithm, and further editing features. We also plan to extend OntoTrack's search facility for regular expression matching as well as for restriction expressions. The link-up to the RACER reasoner is also a subject of optimization. Currently, OntoTrack needs to query the reasoner for all possible consequences of each user change in order to update its internal representation model. Here, an event triggered notification model on reasoner side would significantly speed up this process. In addition, an adequate explanation module is needed in order to distinguish between 'direct' consequences (e. g. an unsatisfiable class because of an user manipulation) and follow-up consequences (e. g. the consequences of an unsatisfiable class with respect to other classes). In order to become a competitive application basic features like undo, print, or various exports into other ontology languages have to be implemented in future versions of OntoTrack.

We plan to cover ontology languages with an expressivity at least comparable to that of OWL Lite. Complex class descriptions like nested restrictions or general inclusion axioms may need additional graphical features in a next evolution step. A graphical representation as well as editing interfaces for disjoint classes, coverings and instances are also on our working agenda. An graphical UML representation for some of those have already been discussed in [BKK$^+$01] and may serve as starting point for our application.

References

[AS02] Jürgen Angele and York Sure. Whitepaper: Evaluation of Ontology-based Tools. Technical report, OntoWeb Deliverable 1.3, 2002.

[Baa03] Franz Baader. *The Description Logic Handbook*, chapter Appendix 1: Description Logic Terminology. Cambridge University Press, 2003.

[BHGS01] Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *Proc. of the German conference on Artificial Intelligence, KI2001*, pages 396 – 408. Springer Verlag, LNAI Vol. 2174, September 2001.

[BKK$^+$01] Kenneth Baclawski, Mieczyslaw K. Kokar, Paul A. Kogut, Lewis Hart, Jeffrey Smith, William S. Holmes, Jerzy Letkowski, and Michael L Aronson. Extending ULM to Support Ontology Engineering for the Semantic Web. In *Proceedings of the Fourth International Conference on UML (UML 2001)*, number 2185 in LNCS, pages 342 – 360, Toronto, Canada, October 2001. Springer Verlag.

[BMG00] Ben Bederson, Jon Meyer, and Lance Good. Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java. *UIST 2000, ACM Symposium on User Interface Software and Technology, CHI Letters*, 2(2):171 – 180, 2000.

[GMF$^+$03] John Gennari, Mark Musen, Ray Fergerson, William Grosso, Monica Crubézy, Henrik Eriksson, Natalya Fridman Noy, and Samson Tu. The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. *International Journal of Human Computer Studies*, 58(6):737 – 758, June 2003.

[HM01] Volker Haarslev and Ralf Möller. RACER System Description. In *Proc. of the International Joint Conference on Automated Reasoning, IJCAR'2001*, Siena, Italy, June 2001.

[McB01] Brian McBride. Jena: Implementing the RDF Model and Syntax Specification. In *Proc. of the Second International Workshop on the Semantic Web, SemWeb'2001*, Hong Kong, China, 2001.

[OC03] Sooyoung Oh and Moonyoung Chung. ezOWL plugin for Protégé. `http://iweb.etri.re.kr/ezowl/`, 2003.

[PGB02] Catherine Plaisant, Jesse Grosjean, and Benjamin B. Bederson. SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation. In *Proc. of the IEEE Symposium on Information Visualization, INFOVIS 2002*, pages 57 – 64, Boston, USA, October 2002.

[PSHH03] Peter Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Working Draft, March 2003.

[SSA01] York Sure, Steffen Stab, and Jürgen Angele. OntoEdit: Guiding Ontology Development by Methodology and Inferencing. In *Proc. of the Confederated International Conferences CoopIS, DOA and ODBASE 2002*, pages 1205 – 1222. Springer Verlag, LNCS Vol. 2519, October 2001.

TooCoM : a Tool to Operationalize an Ontology with the Conceptual Graphs Model

Frédéric Fürst, Michel Leclère, Francky Trichet

Institut de Recherche en Informatique de Nantes
2 rue de la Houssinière - BP 92208
44322 Nantes France
`{furst,leclere,trichet}@irin.univ-nantes.fr`

Abstract. This article deals with the operational use of a domain ontology integrated into a Knowledge-Based System (KBS). It presents TooCoM, a tool dedicated to (1) the definition of ontologies with the Entity-Relationship paradigm and (2) the operationalization of ontologies in the context of the Conceptual Graphs model. TooCoM provides functionalities for specifying an operational scenario of use of the ontology which is under construction, for transcribing this ontology into the corresponding operational form and for using this operational form in an embedded inference engine.

Keywords: Ontology, Conceptual Graphs, Knowledge-Based Systems, Operationalization.

1 INTRODUCTION

Most of works which aims at developing tools for building an ontology focuses on the edition of the conceptual vocabulary, i.e. the terminological level. For instance, Protégé allows the knowledge engineer to build a hierarchy of concepts and to specify predefined properties of the concepts through the Frame model [11]. OntoEdit (renamed Kaon) is also based on the Frame paradigm. As Protégé, it focuses on the structuration of a set of concepts and on the specification of predefined properties of these concepts [20].

None of the tools listed within the OntoWeb project [6] aims at editing, in an intuitive and graphical way, the axioms of a domain. However, in our opinion, axioms are the main operational ressource of an ontology since they constrain the use of the conceptual vocabulary. Consequently, they are the only means to specify the semantics of a domain. For instance, in Protégé, the knowledge engineer must known the Protégé Axiom Language to specify the constraints and/or the rules of the domain. In OntoEdit, the specification of a non-predefined axiom must be done by using a logical formula.

TooCoM is a tool which adresses this problem. It allows the knowledge engineer (1) to specify the conceptual vocabulary of the domain by using the Entity-Relationship paradigm, (2) to specify the axioms of the domain in a graphical way and (3) to easily make these axioms operational in order to perform reasoning in the context of the Conceptual Graphs model[1]. For this last point, TooCoM can be considered as an innovative tool in the sense that it allows the knowledge engineer to follow reasoning processes in a graphical way. This aspect is very important because, in our opinion, this facilitates the appropriation and the control of the semantics which is associated to the ontology under construction. In other words, providing functionalities dedicated to a graphical appropriation of the implications of all the axioms (rules and constraints) of a domain makes the understanding (and therefore the refinement) of the semantics of a domain more easy.

As WebODE implements the METHONTOLOGY methodology to build an ontology [1], TooCoM implements original guidelines to specify axioms at the conceptual level and to specify the operational use of the ontology which determinates the operational form of the axioms.

From a technical point of view, TooCoM is based on CoGITaNT, a framework which offers capabilities to represent and manipulate Conceptual Graphs [10]. TooCoM has been tested in the context of the GINA project (Interactive and Natural Geometry) related to CAD (Computer-Aided Design) [13]. In this experiment, our tool has been used to build and to automatically operationalize an ontology of geometry [9].

The rest of this paper is structured as follows. Section 2 presents how building an ontology with TooCoM, in particular how specifying the conceptual vocabulary and the axioms. Section 3 first introduces the process we advocate to operationalize an ontology and then shows the application of this process in the context of the Conceptual Graphs model and its implementation in TooCoM. Finally, section 4 introduces a discussion about the innovative aspects of TooCoM in comparison with existing tools.

2 DEFINING AN ONTOLOGY WITH THE ENTITY-RELATIONSHIP PARADIGM

Defining an ontology with the Entity-Relationship (E/R) paradigm mainly consists in (1) specifying of the conceptual vocabulary of the domain which is considered and (2) specifying the semantics of the conceptual vocabulary through axioms.

2.1 The specification of the conceptual vocabulary

As implied by the Gruber's definition, (*« an ontology is a formal, explicit specification of a shared conceptualization »* [12]), the building of an ontology is based on a conceptualization, which is a conceptual description of the knowledge covered by the ontology. This description consists of a conceptual vocabulary which, in the context of the E/R paradigm, contains a set of concept types and a set of relation types which can both be structured by using subsomption links.

TooCoM allows the knowledge engineer to define such hierarchies, both for concept types and for relation types. Figure 1 shows an extract of the hierarchy of concept types

[1] Operationalizing knowledge consists in representing it with an operational language, according to an operational goal. An operational language is a formal language (i.e. a language having a syntax and formal semantics) which provides inference mechanisms allowing one to reason from its representations. An operational goal is specified by a *scenario of use* (cf. section 3.1).

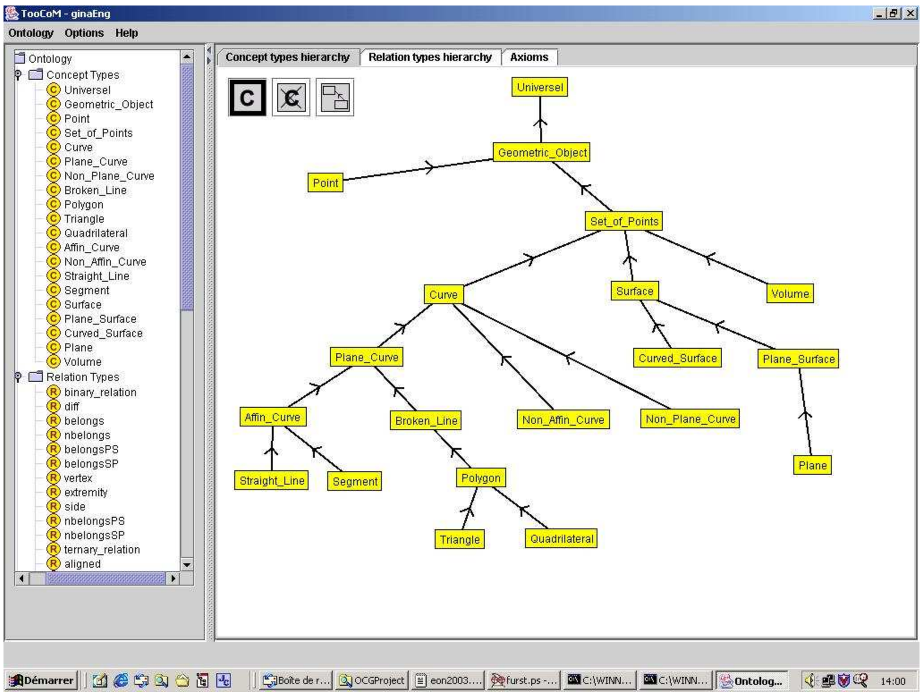


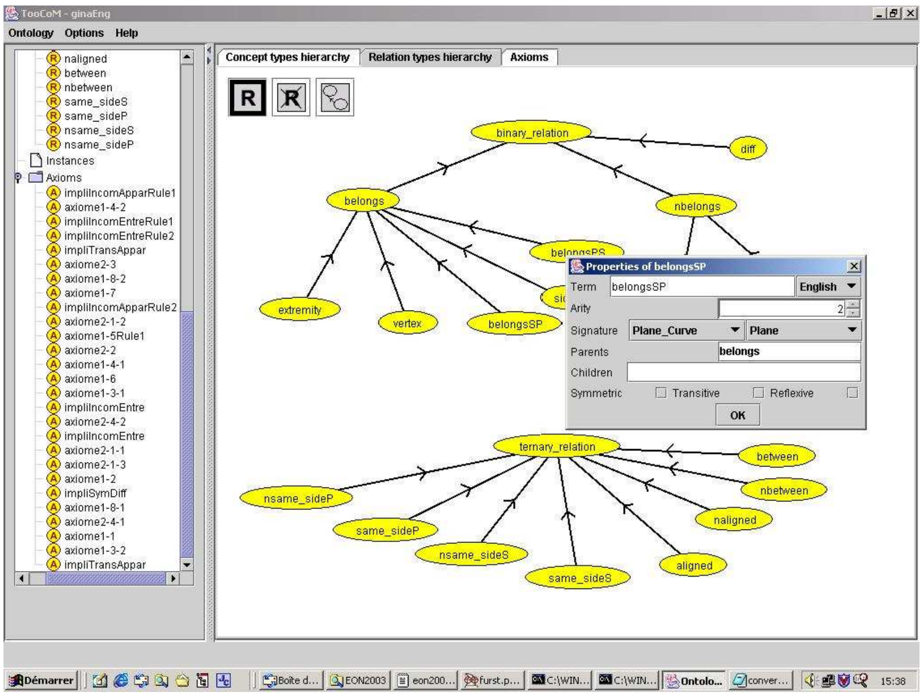
Fig. 1. A hierarchy of concept types in TooCoM. An arrow represents a subsomption link between a concept type and his parent concept type (« *a Triangle is-a Polygon* »).

which has been defined for the GINA project (i.e. an ontology of geometry defined according to Hilbert's book « *Grunlagen der Geometrie* »). Figure 2 shows the hierarchy of relation types.

2.2 The specification of the axioms

Axioms represent the intension of concept types and relation types and, generally speaking, knowledge which is not strictly terminological [19]. Axioms are specific to ontologies and, in our opinion, allow us to distinguish an ontology from a thesaurus. Thesaurus are only based on terminological representations and can be compared to light weight ontologies, whereas heavy weight ontologies contain the whole semantics of a domain [18]. Axioms specify the way the terminological primitives must be manipulated. Two types of axioms can be distinguished :

- the axioms that represent common and well-defined properties of concept types or relation types ;
- the axioms that represent properties specific to the domain .



Properties of belongsSP

Term: belongsSP — English
Arity: 2
Signature: Plane_Curve — Plane
Parents: belongs
Children:
Symmetric ☐ Transitive ☐ Reflexive ☐
OK

Fig. 2. A hierarchy of relation types in TooCoM. The property box of the *belongsSP* relation type is open. Such a box shows the signature, the parents, the children and the algebraic properties of a relation type. For instance the *belongsSP* relationship can only be stated between a *Plane_Curve* and a *Plane*, it has the *belongs* relation type as parent and no child and bears any algebraic property.

The common properties, that we call *axiom schemata*, can correspond to:

– algebraic properties such as symmetry, reflexivity, transitivity;
– the *is-a* link between two concept types or two relation types (subsomption property);
– the signature or the cardinalities of a relation type;
– the exclusivity or the incompatibility between two concept types or two relation types (the incompatibility between two primitives P_1 and P_2 is formalized by $\neg(P_1 \wedge P_2)$, the exclusivity is formalized by $\neg P_1 \Rightarrow P_2$).

Classical axiom schemata can be specified by simply indicated the property of the relation types in the tool box (cf. figure 2), i.e. without creating a new axiom by using the *Axioms* panel. If an additional property of relation type (symmetry, transitivity or reflexivity) is specified, the corresponding axiom is automatically created and added to the ontology.

However, an axiom does not necessarily correspond to a schema. For instance, figure 3 shows the axiom 1.2 of Hilbert's axiomatics. This axiom, which expresses a property of identity between a *Straight_line* and a couple of *Points* does not correspond to a classical axiom schema and must be build in the axiom panel.

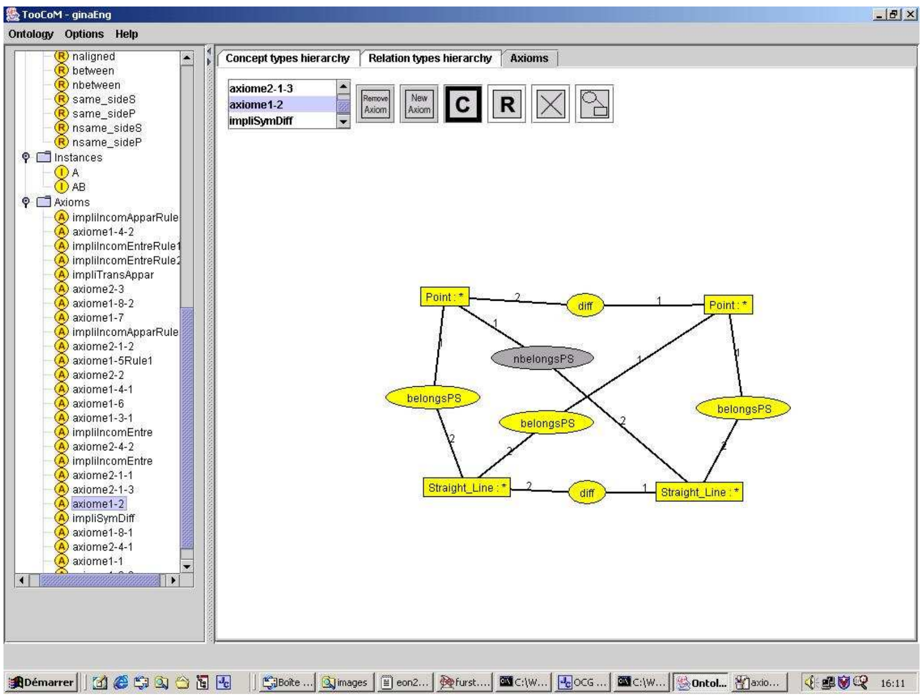


Fig. 3. Representation of an axiom in TooCoM. The yellow (bright) concepts and relationships represent the hypothesis part of the axiom and the gray (dark) concepts and relationships represent the conclusion part. Semantics of this axiom is as follows: *given two different points and two different straight lines, if one of these points belongs to the two lines, and if the other belongs to one the lines, it does not belong to the second line.*

In TooCoM, the subsomption links and the signatures of the relation types are the only properties that are embedded into the modeling paradigm underlying our tool, and they do not have to be expressed by axioms. All other properties of the conceptual primitives have to be specified as axioms via the definition of predefined axiom schemata in hierarchies of concept or relation types, or via the whole creation of an axiom in the *Axioms* panel.

An axiom is composed of an hypothesis part and a conclusion part, respectively represented by a *conceptual graph*[2]. A conceptual graph is a bipartite graph composed of concept vertices (representing objects of the domain) and relationship vertices (describing relationships between objects). Each vertex of a conceptual graph is labeled. A

[2] The Conceptual Graphs model, first introduced by Sowa [17], is a knowledge representation model which belongs to the semantic networks. An extension of this model, the SG family [2], presented in section 3.2, extends the model with reasoning primitives, rules and constraints.

concept vertex is labeled with the concept type from which the represented object is an instance. To identify the represented object, one can possibly add an individual marker. In that case, the vertex is called an individual concept. In other case, one adds to the concept type a star which denotes the generic marker (i.e. the identity of this concept is not defined). Such a vertex is called a generic concept. A relation vertex is simply labeled by a relation type specifying the nature of the link between the neighbouring concepts.

But this representation of axioms does not specify their operational semantics, in the sense that it does not specify the way the axioms will be used in an operational application. Because this operational semantics depends on the operational goal of the KBS, it can not be included in an ontology, which must be independent from any operational goal. Thus specifying this semantics conducts to an *operational ontology*, through an *operationalization* process.

3 OPERATIONALIZING AN ONTOLOGY WITH TooCoM

An ontology is only a conceptual representation of a domain, independently of any operational applications. To use an ontology in a KBS, it is necessary to transcribe the conceptual representation into a form in accordance with the way the KBS will be used. This form must be an operational form, in the sense that the knowledge representation model must offer operational mechanisms, such as inference mechanisms, in order to allow the manipulations to which the KBS is dedicated. For instance, to perform automatic reasoning, the operational formalism must allow the representation of derivation rules and the effective application of these rules on a set of facts. Thus, the use of an ontology in a KBS requires an *operationalization* process, that consists in transcribing the ontology in an operational formalism, in accordance with the operational use of the KBS.

3.1 The scenarii of use and the operationalization of axioms

The operationalization of an ontology is only conceivable for a well defined operational use, characterized by a precise *scenario of use* [5]. A scenario of use is the description of the purposes for which knowledge will be manipulated in the system. Defining a scenario of use mainly consists in describing the way the axioms will be used in the system, because the operational representation of terminological knowledge does not depend on the different contexts of application. Indeed the representation of a concept or a relation type is the same in the case of a system dedicated to knowledge validation or in the case of a system built to produce new facts from a knowledge base. Only the operational representations of the axioms are specific to the goal of the application.

We consider that an axiom can be used to validate knowledge in relation to the ontology or to produce new facts from a base. For instance, the axiom 1.6 of Hilbert *« If two points A and B of a straight line d belong to a plane α, then all the points of d belong to α »* can be used either to deduct the membership of points to a plane, or to indicate that a situation is not in accordance with the semantics of geometry, such as

« there are two points that belong to both a straight line and a plane and a point of the straight line which does not belong to the plane ».

Moreover, an axiom can be used when the user of the system asks for it, or it can be applied automatically by the system everywhere it is possible. The first application is called *explicit*, the second *implicit*. For instance, the axiom 1.3.1 of Hilbert *« On a straight line, there are at least two points »* can be implicitly used if the user is not supposed to apply this axiom before considering points on a straight line or, on the contrary, can be explicitly used if he is supposed to resort to the axiom for considering such points, for instance for educational purposes.

So, operationalizing an ontology requires, for each axiom, the choice of a *context of use* which specifies the purpose for which the axiom will be used and how it will be applied in the system. The different contexts of use we have identified are:

– The **inferential and explicit** context of use: the user applies the axiom by himself on a fact base to produce new facts;

– The **inferential and implicit** context of use: the axiom is applied by the system on a fact base to produce new facts;

– The **validation and explicit** context of use: the user applies the axiom by himself to check that a fact base is in accordance with the semantics of a domain;

– The **validation and implicit** context of use: the axiom is applied by the system to verify that a fact base is in accordance with the semantics of a domain.

A *scenario of use* consists in a set of contexts of use choosen for each axiom of the ontology. Generally speaking, the operational form of an ontology includes inferential mechanisms and validation mechanisms. These mechanisms are required for the automatic (or semi-automatic) manipulation of knowledge. For instance, a scenario dedicated to a computer-aided teaching application allows the user to apply knowledge to deduce new facts or to check his work. Such a scenario comprises automatic inferences and validation processes, in accordance with the level of the user.

Figure 4 presents the general inference cycle through which the axioms are applied in a KBS. First the user can add facts to the fact base, then he can apply an axiom choosen between the inferential and explicit ones. Then the system applies all the inferential implicit axioms in order to sature the fact base with implicit knowledge. Finally, a validation step, which can be partially leaded by the user, permits to detect « semantical inconsistencies » in the fact base.

Two particular scenarii can be distinguished: the pure validation scenario, where the operational ontology is used to check a fact base according to the semantics of a domain (all axioms are operationalized in a validation context of use), and the inferential and implicit scenario, where the operational ontology is used to automatically produce new knowledge (all axioms are operationalized in an implicit context of use). To define the scenario of use of an ontology, the context of use of each axiom must be specified. This context constrains the operational form of the axiom. But, of course, the choice of the operational knowledge representation language also constrains this form.

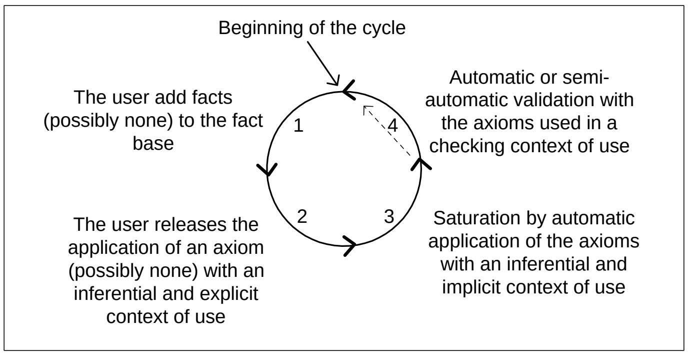

Beginning of the cycle
The user add facts (possibly none) to the fact base
Automatic or semi-automatic validation with the axioms used in a checking context of use
1
4
2
3
The user releases the application of an axiom (possibly none) with an inferential and explicit context of use
Saturation by automatic application of the axioms with an inferential and implicit context of use

Fig. 4. The inference cycle dedicated to the use of an operational ontology.

3.2 The operationalization of the axioms with the Conceptual Graphs model in TooCoM

TooCoM is based on an extension of the Conceptual Graphs model (CGs). The CGs model is an operational knowledge representation language which provides conceptual primitive representations through concepts and relationships between these concepts [17]. The subsomption property and the signature of relationships are integrated in the model. The other axioms, that express the way the primitives must be manipulated, can be represented with three types of reasoning primitives, that have been added as an extension of the model, the SG family [2]:

– The **positive constraints**, with an hypothesis part and a conclusion part, of which the semantics is: if the hypothesis part is present, then the conclusion part must be present (otherwise the constraint is broken);
– The **negative constraints**, with an hypothesis part and a conclusion part, of which the semantics is: if the hypothesis part is present, then the conclusion part must be absent (otherwise the constraint is broken);
– The **rules**, with an hypothesis part and a conclusion part, of which the semantics is: if the hypothesis part is present, then the conclusion part can be produced.

A rule can be implicitly used by the system (i.e. applied everywhere the hypothesis of the rule is present) or explicitly applied by the user (on a given fact in the knowledge base). A negative or positive constraint can be automatically used by the system (i.e. checked everywhere in the knowledge base) or explicitly applied by the user.

In order to allow the automatic operationalization of ontologies in TooCoM, we have defined operationalization mechanisms for each form of axiom. For instance, an axiom can have the following form:

$$\forall x_1, ..., x_n \; H \Rightarrow \exists y_1, ..., y_m \; r_1(..) \wedge .. \wedge r_p(..) \qquad (1)$$

where r_i are relationships between the x_i and/or y_j variables and H a conjonction of predicats which express concepts or relations.

The different operational forms of such an axiom, depending on the context of use, are:

- Inferential and implicit context of use: the axiom is operationalized by an implicit rule which corresponds to the the logical formula $\forall x_1, ..., x_n\ H \Rightarrow \exists y_1, ..., y_m\ r_1(..) \wedge .. \wedge r_p(..)$;
- Inferential and explicit context of use: the axiom is operationalized by an explicit rule which corresponds to the logical formula $\forall x_1, ..., x_n\ H \Rightarrow \exists y_1, ..., y_m\ r_1(..) \wedge .. \wedge r_p(..)$ and p negative constraints which correspond to the statement $\forall x_1, ..., x_n\ H\ (\bigwedge r_i(..))_{i=1..p, i \neq j}$, it can not exist $r'_j(..)$, $j = 1..p$, where r'_j is exclusive with r_j in the ontology[3]. If any relationship exclusive with r_j exists in the ontology, the corresponding constraint is replaced by q negative constraints which correspond to the statement $\forall x_1, ..., x_n\ H\ (\bigwedge r_i(..))_{i=1..p, i \neq j}$, it can not exist $r'_{j_k}(..)$, $k = 1..q$, where r'_{j_k} are all incompatibles with r_j ;
- Validation and implicit (respectively explicit) context of use: the axiom is operationalized by p negative and implicit (respectively explicit) constraints $\forall x_1, ..., x_n\ H\ (\bigwedge r_i(..))_{i=1..p, i \neq j} \Rightarrow r'_j(..)$, $j = 1..p$, where r'_j is exclusive with r_j in the ontology. If any relationship exclusive with r_j exists in the ontology, the corresponding constraint is replaced by q negative constraints which correspond to the statement $\forall x_1, ..., x_n\ H\ (\bigwedge r_i(..))_{i=1..p, i \neq j}$, it can not exist $r'_{j_k}(..)$, $k = 1..q$, where r'_{j_k} are all incompatibles with r_j .

In TooCoM, the user can build an operational ontology by specifying the context of use of each axiom of the ontology. According to this context, each axiom is automatically transcribed into an appropriate form (i.e. a rule, a constraint, a rule and a set of constraints or a set of constraints). Then, the operational ontology, which includes the conceptual primitives and the axioms in an operational form, can be exploited by the TooCoM inference engine which implements the reasoning cycle presented in figure 4.

3.3 The use of an operational ontology in TooCoM

TooCoM provides an inference engine based on the manipulation of conceptual graphs. This inference engine uses the CoGITaNT framework which allows to compare graphs and to apply CG rules through a graph projection operator [10]. By using this inference engine, the knowledge engineer can test the ontology under construction by applying the operational ontology to different situations. For instance he can state a fact represented by a graph and runs the engine over this fact. During the explicit inferential phase, he can choose the axiom he wants to apply and where he wants to apply it. The result of the reasoning process is displayed in real-time in the interface and the user can check if the resulting fact is correct in relation to the result which is intended. Again, as shown in figures 5 and 6, we argue in favor of a graphical semantics. These figures present the running panel of the inference engine.

If the user has a set of competency questions, he can check it with the inference engine. Moreover, the system can indicate exactly what axiom creates an inconsistency or

[3] The incompatibility between two primitives P_1 and P_2 is formalized by $\neg(P_1 \wedge P_2)$, the exclusivity is formalized by $\neg P_1 \Rightarrow P_2$.

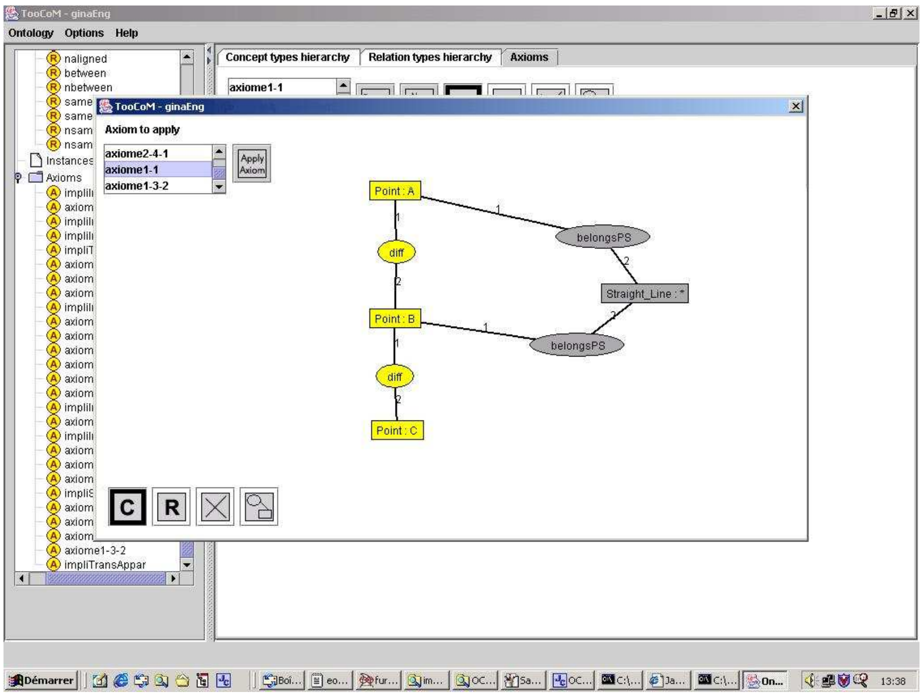


Fig. 5. A step of an inference cycle. The user has build a graph with three points A, B and C where A is different from B and B different from C (the graph appears in bright color). He selects the axiom 1.1 (*given two different points, it exists a straight line to which belong these two points*) which can be applied on the points A and B, or B and C (the conclusion part of the axiom appears in dark color). The system suggests to the user different projections on which the axiom can be applied. By using the keyboard arrows, the user can examine the different projections and apply the explicit axiom where he wants. In this example, the user applies the axiom on the points A and B (cf. figure 6 for the next step).

what axiom is lacking to answer the question. For instance, in the domain of geometry, we have use TooCoM to produce an operational form of the ontology appropriated to the automatic theorem proof checking [9]. In this case, all the Hilbert's axioms have been operationalized through an explicit and inferential context of use and the other axioms (e.g. the exclusivity between relation types) have been operationalized through an implicit and inferential context of use. By testing the proof of some theorems, we have discover some missings, which correspond to implicit knowledge not stated by Hilbert in his book, but really used in the proofs [9].

The building of different kinds of KBS is possible as far as the system can use the general reasoning cycle. In the context of geometry, we can adapt the scenario of use to automatically generate a module of an Intelligent Tutoring System which will use some axioms to validate the student's assertions and others to complete these assertions, whereas the student will use the explicit axioms to prove a theorem or to build a geometric figure.

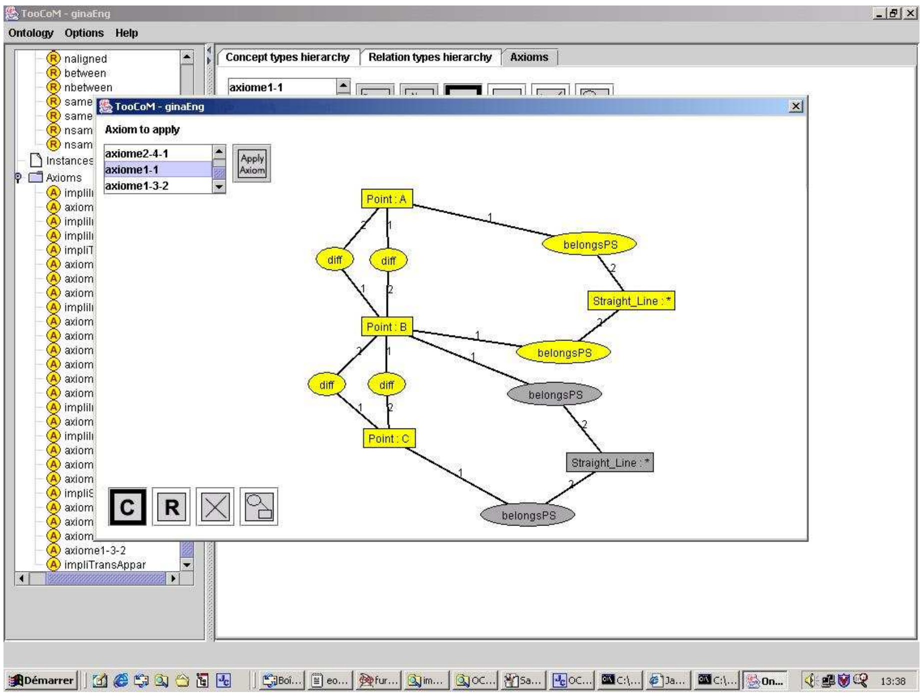


Fig. 6. After applying the axiom 1.1, the system automatically applies the implicit rules, and deduces the difference between the points from the symmetry property of the *diff* (difference) relation type. The user can then apply another axiom, for instance the axiom 1.1.

4 RELATED WORK

The first aspect that differentiates TooCoM from its related tools is that it is based on the Entity-Relation paradigm to structure an ontology, whereas most of other tools dedicated to the building of ontology, like OILEd [3], Protégé [11] or OntoEdit [20], are based on the Frame paradigm. Indeed, TooCoM is based on the Conceptual Graphs model which provides both a conceptual paradigm used to structure the terminological level of the ontology and reasoning mechanisms based on graph homomorphism in keeping with the first order logic.

Then, most of existing tools provides a textual mode to specify conceptual vocabulary and axioms. For instance, in OntoEdit, the specification of a non-predefined type of axiom requires the use of the F-Logic syntax [20]. But some of them allows the knowledge engineer to build ontologies in a graphical way: WebOnto provides a graphical interface for the edition of the conceptual vocabulary but not for the edition of axioms [7]. The graph based paradigm used in TooCoM is more intuitive than a textual one and it allows the knowledge engineer to specify both the terminological knowledge and all kind of axioms in a graphical interface, without knowing a textual axiom language.

We think that a graphic visualization of the inferences carried out is a significant factor which, on the one hand, facilitates the appropriation of a formal system and, on the other hand, allows the expert to validate the adopted model on its own (without reinterpretation of the implemented reasonings by a logician)[4]. The use of a graphical langage to build an ontology, which is a knowledge model, is coherent with the use of graphical languages, as UML, to build modelization in the programming domain.

The second, and most important, innovative aspect of TooCoM, is to allow the representation and the operationalization of all kinds of axioms. As fast as the use of ontology is growing, it becomes necessary to represent more and more complex properties of the concepts. For instance, the specification of OWL [16] includes new properties, as intersection of concept classes or algebraic properties, that do not appear in the RDFS specification. In our opinion, a complete ontology representation language must allow to represent any axiom, and not only predefined axioms. This allows the knowledge engineer to define properties that are not included in the language. For instance, in the domain of geometry, a lot of properties expressed through mathematical axioms can not be related to well defined properties, like algebraic properties.

An other advantage of the operational representation of axioms is the possibility to use ontologies for reasoning. This aspect becomes more and more important for the applications of the Semantic Web [8]: the Web services will use ontologies to reason and this requires the representation of axioms and not only the representation of terminological primitives organized in hierarchies. For instance, the RuleML language [4] is dedicated to the representation of rules and constraints in order to allow deduction, rewriting, and further inferential-transformational tasks. But the operational representation of axioms is conditioned by their operational uses. So, building operational representation of axioms requires an operationalization process through which these representations are produced according to contexts of use.

The representation of all kinds of axioms and their use in an inference engine through an original operationalization process allows to perform the original goal of Protégé, that is the interactive building of a KBS [11]. Moreover, in TooCoM, it is possible to automatically make the ontology operational and to manipulate it at a conceptual level. The context of use of each axiom can be specified and the KBS appropriated to the application which is intended can be automatically generated. As in many tools, this mechanism permits a constraint checking of the ontology. But it also allows the knowledge engineer to easily check the completeness of the ontology, by submitting competency questions to the inference engine.

At this moment, the ontologies can be stored in the BCGCT format [15], which is peculiar to the CoGITaNT framework, or in the CGXML format. These formats allow to represent the terminological primitives of a domain, the subsomption links between these primitives, the instances of concepts types, and axioms in rule form. We plan to add a module in order to allow the storage and the loading of ontologies in other common ontology languages like RDFS or OWL, as far as the expressivity of these langages allows us to represent all axioms.

[4] The validation can then be considered as a simple study of graphical explanations of the reasonings that have been performed by the system.

5 CONCLUSION

TooCoM allows a knowledge engineer to build ontologies within the Entity-Relationship paradigm, and to specify both the terminological knowledge of a domain and the semantics of this domain through axioms. The main characteristics of TooCoM is the possibility to define all kinds of axioms and to generate different operational ontologies from the specification of scenarii of use. So, thanks to a graphical semantics, TooCoM facilitates the appropriation of a global understanding of the semantics of the domain mainly defined by the axioms.

The operationalization mechanism provided by TooCoM permits, via the definition of an operational scenario, to produce operational ontologies. These operational ontologies can be used to validate the ontology itself, by submiting competency questions to the inference engine. This corresponds to a knowledge level prototyping approach [14]. For instance, the experiment we have done in the domain of geometry has lead us to modify our ontology after that the proof of a theorem failed.

The operationalization guideline implemented in TooCoM must be extended to other formalisms than the CGs model. In particular, the use of a combination of OWL, to represent the terminological knowledge, and RuleML, to represent axioms, is planned. It will permits to build operational ontologies that can be used on the Web.

References

1. J. Arpirez, O. Corcho, M. Fernandez-Lopez, and A. Gomez-Perez. Weboe: a workbench for ontological engineering. In *Proceedings of the first International Conference on Knowledge Capture (K-CAP'2001), Victoria, Canada*, 2001.
2. J.F. Baget and M.L. Mugnier. The sg family: Extensions of simple conceptual graphs. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'2001)*, pages 205–210, 2001.
3. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. Oiled: a reason-able ontology editor for the semantic web. In *Proceedings of KI2001, Joint German/Austria Conference on Artificial Intelligence*, volume 2174, pages 396–408. Springer Verlag LNAI, 2001.
4. H. Boley, S. Tabet, and G. Wagner. Design rationale of ruleml : a markup language for semantic web rules. In *Proceedings of the Semantic Web Working Symposium (SWWS'2001)*, 2001.
5. J. Bouaud, B. Bachimont, J. Charlet, and P. Zweigenbaum. Methodological principles for structuring an ontology. In ACM Press, editor, *Proceedings of IJCAI'95 Workshop: Basic Ontological Issues in Knowledge sharing*, 1995.
6. OntoWeb consortium (coordinated by Asuncion Gomez Perez). A survey on ontology tools. technical report IST-2000-29243, IST, 2002.
7. J. Domingue. Tadzebao and webonto: Discussing, browsing and editing ontologies on the web. In *Proceedings of the Eleventh Knowledge Acquisition Workshop (KAW'98)*, 1998.
8. D. Fensel and C. Bussler. Semantic web enabled web services. In *Proceedings of International Semantic Web Conference (ISWC'2002)*, volume 2342, pages 1–2. Springer-Verlag LNCS, 2002.
9. F. Fürst, M. Leclère, and F. Trichet. Contribution of the ontology engineering to mathematical knowledge management. *Annals of Mathematics and Artificial Intelligence, Kluwer Academic Publishers*, (38):65–89, 2003.

10. D. Genest and E. Salvat. A platform allowing typed nested graphs : how cogito became cogitant. In *Proceedings of the International Conference on Conceptual Structures (ICCS'98)*, volume 1453, pages 154–161. Springer-Verlag LNAI, 1998.
11. J.H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubezy, H. Eriksson, N.F. Noy, and S.W. Tu. The evolution of protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58:89–123, 2003.
12. T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
13. O. Lhomme, P. Kuzo, and P. Macé. Desargues, a constraint-based system for 3d projective geometry. *Geometric Constraint Solving and Applications*, ISBN:3-540-64416-4, 1998.
14. A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.
15. CoGITaNT Home Page. http://cogitant.sourceforge.net/docs/index.html.
16. Ontology Web Language Home Page. http://www.w3.org/tr/2002/wd-owl-guide-20021104/.
17. J. Sowa. *Conceptual Structures : information processing in mind and machine*. Addison-Wesley, 1984.
18. S. Staab. An extensible approach for modeling ontologies in rdf(s). In *presentation at the ECDL2000 Workshop on the Semantic Web, http://www.ics.forth.gr/isl/SemWeb/PPT/Staab.ppt*, 2000.
19. S. Staab and A. Maedche. Axioms are objects too: Ontology engineering beyong the modeling of concepts and relations. Research report 399, Institute AIFB, Karlsruhe, 2000.
20. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. Ontoedit: colllaborative ontology development for the semantic web. In *Proceedings of the International Semantic Web Conference*, volume 2342, pages 221–235. Springer-Verlag LNCS, 2002.

A Domain Ontology Engineering Tool with General Ontologies and Text Corpus

Naoki SUGIURA[1], Masaki KUREMATSU[2], Naoki FUKUTA[1], Noriaki IZUMI[3], and Takahira YAMAGUCHI[1]

[1] Department of Computer Science, Shizuoka University
3-5-1 Johoku, Hamamatsu, Shizuoka, JAPAN
{sugiura, fukuta, yamaguti}@ks.cs.inf.shizuoka.ac.jp
http://panda.cs.inf.shizuoka.ac.jp

[2] Iwate Prefectual University, 152-52 Takizawa-aza-sugo, Takizawa, Iwate, JAPAN
kure@soft.iwate-pu.ac.jp

[3] CARC, National Institute of AIST, 2-41-6 Tokyo Waterfront, Aomi, Koto-ku, Tokyo, JAPAN
niz@ni.aist.go.jp

Abstract. In this paper, we describe how to exploit a machine-readable dictionary (MRD) and domain-specific text corpus in supporting the construction of domain ontologies that specify taxonomic and non-taxonomic relationships among given domain concepts. In building taxonomic relationships (hierarchical structure) of domain concepts, some of them can be extracted from an MRD with marked subtrees that may be modified by a domain expert, using matching result analysis and trimmed result analysis. We construct concept specification templates (non-taxonomic relationships of domain concepts) that come from pairs of concepts extracted from text corpus, using WordSpace and an association rule algorithm. Through case studies, we make sure that our system can work to support the process of constructing domain ontologies.

1 Introduction

Although ontologies have been very popular in many application areas (e.g. Semantic Web), we still face the problem of high cost associated with building up them manually. In particular, since domain ontologies have the meaning specific to application domains, human experts have to make huge efforts for constructing them entirely by hand. In order to reduce the costs, automatic or semi-automatic methods have been proposed using knowledge engineering techniques and natural language processing ones[1]. However, most of these environments facilitate the construction of only a hierarchically-structured set of domain concepts, in other words, taxonomic conceptual relationships. For example, DODDLE[2] developed by us uses a machine-readable dictionary (MRD) to support a user to construct concept hierarchy only.

In this paper, we extend DODDLE into DODDLE II that constructs both taxonomic and non-taxonomic conceptual relationships, exploiting WordNet[3] and domain-specific text corpus with the automatic analysis of lexical co-occurrence statistics and an association rule algorithm[4]. Furthermore, we evaluate how DODDLE II works in the field of law, CISG (the Contracts for the International Sale of Goods), and in the field of business, xCBL (XML Common Business Library). The empirical results show us that DODDLE II can support a domain expert in constructing domain ontologies.

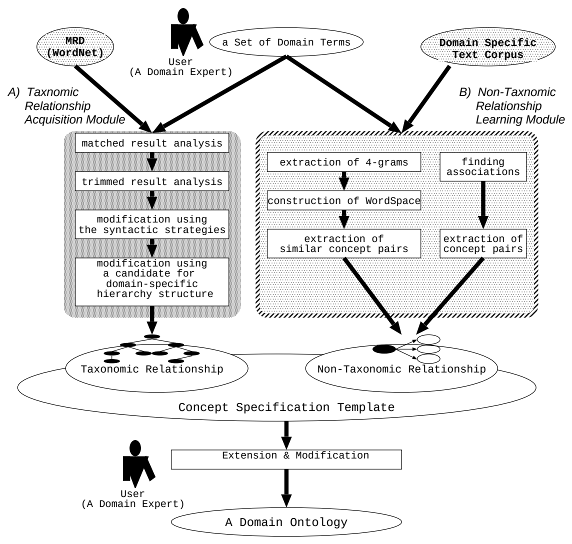

MRD (WordNet)
User (A Domain Expert)
a Set of Domain Terms
Domain Specific Text Corpus
A) Taxnomic Relationship Acquisition Module
B) Non-Taxnomic Relationship Learning Module
matched result analysis
trimmed result analysis
modification using the syntactic strategies
modification using a candidate for domain-specific hierarchy structure
extraction of 4-grams
construction of WordSpace
extraction of similar concept pairs
finding associations
extraction of concept pairs
Taxonomic Relationship
Non-Taxonomic Relationship
Concept Specification Template
Extension & Modification
User (A Domain Expert)
A Domain Ontology

Fig. 1. DODDLE II overview

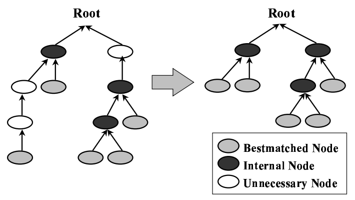


Fig. 2. Trimming Process

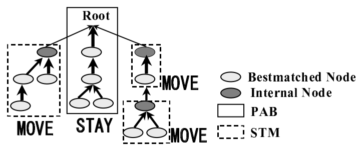


Fig. 3. Matched Result Analysis

2 DODDLE II: A Domain Ontology Rapid Development Environment

2.1 Overview

Figure 1 shows the overview of DODDLE II, "*a Domain Ontology rapiD DeveLopment Environment*". DODDLE II tries to construct a domain ontology from a set of domain terms given by a human expert using the following two components: Taxonomic Relationship Acquisition module (TRA module) using WordNet and Non-Taxonomic Relationship Learning module (NTRL module) using text corpus. WordNet is a existing MRD used in some systems.
A) TRA module tries to support a user in constructing taxonomic relationship (concept hierarchy) using Word Net.
B) NTRL module extracts the pairs of terms that should be related by some relationships from text corpus, analyzing lexical co-occurrence statistics, based on WordSpace[5] and an associate rule algorithm.

We can build concept specification templates by putting together taxonomic and non-taxonomic relationships for the input domain terms. The relationships should be identified in the interaction with a human expert.

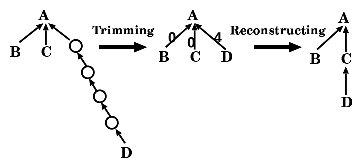


Fig. 4. Trimmed Result Analysis

2.2 Taxonomic Relationship Acquisition

First of all, TRA module does "spell match" between input domain terms and WordNet. The "spell match" links these terms to WordNet. Thus the initial model from the "spell match" results is a hierarchically structured set of all the nodes on the path from these terms to the root of WordNet. However, the initial model has unnecessary internal terms (nodes) and they do not contribute to keep topological relationships among matched nodes, such as parent-child relationship and sibling relationship. So we get a trimmed model by trimming the unnecessary internal nodes from the initial model (see Fig.2). After getting the trimmed model, TRA module refines it by interaction with a domain expert, using Matched result analysis (see Fig.3) and Trimmed result analysis (see Fig.4). TRA module divides the trimmed model into a PAB (a PAth including only Best spell-matched nodes) and an STM (a Subtree that includes best spell-matched nodes and other nodes and so can be Moved) based on the distribution of best-matched nodes. A PAB is a path that includes only best-matched nodes that have the senses good for given domain specificity. Because all nodes have already been adjusted to the domain in PABs, PABs can stay in the trimmed model. An STM is such a subtree that an internal node is a root and the subordinates are only best-matched nodes. Because internal nodes have not been confirmed to have the senses good for a given domain, an STM can be moved in the trimmed model.

In order to refine the trimmed model, DODDLE II can use trimmed result analysis. Taking some sibling nodes with the same parent node, there may be big differences about the number of trimmed nodes between them and the parent node. When such a big difference comes up on a subtree in the trimmed model, it is better to change the structure of it. DODDLE II asks a human expert whether the subtree should be reconstructed. Based on the empirical analysis, the subtrees with two or more differences may be reconstructed.

Finally, DODDLE II completes taxonomic relationships of the input domain terms manually from the user.

2.3 Non-Taxonomic Relationship Learning

NTRL module almost comes from WordSpace, which derives lexical co-occurrence information from a large text corpus and is a multi-dimension vector space (a set of vectors). The inner product between two word vectors works as the measure of their semantic relatedness. When two words' inner product is beyond some upper bound, there are possibilities to have some non-taxonomic relationship between them. NTRL module also uses an association rule algorithm to find associations between terms in text corpus. When an association rule between terms exceeds user-defined thresholds, there are possibilities to have some non-taxonomic relationships between them.

Construction of WordSpace WordSpace is constructed as shown in Fig.5.

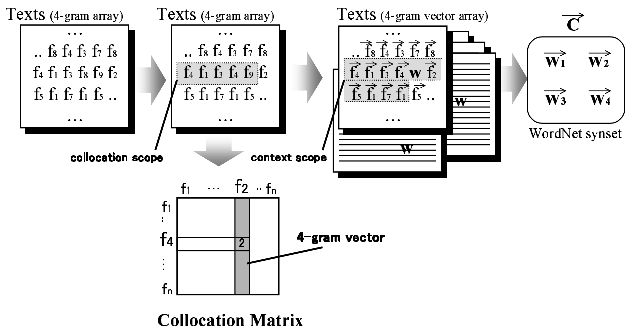


Fig. 5. Construction Flow of WordSpace

1. Extraction of high-frequency 4-grams Since letter-by-letter co-occurrence information becomes too much and so often irrelevant, we take term-by-term co-occurrence information in four words (4-gram) as the primitive to make up co-occurrence matrix useful to represent context of a text based on experimented results. We take high frequency 4-grams in order to make up WordSpace.
2. Construction of collocation matrix A *collocation matrix* is constructed in order to compare the context of two 4-grams. Element $a_{i,j}$ in this matrix is the number of 4-gram f_i which comes up just before 4-gram f_j (called *collocation area*). The collocation matrix counts how many other 4-grams come up before the target 4-gram. Each column of this matrix is the *4-gram vector* of the 4-gram f.
3. Construction of context vectors A *context vector* represents context of a word or phrase in a text. A sum of 4-gram vectors around appearance place of a word or phrase (called *context area*) is a context vector of a word or phrase in the place.
4. Construction of word vectors A word vector is a sum of context vectors at all appearance places of a word or phrase within texts, and can be expressed with Eq.1. Here, $\tau(w)$ is a vector representation of a word or phrase w, $C(w)$ is appearance places of a word or phrase w in a text, and $\varphi(f)$ is a 4-gram vector of a 4-gram f. A set of vector $\tau(w)$ is WordSpace.

$$\tau(w) = \sum_{i \in C(w)} \Big(\sum_{f \text{ close to } i} \varphi(f) \Big) \tag{1}$$

5. Construction of vector representations of all concepts The best matched "synset" of each input terms in WordNet is already specified, and a sum of the word vector contained in these synsets is set to the vector representation of a concept corresponding to a input term. The concept label is the input term.
6. Construction of a set of similar concept pairs Vector representations of all concepts are obtained by constructing WordSpace. Similarity between concepts is obtained from inner products in all the combination of these vectors. Then we define certain threshold for this similarity. A concept pair with similarity beyond the threshold is extracted as a similar concept pair.

Finding Association Rules between Input Terms The basic association rule algorithm is provided with a set of transactions, $T := \{t_i \mid i = 1..n\}$,where each transaction t_i consists of a set of items, $t_i = \{a_{i,j} \mid j = 1..m_i, a_{i,j} \in C\}$ and each item $a_{i,j}$ is form a set of concepts C. The algorithm finds association rules $X_k \Rightarrow Y_k : (X_k, Y_k \subset C, X_k \cap Y_k = \{\})$ such that measures for support and confidence exceed user-defined thresholds. Thereby, support of a rule $X_k \Rightarrow Y_k$ is the percentage of transactions that contain $X_k \cup Y_k$ as a subset (Eq.2)and confidence for the rule is defined as the percentage of transactions that Y_k is seen when X_k appears in a transaction (Eq.3).

$$support(X_k \Rightarrow Y_k) = \frac{\mid \{t_i \mid X_k \cup Y_k \subseteq t_i\} \mid}{n} \tag{2}$$

$$confidence(X_k \Rightarrow Y_k) = \frac{\mid \{t_i \mid X_k \cup Y_k \subseteq t_i\} \mid}{\mid \{t_i \mid X_k \subseteq t_i\} \mid} \tag{3}$$

As we regard input terms as items and sentences in text corpus as transactions, DODDLE II finds associations between terms in text corpus. Based on experimented results, we define the threshold of support as 0.4% and the threshold of confidence as 80%. When an association rule between terms exceeds both thresholds, the pair of terms are extracted as candidates for non-taxonomic relationships.

Constructing and Modifying Concept Specification Templates A set of similar concept pairs from WordSpace and term pairs from the association rule algorithm becomes concept specification templates. Both of the concept pairs, whose meaning is similar (with taxonomic relation), and has something relevant to each other (with non-taxonomic relation), are extracted as concept pairs with above-mentioned methods. However, by using taxonomic information from TRA module with co-occurrence information, DODDLE II distinguishes the concept pairs which are hierarchically close to each other from the other pairs as *TAXONOMY*. A user constructs a domain ontology by considering the relation with each concept pair in the concept specification templates, and deleting unnecessary concept pairs.

3 Case Studies

In order to evaluate how DODDLE II is doing in a practical field, case studies have been done in particular field of law and business. The particular field of law is called "Contracts for the International Sale of Goods"(CISG)[6] and the particular field of business is called "XML Common Business Library"(xCBL)[7]. DODDLE II is being implemented on Perl/Tk now. Figure 6 shows s snapshot.

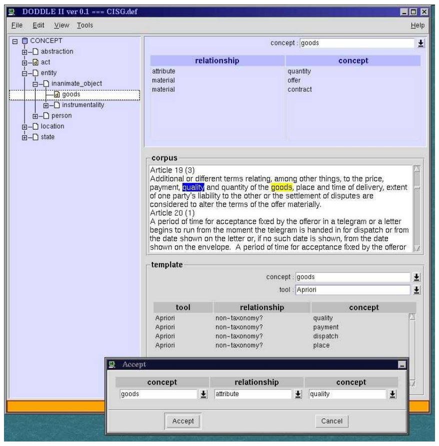
3.1 A Case Study in the Law Field

Input terms in the Case Study with CISG

A layer is a user in this case study. Table 1 shows significant 46 legal terms extracted by the user from CISG Part-II. He gave them DODDLE II as input terms.

Taxonomic Relationship Aqcuisition

Table 2 shows the number of concepts in each model under taxonomic relationship acquisition and Fig.7 shows the concept hierarchy constructed by the user using DODDLE II. Table 3 shows the evaluation of two strategies by the user. Precision is the percentage of extracted subtrees that were accepted, recall per path is the percentage of extracted paths that were accepted and recall per subtree is the percentage of extracted subtrees that were accepted. Precision and both recalls are less than 0.3 and are not good.



concept : goods

relationship	concept
attribute	quantity
material	offer
material	contract

corpus

Article 19 (3)
Additional or different terms relating, among other things, to the price, payment, quality and quantity of the goods, place and time of delivery, extent of one party's liability to the other or the settlement of disputes are considered to alter the terms of the offer materially.
Article 20 (1)
A period of time for acceptance fixed by the offeror in a telegram or a letter begins to run from the moment the telegram is handed in for dispatch or from the date shown on the letter or, if no such date is shown, from the date shown on the envelope. A period of time for acceptance fixed by the offeror

template

concept : goods
tool : Apriori

tool	relationship	concept
Apriori	non-taxonomy?	quality
Apriori	non-taxonomy?	payment
Apriori	non-taxonomy?	dispatch
Apriori	non-taxonomy?	place

Accept

concept	relationship	concept
goods	attribute	quality

Fig. 6. The Ontology Editor

Table 1. Significant 46 Concepts in CISG Part-II

acceptance	act	addition	address	assent	circumstance
communication	conduct	contract	counteroffer	day	delay
delivery	discrepancy	dispatch	effect	envelope	goods
holiday	indication	intention	invitation	letter	modification
offer	offeree	offerer	party	payment	person
placeofbusiness	price	proposal	quality	quantity	rejection
reply	residence	revocation	silence	speechact	system
telephone	telex	time	transmission	withdraw	

But about 70 % of the concept hierarchy (taxonomic relationships) were constructed with TRA module support and about half portion of them results in the information extracted from WordNet. Therefore we evaluated TRA model worked well in this case study. The detail of this case study is described in [2].

Non-Taxonomic Relationship Learning

Construction of WordSpace High-frequency 4-grams were extracted from CISG (about 10,000 words) standard form conversion removed duplication, and 543 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. As CISG is the comparatively small-scale text, the extraction frequency was set as 7 times this case. In order to construct a context vector, a sum of 4-gram vectors around appearance place circumference of each of 46 concepts was calculated. In order to construct a context scope from some 4-grams, it consists of putting together 60 4-grams before the 4-gram and 10 4-grams after the 4-grams independently of length of a sentence. For each of 46 concepts, the sum of context vectors in all the appearance places of the concept in CISG 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 the 1,035 concept pairs which are all the combination of 46 concepts, and having used threshold as 0.87, 77 concept pairs were extracted.

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

Model	Input Terms	Initial Model	Triimed Model	Concept Hierarchy
# Concept	46	133	56	61

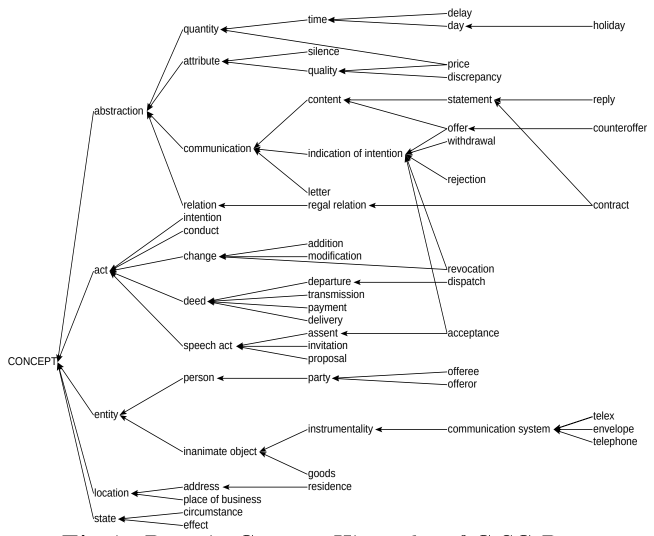


Fig. 7. Domain Concept Hierarchy of CISG Part II

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

	Precision	Recall per Path	Recall per Subtree
Matched Result	0.25(4/16)	0.23(5/21)	0.19(4/19)
Trimmed Result	0.3(3/10)	0.3(6/20)	0.15(3/20)

Finding Associations between Input Terms In this case, DODDLE II extracted 55 pairs of terms from text corpus using the above-mentioned association rule algorithm. There are 15 pairs out of them in a set of similar concept pairs extracted using WordSpace.

Constructing and Modifying Concept Specification Templates Concept specification templates were constructed from two sets of concept pairs extracted by WordSpace and Associated Rule algorithm. In concept specification templates, such a concept is distinguished as *TAXONOMY* relation. As taxonomic and non-taxonomic relationships may be mixed in the list based on only context similarity, the concept pairs which may be concerned with non-taxonomic relationships are obtained by removing the concept pairs with taxonomic relationships. After the user thought concept definitions, the user modified concept specification templates. Figure 8.(A) shows concept specification templates about the concept "goods". Figure 8.(B) shows the definition of the concept "goods" constructed from consideration of concept pairs in the templates.

Evaluation of Results of NTRL module The user evaluated the following two sets of concept pairs: one is extracted by WordSpace(WS) and the other is extracted by Association Rule algorithm (AR). Figure 9 shows three different sets of concept pairs from the user,

(A) Concept Specific Template
non-TAXONOMY? : **quality**
non-TAXONOMY? : **quantity**
non-TAXONOMY? : **contract**
non-TAXONOMY? : **act**
non-TAXONOMY? : **delivery**
non-TAXONOMY? : **effect**
non-TAXONOMY? : **party**
non-TAXONOMY? : **payment**
non-TAXONOMY? : **person**
non-TAXONOMY? : **price**
non-TAXONOMY? : **time**

$\Rightarrow$

(B) Concept Definition
ATTRIBUTE : **quality**
ATTRIBUTE : **quantity**
MATERIAL : **contract**
MATERIAL : **offer**

Fig. 8. The Concept Specification Templates and Concept Definition for "goods"

Table 4. Evaluation by the User with Legal Knowledge

	WordSPace (WS)	Association Rules (AR)	The Join of WS and AR
# Extracted concept pairs	77	55	117
# Accepted concept pairs	18	13	27
# Rejected concept pairs	59	42	90
Precision	0.23(18/77)	0.24(13/55)	0.23(27/117)
Recall	0.38(18/48)	0.27(13/48)	0.56(27/48)

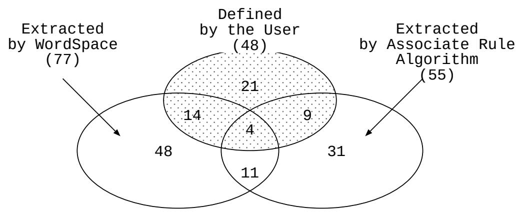

Extracted by WordSpace (77)
Defined by the User (48)
Extracted by Associate Rule Algorithm (55)
21
14
9
4
48
31
11

Fig. 9. Three Different Sets of Concept Pairs from User, WS and AR

WS and AR. Table 4 shows the details of evaluation by the user, computing precision and recall. Precision is the percentage of concept pairs accepted by a user that were extracted by DODDLE II. Recall is the percentage of concept pairs extracted by DODDLE II that were defined by a user. Looking at the field of Precision in Table 4, there is almost no differences among three kinds of results from WS,AR and the join of them. However, looking at the field of Recall in Table 4, the recall from the join of WS and AR is higher than that from each WS and AR, and then goes over 0.5.

3.2 A Case Study in the Business Field

Input terms in the Case Study with xCBL

Table 5 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.

Taxonomic Relationship Acquisition

Table 6 shows the number of concept pairs in each model under taxonomic relationship acquisition and table 7 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

Table 5. 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 6. 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

Table 7. 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)

constructed with TRA module support. We evaluated TRA module worked well in this case study.

Non-Taxonomic Relationship Learning

Construction of WordSpace High-frequency 4-grams 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. As xCBL text are shorter than CISG text, the extraction frequency was set as 2 times this case. 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.

Finding Associations between Input Terms DODDLE II 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 II constructed concept specification templates from two sets of concept pairs extracted by WordSpace and Associated Rule algorithm. However, the user didn't have enough time to modify them and didn't finish to modify them.

Evaluation of Results of NTRL 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 10 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 8 shows the details of evaluation by the user, computing precision only. Because the user didn't define concept definition in advance, we can not compute recall. Looking at the field of precision in Table 8, 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.

3.3 Results and Evaluation of Case Studies

In regards to support in constructing taxonomic relationships, the precision and recall are less than 0.3 in both case studies and there is almost no difference. Generally, 70 %

Table 8. 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)

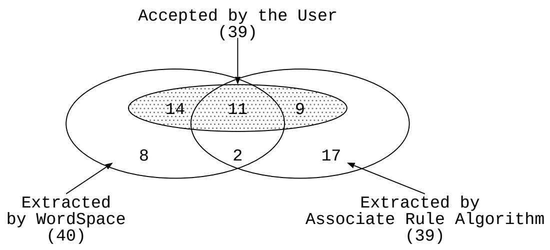


Fig. 10. Two Difference Sets of Concept Pairs from WS and AR and Concept Sets have Relationships

or more support comes from TRA module. About more than half portion of the final domain ontology results in the information extracted from WordNet. Because 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, but the precision in the case study with CISG is less than 0.3 and not good. Generating non-taxonomic relationships of concepts is harder than modifying and deleting them. Therefore, DODDLE II 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 the article, but DODDLE II 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.

4 Related Work

Navigli et,al. proposed OntoLearn [8][9][10], 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 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 [11], we had an experience that even if the concepts are in the MRD, 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. [13]). In [14], 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 [12]. 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.

5 Conclusions

In this paper, we discussed how to construct a domain ontology using an existing MRD and text corpus. In order to acquire taxonomic relationship, two strategies have been proposed: matched result analysis and trimmed result analysis. Furthermore, to learn non-taxonomic relationships, concept pairs may be related to concept definition, extracted on the basis of the co-occurrence information in text corpus, and a domain ontology is developed by the modification and specification of concept relations with concept specification templates. It serves as the guideline for narrowing down huge space of concept pairs to construct a domain ontology.

It is almost craft-work to construct a domain ontology, and still difficult to obtain the high support rate on system. DODDLE II mainly supports for construction of a concept hierarchy with taxonomic relationships and extraction of concept pairs with non-taxonomic relationships now. However a support for specification concept relationship is indispensable. The future works are as follows: improvement in the scalability of the definition support by learning of heuristics and introduction of the useful datamining method instead of WordSpace, and system integration of taxonomic relationship acquisition module and non-taxonomic relationship learning module (now implementing).

6 Acknowledgment

This work has been supported by Naomi Nakaya and Takamasa Iwade (former students of Shizuoka University, Japan).

References

1. Y. Ding and S.Foo: “Ontology Research and Development, Part 1 – A Review of Onlotogy”, Journal of Information Science, Vol.28, No2, 123 – 136, 2002
2. Rieko Sekiuchi, Chizuru Aoki, Masaki Kurematsu, and Takahira Yamaguchi: “DODDLE : A Domain Ontology Rapid Development Environment”, PRICAI98, 1998
3. C.Fellbaum ed: “Wordnet”, The MIT Press, 1998. see also URL: http://www.cogsci.princeton.edu/~wn/
4. Rakesh Agrawal and Ramakrishnan Srikant :“Fast algorithms for mining association rules,”, Proc. of VLDB Conference, 487–499, 1994
5. Marti A. Hearst and Hinrich Schutze: “Customizing a Lexicon to Better Suit a Computational Task”, in Corpus Processing for Lexical Acquisition edited by Branimir Boguraev & James Pustejovsky, 77–96
6. Kazuaki Sono and Masasi Yamate: United Nations convention on Contracts for the International Sale of Goods, Seirin-Shoin, 1993
7. xCBL.org: http://www.xcbl.org/xcbl40/documentation/ listofdocuments.html
8. R. Navigli and P. Velardi: “Ontology Learning and Its Application to Automated Terminology Translation”, IEEE Intelligent Systems, JANUARY/FEBRUARY, pp.22–31, 2003.
9. Roberto Navigli and Paola Velardi: “Automatic Adaptation of WordNet to Domains”, Proc. of International Workshop on Ontologies and Lexical Knowledge Bases(OntoLex2002), 2002.
10. P. Velardi, M. Missikoff, and P. Fabriani : “Using Text Processing Techniques to Automatically enrich a Domain Ontology”, Proc. of ACM Conf. On Formal ontologies and Information Systems(ACM FOIS), pp.270–284, October 2001.
11. T. Yamaguchi: “Constructing Domain Ontologies Based on Concept Drift Analysis”, Proc. of the IJCAI99 Workshop on Ontologies and Problem Solving methods(KRR5), August 1999.
12. Alexander Maedche and Steffen Staab: “Discovering Conceptual Relations from Text”, ECAI2000, 321–325, 2000
13. Udo Hahn and Klemens Schnattingerg: “Toward Text Knowledge Engineering”, AAAI98, IAAAI-98 proceedings, 524-531, 1998
14. David Faure and Claire Nédellec: “Knowledge Acquisition of Predicate Argument Structures from Technical Texts Using Machine Learning: The System ASIUM”, EKAW’99

An Ontology-Driven Application to Improve the Prescription of Educational Resources to Parents of Premature Infants

Howard Goldberg[1], Alfredo Morales[1], David MacMillan[2], and Matthew Quinlan[2]

[1] Clinician Support Technologies Inc, 1 Wells Avenue, Newton, MA 02459
[2] Network Inference Limited, 25 Chapel Street, London NW1 5DH

Abstract. CST's Baby CareLink provides a 'collaborative healthware' environment for parents of premature infants that incorporates just-in-time learning as one means of knowledge exploration and patient empowerment [1], [2], [3]. As the Baby CareLink content base has continued to grow, it has become increasingly difficult for content prescribers to identify all relevant resources for parents at a given point in time in their child's course of care. In addition, the growing content base has become increasingly difficult to maintain without a rich indexing system. In order to address these issues, we have developed an ontology-driven application that supports the indexing and retrieval of educational materials according to rich descriptions of premature infants.

We have developed an initial OWL-DL-based [4] ontology describing relevant concepts in the domain of neonatology, including clinical conditions, diagnostic testing, therapies, durable medical equipment, and the infants themselves. We have developed an initial terminologic model describing typical clinical problems and therapies that occur over the clinical course of these premature infants. Indexers tag educational resources through a web based client application that allows them to create rich descriptions of educational resources based on the reference ontology. Clinical end-users interact with a client application that identifies educational resources appropriate to clinical scenarios occurring over the course of a typical premature infant's development based on the description generated from records of existing infants in the Baby CareLink system, or according to a user-created description.

Network Inference's tools were used to develop the neonatology ontology and implement the run-time system. Construct™, a Visio-based ontology modeling tool was used to develop the reference ontology. This tool allowed the representation of the domain's concepts, subsumption relationships, properties, instances, and axioms diagrammatically. Cerebra Server™, an OWL-DL-based inferencing platform was used to provide logical consistency-checking at modeling time directly from Construct™. Cerebra Server™ was integrated with Baby CareLink and to the indexing and retrieving tools through a .Net connector that provides an API for 1) extending the ontology at indexing time with newly-classified educational resources, 2) dynamically creating instances of individual premature babies using data from Baby CareLink and 3) querying the ontology at run-time.

This knowledge centric approach to the identification and recommendation of educational resources is anticipated to better individualize information for parents of infants managed through Baby CareLink and increase the efficacy of the information prescription process. The approach is expected to increase the return on investment provided by Baby CareLink through reduction in the time required by nurses and care managers to interact with the system.

Introduction

CST's Baby CareLink provides a 'collaborative healthware' environment for parents of premature infants that incorporates just-in-time learning as one means of knowledge exploration and patient empowerment [1], [2], [3]. As the Baby CareLink content base has continued to grow, it has become increasingly difficult for content prescribers to identify all relevant resources for parents at a given point in time in their child's course of care. In addition, the growing content base has become increasingly difficult to maintain without a rich indexing system. In order to address these issues, we have developed an ontology-driven application that supports the indexing and retrieval of educational materials according to rich descriptions of premature infants.

Ontology Design

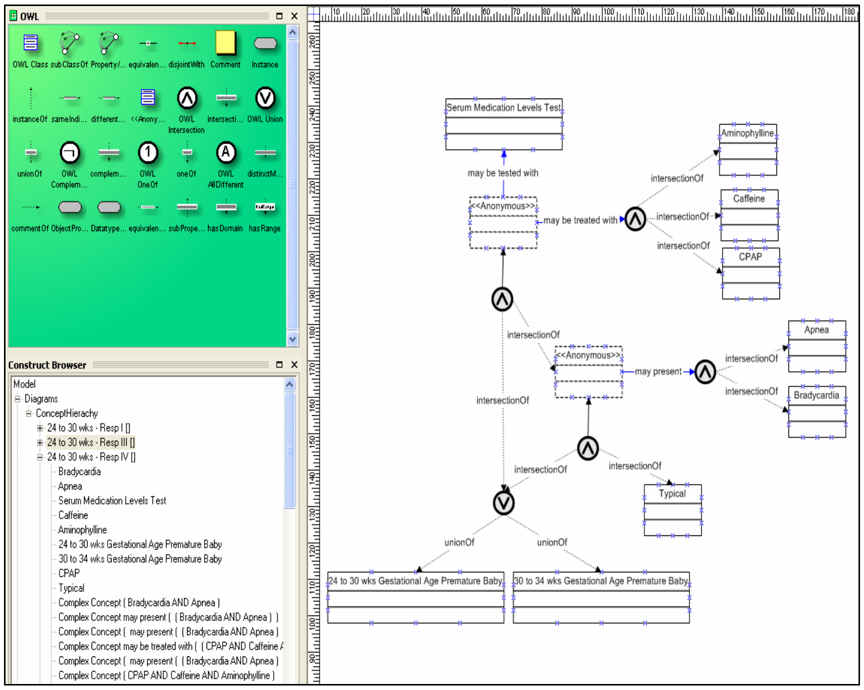
Initial work on the Proof of Concept (POC) focused on modelling the neonatology domain, representing educational resources, and designing appropriate queries. The resulting model represents a base ontology composed of two main types of constructs—hierarchies of core concepts and axioms describing prototypical premature infants. The resulting ontology comprises approximately 300 entities, including concepts, object properties, and complex concepts.

The foundation for the model represents the core domain concepts to be used within the ontology. The core concepts are organized in hierarchies describing babies, problems, treatments, tests and educational resources. Each category was elaborated to a level of granularity required in the context of the POC. For example, Treatment comprises Therapies, Medications and Procedures. Categories of Baby were defined according to standard properties, such as gestational age, using classes with customized complex datatypes to express value ranges within which patients could be classified. A number of Object Properties were defined (e.g. "may present", "may be treated", "may be tested") in order to facilitate the definition of relationships between concepts. These are also used later in the process of querying the live ontology

An additional class called 'Typical' was introduced into the model. The Typical class facilitates the creation of a taxonomic model describing prototypical premature infants, their presenting problems, and usual therapies. This strategy addresses reuse by permitting very general axioms in the model, e.g., "*A typical 24-36 week old* *premature infant presenting with respiratory distress syndrome may be treated with surfactant and either mechanical ventilation or CPAP*" without burdening all instantiations of 24-30 week-old babies with the generalization. In the application, we use this duality to enable caregivers to subsequently differentiate between educational resources relevant to those problems a baby might be **expected** to present and those based upon what the baby is **known** to present.

The model of prototypical premature infants contained in the ontology consists of complex concepts that define required and optional combinations of tests and treatments for problems presented by babies within specific age ranges. The OWL-DL constructs *intersectionOf* and *unionOf* were used to define relationships and restrictions over concepts in the ontology. Problems, Treatments and Tests were associated graphically with the intersection of the Typical class and sub-classes of Baby (see Figure 1).

Finally, the Educational Resource class was represented. Education Resources 'refer to' arbitrarily complex descriptions of premature infants. The representations used to index educational resources are created programmatically through end-user tools described in the following section.



Model
- Diagrams
  - ConceptHierachy
    - 24 to 30 wks - Resp I []
    - 24 to 30 wks - Resp III []
    - 24 to 30 wks - Resp IV []
      - Bradycardia
      - Apnea
      - Serum Medication Levels Test
      - Caffeine
      - Aminophylline
      - 24 to 30 wks Gestational Age Premature Baby
      - 30 to 34 wks Gestational Age Premature Baby
      - CPAP
      - Typical
      - Complex Concept (Bradycardia AND Apnea)
      - Complex Concept may present ((Bradycardia AND Apnea))
      - Complex Concept (may present ((Bradycardia AND Apnea)
      - Complex Concept may be treated with ((CPAP AND Caffeine A
      - Complex Concept (may present ((Bradycardia AND Apnea)
      - Complex Concept (CPAP AND Caffeine AND Aminophylline)

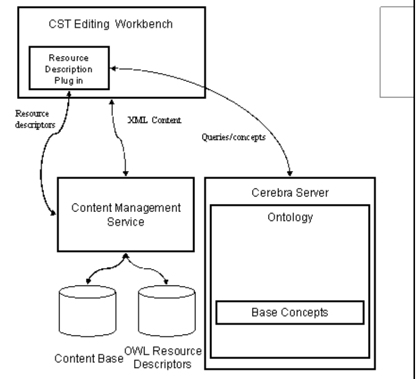
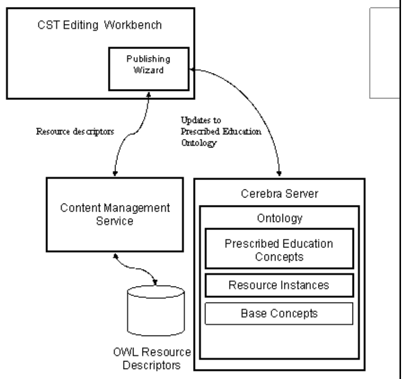
Fig. 1: Description of Babies with Gestational Age 24-34 weeks, in relation to specific problems, treatments, and tests

System Overview

The application is architected to integrate at author-time with Baby CareLink's content management system (CMS). Domain experts create and publish educational resources through a web-based editing workbench, part of the CMS. Two further tools were added to the workbench to support the description of resources—the Resource Descriptor Plug-in and the Publishing Wizard. Figure 2 shows the architectural components and their interactions at each stage.

The Resource Descriptor Plug-in allows domain experts to generate metadata about resources in the Baby CareLink content base, using templates that represent prototypical relationships between concepts and constructs that exist in the base ontology. The plug-in dynamically obtains concepts through XQueries posted to Cerebra via its client API. The outputs of the plug-in are resource descriptors in the form of OWL-DL fragments. Each resource descriptor is an instance of the Educational Resource concept, restricted by the combinations of Problems, Treatments and/or Tests relating to a rich description for a premature infant. The source for resource descriptors is stored in the content management system.

The Publishing Wizard facilitates the deployment of resource descriptors into Cerebra. It interacts with both the CMS and Cerebra to merge completed resource descriptors into the active ontology. The source for the resulting extended ontology must also be saved should the ontology need to be reloaded into Cerebra.

 CST Editing Workbench Resource Description Plug in Resource descriptors XML Content Queries/concepts Content Management Service Cerebra Server Ontology Base Concepts Content Base OWL Resource Descriptors	 CST Editing Workbench Publishing Wizard Resource descriptors Updates to Prescribed Education Ontology Content Management Service Cerebra Server Ontology Prescribed Education Concepts Resource Instances Base Concepts OWL Resource Descriptors
Figure 2.a – Domain expert creates resource descriptors using templates and concepts obtained from the ontology.	Figure 2.b – Once all education resources are described by resource descriptors, these are published to Cerebra.

In addition to the author-time tools, Baby CareLink was extended in order to represent patients within the inferencing system. A Patient Profiler module was developed that synthesizes CareLink data from registered infants into instances of the Baby concept. These instances serve as the focal point with which to identify educational resources for each patient. The Patient profiler acts as a daemon, executing at periodic intervals and forcing reclassification of the resulting ontology once new Baby instances have been merged.

At run-time, the Information Prescription Pad extends the existing Prescribed Education module by providing rapid, fine-grained searches against the universe of educational resources known to Cerebra. Clinicians interact with the Pad's user interface in order to create a description of an appropriate baby to be used for search. The interface presents a series of templates for completion using concepts from the ontology. Clinicians may build a description from an existing baby or a prototypical baby.

The Information Prescription Pad queries Cerebra in order to retrieve concepts that may be valid fillers for slots of the selected template. Posting an instanceProperty query to Cerebra using a Baby instance and an object property will return all valid fillers for the specified property. If the clinician is interested in what typically could be relevant to a particular patient, the Pad uses the specific Baby instance representing that patient as part of the query – i.e., the conjunction of 'Premature Baby 1' and the concept 'Typical'. If the clinician is interested in a prototypical baby, the Pad will use the concept that represents that type of baby when querying Cerebra Server – i.e., 'Premature Baby with 24 to 30 weeks Gestational Age' and the concept 'Typical'. Once all templates have been completed, the Information Prescription Pad formulates an XQuery and posts it to Cerebra through the client API.

The query returns instances of Educational Resources that match the criteria specified in the template. Each instance has a property containing the URI of the resource to be prescribed. The Information Prescription Pad renders the list of applicable resources and provides both a way to review the document and to assign it to the parents of a baby. After relevant resources have been chosen, the Information Prescription Pad updates the User Profile of the chosen parent, assigning the resources for her review the next time that she accesses Baby CareLink.

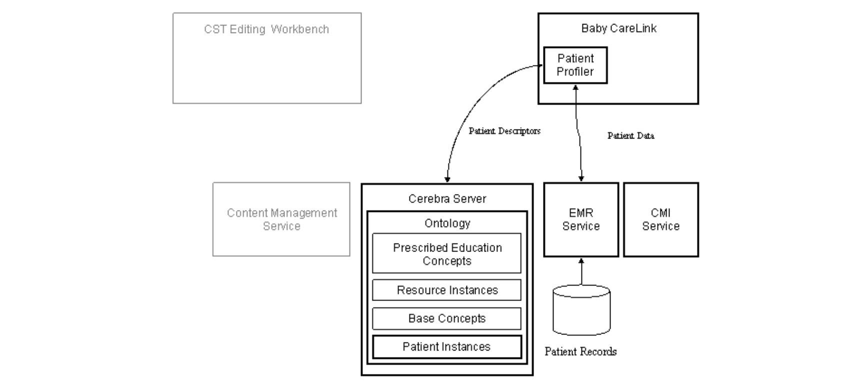


Fig. 3.a. Baby CareLink creates instances describing patients and merges them into Cerebra.

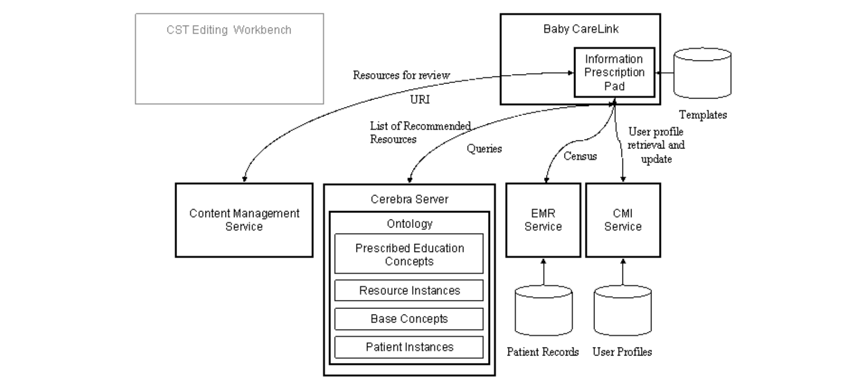


Fig. 3.b. Clinicians use the Information Prescription Pad to query Cerebra for educational resources

Evaluation

At the time of submission, the POC system is being alpha tested at CST. Domain experts are in the process of defining resource descriptors for the more than 800 documents that comprise the Baby CareLink content base, using the Resource Descriptor Plug-in added to the Editing Workbench. The first stage of the process centers on creating metadata for the subset of documents related to respiratory problems, their treatments and the tests that could be performed on a baby presenting such problems. The domain experts' feedback is being used to refine the templates originally developed.

Conclusions and Impacts

We have developed an ontology-driven application that facilitates the prescription of educational materials to parents of premature infants. Our ontology supports reasonably high-fidelity representations of neonates, their clinical problems, and ongoing treatments. These representations allow for a rapid, fine-grained search against a document set of approximately eight hundred documents. The ontology has also supported the development of a taxonomic model of potential problems and treatments occurring in typical neonates over time. The taxonomic model supports the creation of a module to answer "what if" questions regarding typical clinical scenarios, e.g., "What are the typical treatments for a 27-week-only baby with respiratory distress syndrome?"

This early work is significant in several dimensions. We were able to develop a small, but robust ontology supporting a sophisticated retrieval application using the current draft of OWL-DL. The feature set of the Cerebra Server provided model-time validation of the developing ontology, real-time updates to the ontology, and adequate querying and inferencing capabilities for both terminologic reasoning, and reasoning about instances. We are able to incorporate Cerebra as a modular inferencing service within a larger software architecture.

While the CST modelers had previous familiarity with description logic systems such as LOOM [5], they found that OWL-DL still had a significant learning curve. For example, understanding OWL representations for necessary and sufficient conditions requires multiple re-readings of the OWL reference manual to ensure correct subsumption relationships. The Construct modeling tool simplifies this by hiding the verbosity of the OWL language, but the modelers also found it was necessary to think in terms of axiomatic representations in addition to object models. Much of the collaborative effort between CST and Network Inference was spent in identifying critical OWL modeling idioms that would scale as the ontology grew. Additionally, while terminologic query capabilities were adequate for this application, there remains ground for additional improvements in the expressiveness of a query language for OWL. Standard notions of time, which are important for our application, remain to be adapted for an OWL environment. Finally, while we were able to adapt datatypes for use in characterizing age ranges for our infant models, it would be useful to incorporate numeric comparison operators into OWL as well.

Future work will examine the use or integration of existing healthcare ontologies into our system. Generalization of our work to additional medical domains is best accomplished through extension of existing ontologies as opposed to *de novo* development. Fortunately, there is an existing base of formal ontologies for the health care domain, such as GALEN [6] and SNOMED [7], with which to develop applications. The features and employed idioms of these ontologies must be evaluated to establish their use in knowledge mediators such as the one we have developed. Additionally, these are very large ontologies, whose applicability to Description Logic reasoning and performance characteristics must be determined for the inference engine and architecture we are deploying.

Our early work adds additional evidence to the utility of ontology-driven knowledge mediators, as previously demonstrated by systems such as TAMBIS [8] and Ariadne/SIMS [9]. This work adds the additional capability for the system to reason over the domain of interest through the additional terminologic model of typical premature infants over time. By addressing real-world implementations of ontology-driven knowledge mediators, we believe this class of mediators may be one of the early 'killer applications' for semantic web technologies.

References

1. Gray JE, Safran C, Davis RB, Pompilio-Weitzner G, Stewart JE, Zaccagnini L, Pursley D.: Baby CareLink: Using the Internet and Telemedicine to Improve Care for High-Risk Infants. In: Pediatrics. (2000) Dec; 106(6):1318-24.
2. Goldberg H, Morales A, Gottlieb L, Meador L, Safran C. Reinventing Patient-Centered Computing for the Twenty-first Century. Medinfo 2001, (2001) 1455- 1458
3. Safran, C. The Collaborative Edge: Patient Empowerment for Vulnerable Populations. International Journal of Medical Informatics 69 (2003) 185-190.
4. McGuinness, D. L., van Harmelen, F.: OWL Web Ontology Language Overview W3C Working Draft. W3C (2003).
5. MacGregor RM. Using a Description Classifier to Enhance Deductive Inference. *Proceedings Seventh IEEE Conference on AI Applications*, pp. 141-147, 1991.
6. Rector AL, Bechhofer SK, Goble CA, Horrocks I, Nowlan WA, Solomon WD. The GRAIL Concept Modelling Language for Medical Terminology**.** *Artificial Intelligence in Medicine,* Volume 9, 1997.
7. Spackman KA, Campbell KE, Cote RA. SNOMED RT: A reference terminology for health care. *Proceedings/AMIA Annual Fall Symposium*:640-4, 1997.
8. Goble CA, Stevens R, Ng G, Bechhofer S, Paton NW, Baker PG, Peim M, and Brass A. Transparent Access to Multiple Bioinformatics Information Sources. IBM Systems Journal, 40(2):532 - 552, 2001.
9. Knoblock CA. Planning, executing, sensing, and replanning for information gathering. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

Description of the experiment. Tools Interoperability

The objective of the interoperability experiment proposed in EON2003 is to analyze how ontologies can be exchanged (exported and/or imported) between different tools - either ontology-related or general software-engineering related – and/or languages. We foresee to obtain as results of this experiment a set of conclusions, metrics and guidelines to assess on the quality of exports and imports, interoperability, and on how exported/imported ontologies can be integrated in different tools. These guidelines could be also used to decide in which cases it is better to use one ontology tool or another for different domains and with different modelling/reasoning needs.

Although it is not a requisite to perform this experiment, we recommend to use the same ontology description that was used in EON2002, in the travelling domain, which can be found in the EON2002 site (*http://km.aifb.uni-karlsruhe.de/eon2002*).

As an example, the following protocol can be used to perform this experiment:
1. Develop an ontology with an ontology tool (or reuse it from the EON2002 ontologies repository).
2. Export the ontology to other ontology languages and/or tools, depending on the export capabilities of the selected tool.
3. Assess the quality of the transformations performed by the selected tool, analyzing the losses of information in the translation process.
4. Import the ontology to tools able to import the format in which the ontology is available. This import can be also performed with the same tool used to develop the ontology.
5. Assess the quality of the transformation performed by the selected tools, analyzing the losses of information in the translation process.
6. Analyze the differences between the original ontology and the ontology that results from this circular transformation.

The previous protocol is suggested as a possible choice for experimenting interoperability between tools and languages. The workshop will be open to any other configurations where the central issue of interoperability is handled, such as comparing how different tools export to another language/tool, how they import from another language/tool, etc.

Using XSLT for Interoperability: DOE and The Travelling Domain Experiment

Antoine Isaac[1,2], Raphaël Troncy[1,3], and Véronique Malaisé[1,4]

[1] Institut National de l'Audiovisuel, Direction de la Recherche, Équipe DCA
4, Av. de l'Europe - 94366 Bry-sur-Marne
{aisaac,rtroncy,vmalaise}@ina.fr
http://www.ina.fr/

[2] Université de Paris-Sorbonne, LaLICC, http://www.lalic.paris4.sorbonne.fr

[3] INRIA Rhône-Alpes, Équipe EXMO, http://www.inrialpes.fr/exmo

[4] AP-HP, Équipe STIM, http://www.biomath.jussieu.fr

1 Introduction

This paper presents the results of the use of the *Differential Ontology Editor*[5] (**DOE**) during the second experiment on the evaluation of ontology-related technologies that was initiated by the OntoWeb thematic network[6].

The first experiment aimed at evaluating the modeling of an ontology in various environments through a shared description of the travelling domain written in natural language. The workshop (organized at the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2002) in Spain) showed that the results were very heterogeneous [1]. This second experiment proposes to analyze how an ontology can be exchanged (exported and/or imported) between different tools.

We have used the following protocol to perform this experiment:

1. Reuse the ontology modeled with the DOE tool for the EON 2002 Workshop;
2. Perform a circular transformation with RDFS as exchange language, that is:
   - Export the DOE ontology in RDFS [10], import it in each of the four following environments: **Protégé2000** [5], **OilEd** [4], **OntoEdit** [11] and **WebODE** [2] and assess the quality of the transformation;
   - Modify the ontology in these environments and export it again in RDFS; Import the result in DOE and assess the quality of the transformation;
   - Compare the original ontology and the final ontology that results from this circular transformation.
3. Perform another circular transformation with the same tools, but with OWL [6] as exchange language.

[5] The tool is available for free at http://opales.ina.fr/public/. A repository of ontologies are available in various languages at http://opales.ina.fr/public/eon2003/.

[6] See the Special Interest Group homepage at http://delicias.dia.fi.upm.es/ontoweb/sig-tools/index.html.

The remainder of the paper is organized as follows. In section 2, we draw some conclusions about the last experiment and the very heterogeneous ontologies developed. Section 3 presents the DOE methodology, the DOE tool and our design choices for modeling the travelling domain ontology. We also introduce the way we use XSLT stylesheets to exchange the ontology model, since all languages have an XML serialization. The experiment begins in section 4, where we follow the protocol with RDFS as exchange language, and continues in section 5, with OWL as exchange language. Finally, we give in section 6 the conclusions that can be derived from these experiments.

2 Lessons Learned From the EON'2002 Workshop

The first EON workshop led to the creation of ten ontologies that were very heterogeneous with respect to their conceptualization. First of all, the ontologies can be classified in two categories: the ones with top-level concepts and relations, and the ones without. The first category is clearly more interested in the taxonomy structure (with design decisions inspired by the philosophy) whereas the second one is rather focused on the concepts that the user needs. Regarding the description of the domain, all ontologies have, more or less, modeled the same concepts. However, the taxonomies produced are quite different. The main differences appear in the branches "means of transport" and "reservation".

According to us, these differences are mainly due to the lack of particular use cases that could guide the ontology design. The application using this ontology was not specified and each ontologist has freely interpreted how to define and classify the concepts. The knowledge representation paradigm supporting the modeling and the tool itself contribute to the variability of the ontologies built. For instance, tools based on the frame paradigm support concept attributes (binary relations) whereas DOE can model n-ary relations between concepts.

3 DOE: The Differential Ontology Editor

Many approaches (for a complete survey, the reader can refer to the OntoWeb Technical RoadMap[7]) have been reported to build ontologies, but only few of them detail the steps needed to obtain and structure the taxonomies. This observation has previously led us to propose a methodology entailing a semantic commitment to normalize the meaning of the concepts [3]. We briefly present this methodology in section 3.1. In section 3.2, we describe how the DOE editor implements this methodology, and particularly how its exchange facilities, via import and export procedures, are performed by XSLT transformations. Finally, we present the design decisions that we have made to model the travelling domain ontology (section 3.3).

[7] http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html

3.1 General DOE Methodology

As shown in [3], none of the methodologies reported to build ontologies force the ontologist to explicit the real meanings of the concepts. The terms used to refer to the concepts are still liable to multiple interpretations. This results in possible misunderstandings and consequently bad modeling and use of the ontology. As a solution, we have already suggested a three-steps methodology that consists in:

- a *semantic normalization*: aims at reaching a semantic agreement about the meaning of the labels used for naming the concepts;
- a *formalization*: aims at formalizing the ontology, that is, define the concepts (in the Description Logics sense), constrain the relations, add axioms (e.g. algebra properties), add instances, etc.
- an *operationalization*: allows to equip the concepts with the possible computational operations available in a particular knowledge representation language.

The first step is based on *differential semantics* [8]. Practically, the ontologist has to express, in natural language, the similarities and differences of each notion (concept or relation) with respect to its neighbors: its parent-notion and its siblings-notions. The result is a taxonomy of notions, where the meaning of a node is given by the gathering of all similarities and differences attached to the notions found on the way from the root notion (the more generic) to this node. We follow four principles to explicit this information:

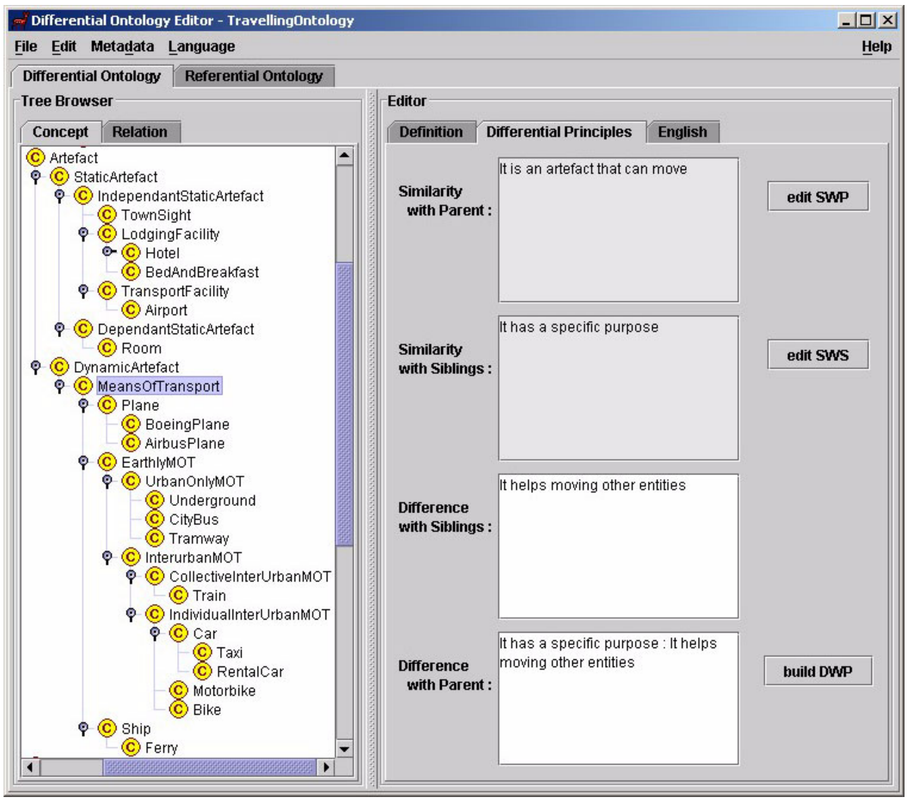
- The *similarity with parent* principle (or **SWP**): explicits why the notion inherits properties of the one that subsumes it;
- The *similarity with siblings* principle (or **SWS**): gives a semantic axis, a property – assuming exclusive values – allowing to compare the notion with its siblings.
- The *difference with siblings* principle (or **DWS**): precises here the property value allowing to distinguish the notion from its siblings;
- The *difference with parent* principle (or **DWP**): explicits the difference allowing to distinguish the notion from its parent;

For example, Figure 1 gives the differential principles bound to the notion `MeansOfTransport`.

In the second step, the notions that come from this conceptualization are added with a formal meaning. The ontologist can define some concepts, precise the domains of the relations, add axioms and rules or give instances for some concepts. Tools make also some consistency checking. For instance, they have to check the propagation of the arity along the hierarchy of relations – if specified – and the inheritance of the domains. The third and last step of the methodology allows to export the ontology modeled into a knowledge representation language.

3.2 The DOE Editor

DOE is a simple prototype that supports partially the three steps of the methodology detailed above. It is not intended to bring a direct competition with other existing environments. Rather, its purpose is to demonstrate by experimentation how taxonomy structuring can benefit from our proposed methodology.



Similarity with Parent : It is an artefact that can move

Similarity with Siblings : It has a specific purpose

Difference with Siblings : It helps moving other entities

Difference with Parent : It has a specific purpose : It helps moving other entities

Fig. 1. The differential principles bound to the notion *MeansOfTransport* in the *DOE* tool

During the first step, the ontologist can enter the definition of the notions according to our principles. The tool automatizes partly this task. The Figure 1 shows the interface with the concept `MeansOfTransport` highlighted, and its differential principles fillers. For the second step, the taxonomies built previously are shown and the editor allows the ontologist to specialize existing concepts and relations (without the differential information), as well as to specify the arity and domains of the relations. The last step is implemented by exporting the referential ontology into commonly-used knowledge representation languages

(RDFS, DAML+OIL or OWL for instance) that can be used by specific application environments. This export mechanism also allows to refine the ontologies built, using other editors and the features they support.

All the exchange facilities (import and export from various languages[8]) are performed, in an original way, with XSLT [12] transformations. Actually, all proposed languages for representing ontologies on the Web have an XML serialization and the ontology editors themselves have usually their model described in XML. Therefore, XSLT, which is a language for transforming XML documents into other XML documents, seems adapted to perform this task. The ontologist can also use its own stylesheet, dynamically from the file menu, in order to import (resp. export) ontologies. In this case, the user has to specify the URI of the stylesheet and the input (resp. output) source. This feature provides a flexible way to accomplish the interoperability between DOE and others ontology builder tools.

3.3 The Travelling Domain Ontology

After comparing with the other ontologies presented at the previous workshop, we made some minor changes in our ontology (add some concepts and relations). Our ontology contains a top level to be consistent with our methodology. The first distinction that we make concerns the possibility for entities to be spatio-temporally located or not (`ConcreteEntity` and `AbstractEntity`).

The `ConcreteEntity` is then considered mostly in regard of its spatial or temporal location. `TemporalEntity` includes the `Reservation` types. One difference with other ontologies is the treatment of *flight*. Here, this concept is seen as a special reservation. For any kind of reservation, the relation `motUsed` can be established with a particular means of transport (e.g. `Plane` for the `FlightReservation`). There are three kinds of `SpatialEntity`: `BiologicalObject` (the travel agent or the customer), `GeographicObject` (continent, country, city or resort) and `Artefact`. This last branch is composed of the different means of transport (by air, sea or earth) and of all types of building (town sight, hotel or transport facility).

The taxonomy of relations is not very deep. They are grouped according to their domain, like attributes (or slots) in other knowledge representation paradigms. The DOE editor, because of its Conceptual Graphs model, supports n-ary relations. We found this possibility particularly useful to model the relation `distance` that involves two spatial objects and the distance metrics.

Finally, it is not possible to write axioms in DOE because its purpose is mainly to guide the ontologist during the very first steps of the ontology conceptualization. Therefore, we could not specify that a travel between America and Europe could only be done with an airplane or a ship, or that the one to five star hotels were the only possible hotel categories.

[8] Technically, our editor can import ontologies written in RDFS and OWL, and supports export in RDFS, OWL, DAML+OIL and CGXML (a language for Conceptual Graphs specification). For adequacy concerns, we have limitted our paper to the languages currently focused on by the Semantic Web community.

4 RDFS as Exchange Language

The four following environments (Protégé2000 v2.0 beta, OilEd v3.5, OntoEdit v2.6 free release and DOE v1.5) can import and export RDFS ontologies. Consequently, we can do the export/import loop from DOE to each of them and come back. The RDFS import functionality seems to be unavailable online for the WebODE v2.0 tool and hence, cannot be tested. However, the export feature is available and we tested it after having imported our ontology via OWL.

4.1 From DOE To Other Environments

As seen in section 3.1, the main contribution of the DOE tool is to force the ontologist to assign a clear meaning to concepts through the use of differential principles. Our experiment has shown that we can keep a trace of this *semantic commitment* in produced RDFS ontologies by exporting all the related information into the `rdfs:comment` element.

Regarding formal expressiveness, our model is very limited. In fact it is very close to RDFS: DOE allows concept/relation specialization, domain and range assignment for relations, and concept instanciation. Therefore, it is not surprising that our editor easily manages to translate this information.

We then tried to open the produced RDFS file in other environments (the results are summarized in Table 1). OilEd was able to read it properly, like Protégé2000, which nevertheless encountered some problems dealing with the accents on letters found in the ontology. However, these two editors were not able to import the metadata associated to the ontology even if they were written according to the Dublin Core recommendation. OntoEdit managed to import the ontology model, but transformed the Dublin Core container by adding 11 DC elements to the relation list. It also added a mysterious instance that did not appear anywhere in the display and, after a glance at the exported RDFS file, proved to be a generated instance that has the DC attributes entered in the container. Finally, its pure frame-oriented interface did not show properties that had no domain defined, whereas they still existed in the model, which is quite disturbing.

4.2 From Other Environments To DOE

More problems occurred when trying to import back the RDFS ontologies exported by other environments. Firstly, we could not properly import the file exported by Protégé2000. We must mention that neither OilEd nor OntoEdit succeeded in this ordeal: it is due to RDFS errors, such as the use of `rdfs:label` as an attribute (instead of a sub-element) of `rdfs:Class`.

Secondly, both OntoEdit and WebODE do not use the RDF abbreviated syntax to encode the ontology. All class and property definitions are serialized as `rdf:Description` instead of `rdfs:Class` and `rdf:Property`. Consequently,

	DOE	**Protégé**	**OilEd**	**OntoEdit**	**WebODE**
Number of Concepts	79	79[a]	79	79	*NOT available online*
Number of Relations	48	48[b]	48	59	
Number of Instances	22	22	22	23	
Multiple Inheritance	exported	preserved	preserved	preserved	
Domains assignment	exported	preserved	preserved	preserved	
Ontology metadata	exported	omitted	omitted	transformed	
Differential Definition	exported	displayed	displayed	displayed	

[a] Protégé adds 15 system classes.
[b] Protégé adds 34 system slots.

Table 1. Statistics of the travelling ontology modeled in DOE, exported in RDFS and imported in several environments

we built a more intricate XSLT stylesheet[9] in order to deal with every possible serialization. We also have to specify, by hand, an XML encoding information adapted for the letter with accents. With this minor change, we are able to properly import OntoEdit and WebODE outputs. The only thing we do not get back is the relation hierarchy, which is not exported by WebODE: during the OWL import, this information is translated into logical axioms that are not serialized into `rdfs:subPropertyOf` subelements during the export. Both tools export our differential definitions, but we cannot import them properly: since they are stored in unstructured `rdfs:comment` elements, it would require string parsing to get the original structure. Consequently, we store them in a text element that is usually used in our model to store unstructured definition elements.

Things were more simple with OilEd, which uses "pure" RDFS serialization. We got our two taxonomies back, as well as the domain and range assignments for the relations. However, instances are not dealt with by OilEd's simple-RDFS export. Our differential information was forgotten too: whereas OilEd successfully gets and displays `rdfs:comment` content, it does not export it. Therefore, we lost the most valuable piece of information issued by our editor.

4.3 Conclusion

Roughly speaking, there is no loss of information when exporting our ontologies in other environments with RDFS. One may ironically insist on the fact that it is because we have little formal information to lose. However, it is very important for us that such a step be a success, since we advocate using our tool as a precondition for using other tools to further formalize ontologies.

The problems with RDFS import (in fact, OntoEdit and WebODE RDFS) is strongly linked to the fuzziness of this norm syntax. We have dealt with every

[9] It also has to deal with other alternatives, such as using `rdf:about` or `rdf:ID` to specify the name of an entity.

possible encoding for a class statement, but one may wonder whether the best solution is to question the variability of RDFS encoding.

We also have to improve the theoretical validity (with respect to our own approach) of our translations: for example, when importing a concept inheriting from multiple parents in the hierarchy corresponding to our first methodological step, we choose the "differential image" of the first parent encountered to be the parent of the "differential image" examined concept[10].

5 OWL as Exchange Language

Among the five environments we study, three of them are able to import and export OWL ontologies: Protégé2000 v2.0 beta, WebODE v2.0 and DOE v1.5. However, OilEd has the ability to export models in OWL format: to test it, we have imported an RDFS ontology in OilEd and then exported it in OWL.

5.1 From DOE To Other Environments

Our experiment with Protégé2000 has been quite successful (see Table 2). It managed to import our OWL file, getting back the two taxonomies with multiple inheritance, differential definitions, domain and range assignments, as well as instance definitions. However, some instances whose name did not follow XML specification were collapsed after their first digits being truncated. For example, reading `717` and `777`, two instances of the `BoeingPlane` concept, resulted in creating one instance whose name was an empty string.

Furthermore, all instances that are not in the Protégé namespace[11] were not imported. Consequently, the OWL export of DOE puts all instances in this namespace, which is a strange hack to allow the interoperability between these two tools.

Concerning WebODE, concept and relation hierarchies were properly imported, as were the differential definitions and domain and range assignments. However, it failed in getting back the instances, whatever namespace they are linked to.

5.2 Other Environments To DOE

When importing Protégé2000 OWL file, DOE got very limited information. It is partly due to the restrictions of our model, which is limited compared to OWL expressiveness, and partly due to the choice of XSLT. For instance, we could not import individuals, since Protégé2000 declares them using the RDFS elements `<namespace:ClassName rdf:ID="instanceID"/>` which are difficult to catch

[10] Our differential taxonomy is a tree. However, our referential one is not, which implies that there is no real loss of formal inheritance information when importing a concept with multiple parents.

[11] `http://owl.protege.stanford.edu`

	DOE	Protégé	WebODE
Number of Concepts	79	79[a]	80[b]
Number of Relations	48	48[c]	48
Number of Instances	22	20	0
Multiple Inheritance	yes	yes[d]	yes
Domains assignment	exported	preserved	preserved[e]
Ontology metadata	exported	missing	missing
Differential Definitions	exported	displayed	displayed

[a] Protégé adds 15 system classes and 19 OWL-related classes.
[b] WebODE adds the `owl:Thing` concept.
[c] Protégé adds 34 system slots and 21 OWL-related slots.
[d] Slot inheritance is not displayed in the interface, but preserved in the model.
[e] WebODE explicitly assigns `owl:Thing` to relation domains and ranges that are not defined.

Table 2. Statistics of the travelling ontology modeled in DOE, exported in OWL and imported in Protégé and WebODE

and to transform with XSLT features. We could use string-parsing features, but the result would be quite hazardous.

The problem of instance import does not appear any more when importing OilEd OWL ontologies. Since instances are serialized using the element `owl:Individual` from the OWL Presentation Syntax [7], DOE could easily get them back using an XSLT stylesheet dedicated to the translation of OWL-Presentation Syntax ontologies. But OilEd has a weird strategy in encoding the ontologies, mixing the OWL RDF-XML exchange syntax for the terminological part and the OWL Presentation Syntax for the assertional part of the model.

OWL individuals are not exported by WebODE, as well as the subsumption relation between relations (during the import, it is translated into logical axioms which are not translated back into `rdfs:subPropertyOf` subelements). However, we got back the hierarchy of concepts, together with the `rdfs:comment` including our differential definitions. We also imported the list of relations with their domain definition, but this one is incomplete if it uses the `owl:Thing` concept explicitly introduced by WebODE, an error due to an incomplete namespace declaration when using `owl:Thing` in the export file.

6 Conclusion

We have carried out the experiment proposed by the EON Workshop in order to prove that the interoperability between different ontology editors is feasible. We started from DOE, a tool implementing a methodology for building ontologies based on differential semantics. We then exported the travelling domain ontology in various environments and came back to DOE using XSLT transformations. The model of the DOE editor is very simple. It can be easily exported to other environments, which confirms our primary intuition: DOE can interoperate with them.

	Export	**Import**
Global comments	OK, put aside some string encoding and namespace problems	OK, but difficult when multiple serializations are allowed for a single fact
Concept/relation lists	OK	OK, but new concept and relation roots were added
Instances	almost OK	difficult with the abbreviated syntax
Formal definitions	OK, but limited information was exported	OK, but limited information had to be imported
Differential definitions	OK, displayed in unstructured comments fields	OK, but imported in unstructured comments fields
Ontology metadata	Dublin Core not properly dealt with by other environments	Dublin Core not dealt with by other environments

Table 3. Summary of the experiment

Using XSLT transformations in a limited expressiveness context has shown both successes and limits: we are able to export ontologies to other ontological frameworks, in a satisfactory way, the information we are interested in (the results are summarized in table 3). However, when we import ontologies built in those frameworks, we face more problems: some are linked to theoretical considerations (the status of the imported information regarding our methodology), others are linked to practical implementation shortfalls (the limited expressiveness of our formal apparatus), and others are linked to the lack of maturity of Semantic Web standards. However, the most important information is quite successfully dealt with, which comfort us in thinking that a limited but satisfying interoperability can be easily achieved by simple, syntax-based means.

Acknowledgments

We would like to thank Oscar Corcho for its helpful comments and for reviewing early drafts of this paper.

References

1. J. Angele, and York Sure (eds.). *Evaluation of Ontology-based Tools (EON'02)*, Proceedings of the 1st International Workshop EON2002, Sigüenza, Spain, CEUR-WS Publication, Vol. 62. `http://CEUR-WS.org/Vol-62/`

2. J. Arpirez, O. Corcho, M. Fernández-López, and A. Gómez-Pérez. WebODE : a Workbench for Ontological Engineering. In *Proc. of the 1st international Conference on Knowledge Capture (K-CAP'01)*, Victoria, Canada, 2001.
3. B. Bachimont, A. Isaac, and R. Troncy. Semantic Commitment for Designing Ontologies: A Proposal. In *Proc. of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02)*, Lecture Notes in Artificial Intelligence, Vol 2473, pages 114-121, Sigüenza, Spain, 2002.
4. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *Proc. of KI2001, Joint German/Austrian conference on Artificial Intelligence*, Lecture Notes in Artificial Intelligence, Vol 2174, pages 396-408, Vienna, Austria, 2001.
5. N.F. Noy, R.W. Fergerson, and M.A. Musen. The Knowledge Model of Protégé2000: Combining Interoperability and Flexibility. In *Proc. of the 12th International Conference on Knowledge Engineering and Knowledge Managment (EKAW'00)*, Juan-les-Pins, France, 2000.
6. OWL, Web Ontology Language Reference. W3C Candidate Recommendation, 18 August 2003. `http://www.w3.org/TR/owl-ref/`
7. OWL Web Ontology Language XML Presentation Syntax. W3C Note, 11 June 2003. `http://www.w3.org/TR/owl-xmlsyntax/`
8. F. Rastier, M. Cavazza, and A. Abeillé. *Sémantique pour l'analyse.* Masson, Paris, France, 1994.
9. RDF, Ressource Description Framework Primer. W3C Working Draft, 05 September 2003. `http://www.w3.org/TR/rdf-primer/`
10. RDF Schema, RDF Vocabulary Description Language 1.0. W3C Working Draft, 05 September 2003. `http://www.w3.org/TR/rdf-schema/`
11. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer and D. Wenke. OntoEdit: Collaborative Ontology Engineering for the Semantic Web. In *Proc. of the 1st International Semantic Web Conference 2002 (ISWC 2002)*, Lecture Notes in Computer Science, Vol 2342, pages 221-235, Sardinia, Italia, 2002.
12. XSLT, XSL Transformations W3C Recommendation, 16 November 1999. `http://www.w3.org/TR/xslt`

SemTalk EON2003 Semantic Web Export / Import Interface Test

Christian Fillies
Semtation GmbH
cfillies@semtalk.com

Introduction

SemTalk is a MS-Visio based graphical modelling tool, which is used for a broad range of applications as there are Business Process Modelling, SAP Product Configuration and visual glossaries. Since it is based on an open extendable meta model new modelling tools can be created with reasonable effort. Most of these solutions make use of SemTalk's ability to represent ontologies or at least taxonomies in a visual way using MS-Visio and MS-Office clipart symbols. A typical SemTalk model is being published as HTML on the intranet or it is used to generate Word or PowerPoint documentation.

SemTalk Engine

The native modelling language supported by the SemTalk [1] consistency engine is settled somewhere in the middle between RDFS and OWL. It supports multiple inheritance, instances, object- and data type properties. Graphically we follow the approach of the UML tools with boxes for classes and labelled arcs for object type properties. Data type properties are being displayed inside the rectangle of the class. We prefer this relatively compact notation in contrast to the graphical DAML notation used in VisioDAML.
We experienced the SemTalk language constructs as being as complex and powerful enough to express most of the business problems in the SemTalk application domains. The majority of users who create ontologies are domain experts and not experts for description logic. Only a minority of the resulting models is going to be interpreted by machines (except for SAP Product Configuration, which also requires additional language concepts understood by SAP Internet Pricing Configurator).

SemTalk Interfaces

In collaboration with Ontoprise GmbH we have created an F-Logic export interface to communicate with Ontobroker™ and OntoEdit™ [2].
In collaboration with Network Inference Ltd. SemTalk has been customized to cover full OWL graphically [3]. Because the SemTalk engine has been left unchanged, it can not be used to check complex expressions, disjointness or equivalence. The Network Inference product "Construct™" is designed for reasoning on the graphically created OWL model with the Cerebra™ engine.

SemTalk has export- import interfaces to RDFS and DAML. The main goal of these interfaces is to make use of existing ontologies in various SemTalk modelling scenarios. These interfaces are limited to the language subset of the SemTalk engine. E.g. a DAML disjointness axiom is being ignored by the DAML parser, DAML lists etc. are not being recognized. This limitation applies to all DAML and RDFS imports described in the following chapter. For the OWL implementation these restrictions do not apply anymore:

Full OWL can be parsed and generated. We expect significantly higher quality of the import once all tools will support OWL. It is currently not planned to complete the SemTalk DAML export. All further development will be done on OWL.

Results of the Experiment

Screenshots of the resulting models are in the appendix. The resulting models will be made available on http://www.semtalk.com

Loom	We did not try to convert the Lisp files
OilEd	After fixing some issues on the SemTalk DAML import, a subset of the model could be imported. The OildEd model differs significantly from the other models because it makes frequent use of those DAML features which are not support by SemTalk for DAML: intersectionOf, unionOf etc. On the other hand this model is quite close to OWL. We tried to rename some XML elements to OWL, but finally failed to import it mainly because of the combination of "cons"-ed Lists and operators.
OntoEdit	• Since SemTalk has only an F-Logik export and not an F-Logik import function, the flo file could not be imported. • Using DAML import classes, instances and properties could be imported. Cardinalities are ignored.
OpenKnoME	We did not try to convert the Smalltalk files
Protégé	Using RDFS import. Ignored by SemTalk RDFS Import even if the SemTalk engine could represent them: • OverridingProperty • Cardinalities • Allowed Values / Defaultvalues • All Data types • Inverse properties are mapped as properties
Terminae	We did not try to convert the text / Oil files
WebODE	Failed to import classes as rdf:description with rdf:type Class
KAON	Successful import after manually removing the XML-namespace "a:"

The overall impression from a SemTalk standpoint is, that SemTalk failed to import DAML models with complex expressions. This issue has already been fixed for OWL, which is in turn not supported by the current versions of the other tools. SemTalk succeeded in importing taxonomies from all tools, which support DAML or RDFS.

From a business point of view the lack of importing models having axioms and rich logical expressions is not very relevant since those expressions are not included in the other SemTalk methodologies such as Business Process Modelling. Being able to import taxonomies with subclassing and properties is the main point for our current customers.

Being a graphical OWL editor has not been the major goal of SemTalk in the past. The first solution for OWL is the Construct version of SemTalk developed with Network Inference early 2003. The intension of "Construct" is to replace the non-graphical OilEd by an easy-to use graphical tool.

The problems we found using the more sophisticated features of OWL in practice are, that:

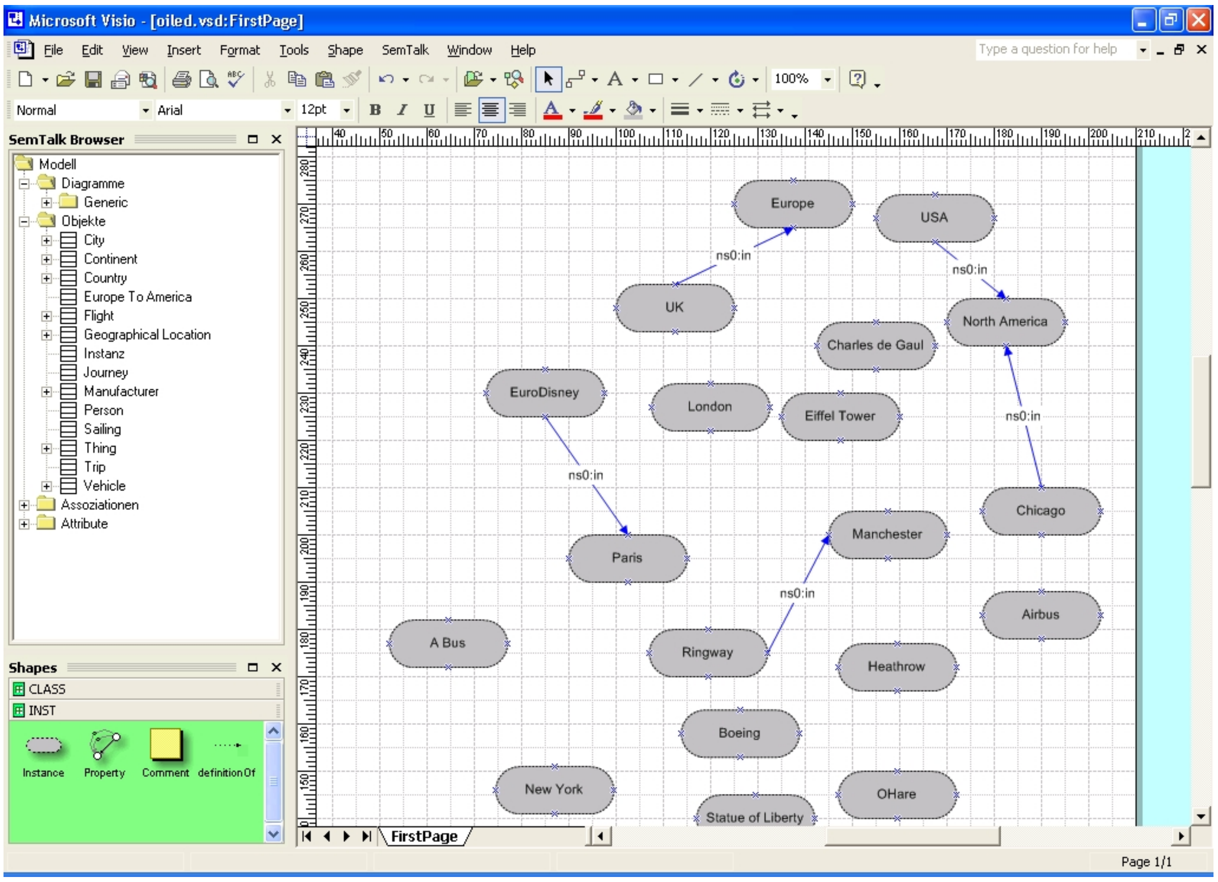
1. Most end users will not even try to understand description logic. Ontology modelling is very often ignored at all. The major problem which arises is how to use "subClassOf" properly. Concepts like disjointness, equivalence, one of etc. are not understood by casual users without further explanation.
2. A real WYSIWYG implementation with permanent and incremental consistency checking is needed to make it usable for a larger community, which none of the existing engines can provide yet.

References

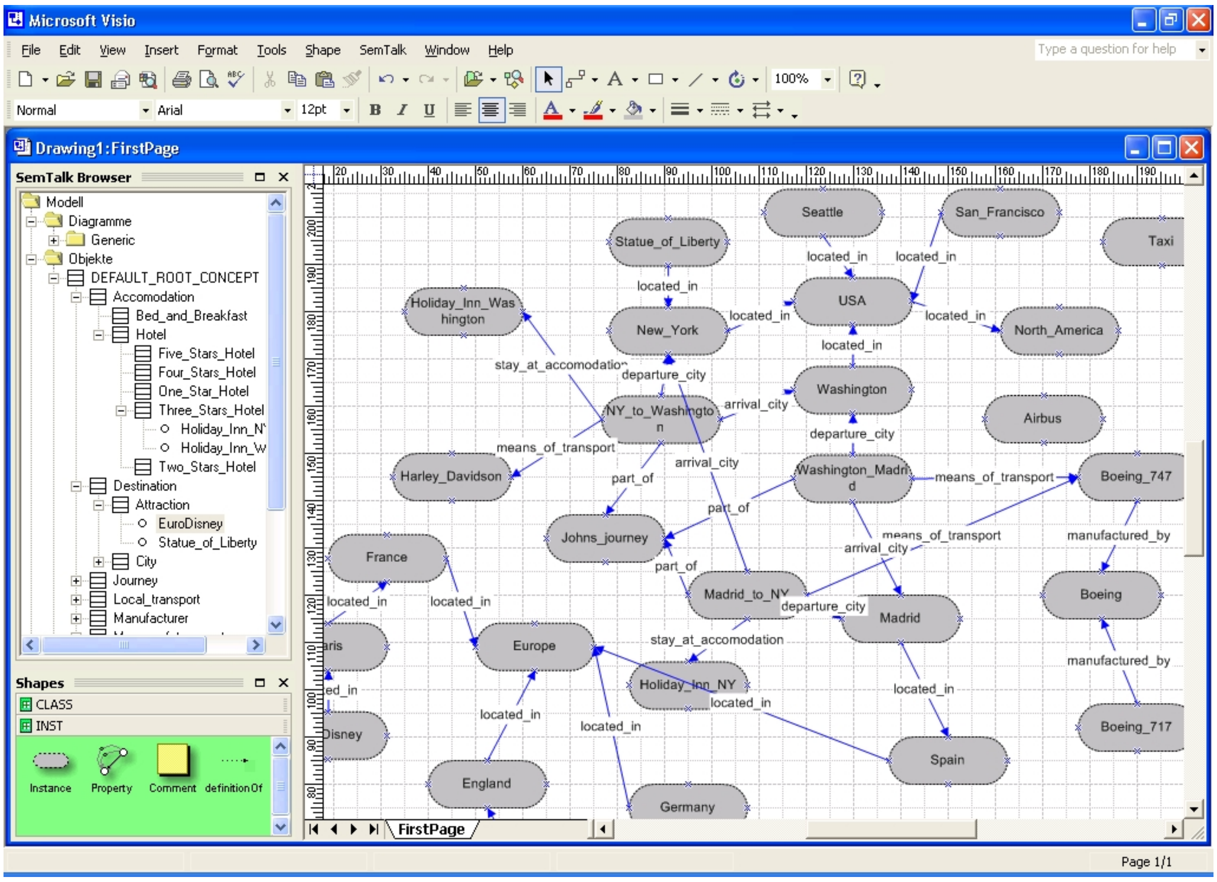
1. Fillies, C., Wood-Albrecht, G., Weichhardt, F.: A Pragmatic Application of the Semantic Web Using SemTalk, WWW2002, May 7-11, 2002, Honolulu, Hawaii, USA ACM 1-5811-449-5/02/0005
2. Fillies, C.; Sure,Y: On Visualizing the Semantic Web in MS Office, IV02 LONDON • ENGLAND
3. Fillies, C., Ng, G., Thunell, A.: Cerebra Construct: Inferences for End Users, WWW2003, May 20-24, 2002, Budapest, Hungary, Poster

Appendix Screenshots

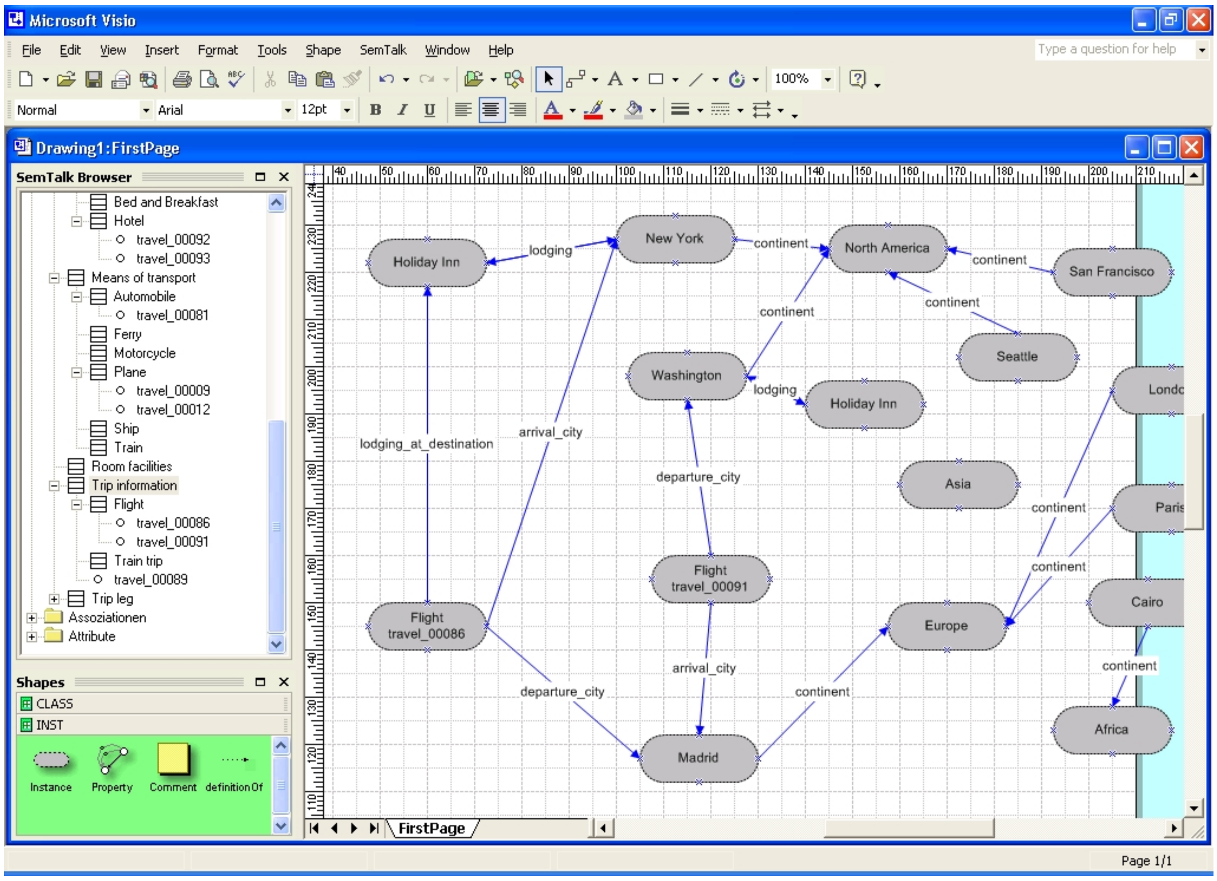
The following screenshots show results of imports from OilEd, OntoEdit, Protégé and KAON. The last screenshot shows a part of the OilEd model rebuild with the OWL shapeset made for Construct.



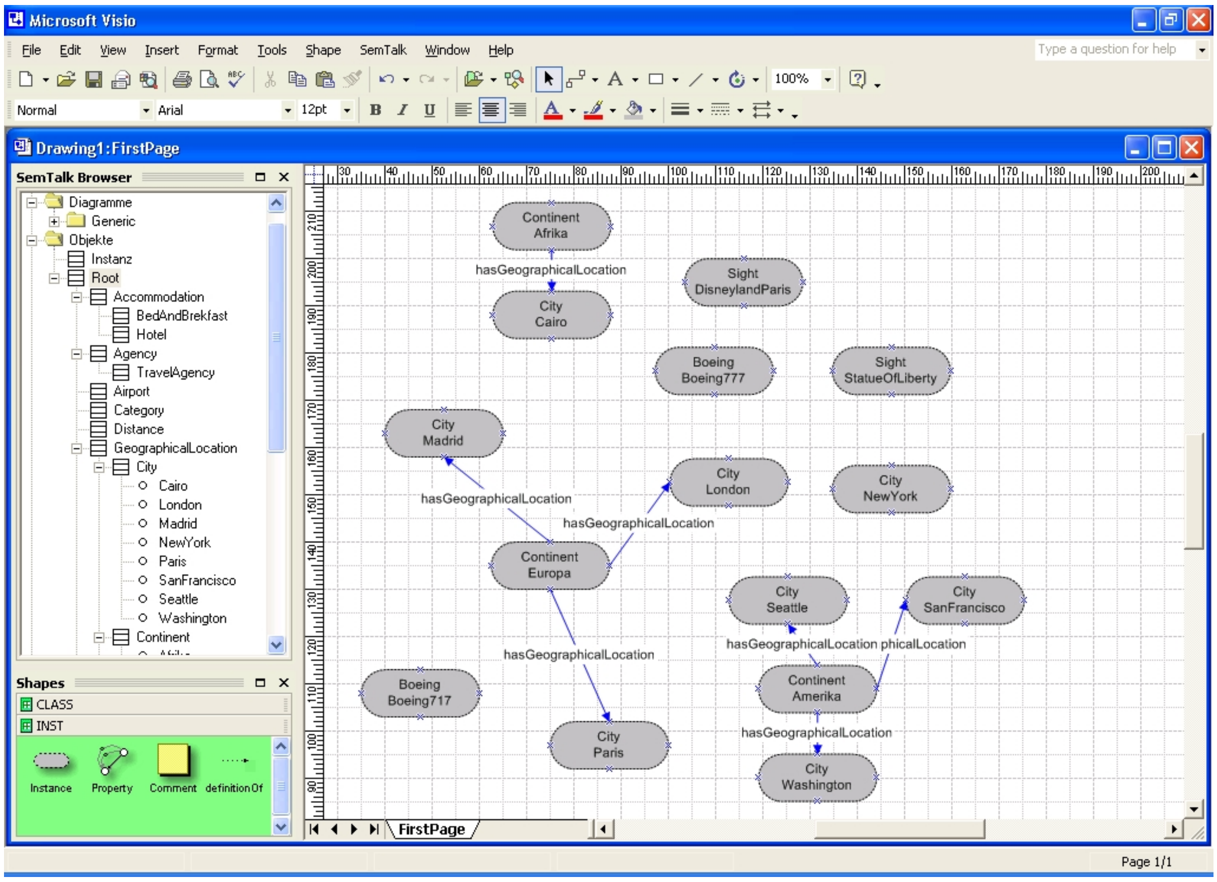
Screenshot 1 Oiled DAML Import



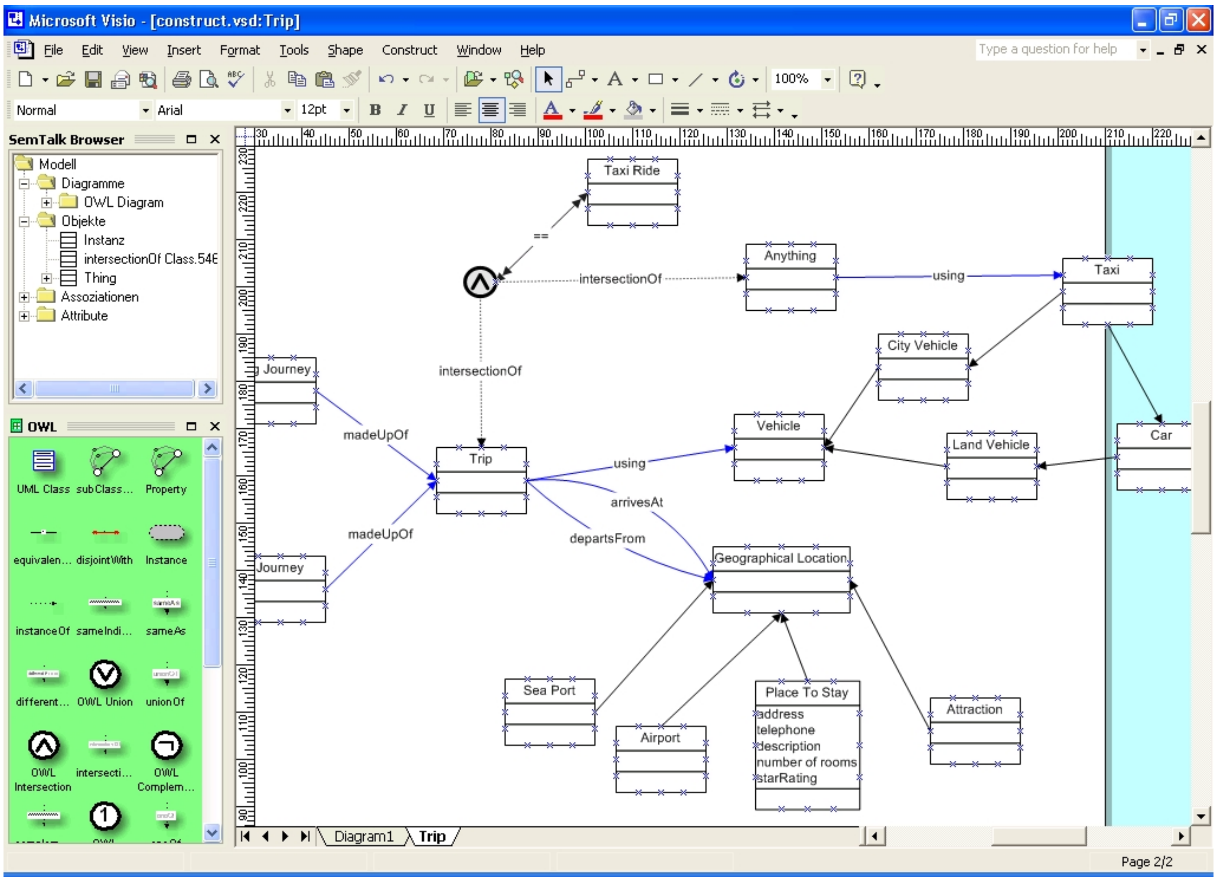
Screenshot 2 Instances of the OntoEdit DAML Import



Screenshot 3 Instances of the Protege RDFS Import



Screenshot 4 KAON DAML Import



Screenshot 5 Subset of the OilEd Model redone with the OWL Shapeset

Evaluation experiment of ontology tools' interoperability with the WebODE ontology engineering workbench

Óscar Corcho, Asunción Gómez-Pérez, Danilo José Guerrero-Rodríguez, David Pérez-Rey, Alberto Ruiz-Cristina, Teresa Sastre-Toral, M. Carmen Suárez-Figueroa(*)

Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn.
Boadilla del Monte, 28660. Madrid, Spain
(*)Contact author: mcsuarez@delicias.dia.fi.upm.es

Abstract. This paper presents the results of the interoperability experiment proposed in EON2003, using the following ontology tools: Protégé-2000 and WebODE. We will show which knowledge is preserved and which knowledge is lost in the import/export processes between tools when using RDF(S) as an intermediate language.

1 Introduction

Protégé-2000 1.8[1] [6] and WebODE 2.0[2] [4, 1] are ontology platforms which are able to import and export ontologies in different languages (RDF(S), DAML+OIL, etc.). These ontology platforms and their RDF(S) import and export services have been used in our interoperability experiment.

This document analyzes how ontologies are exchanged (exported and imported) between the previous ontology tools using RDF(S) [2, 5]. We have studied which type of knowledge is preserved and which knowledge is lost during ontology export and import in such tools. In our experiment we have reused the travel ontology built in WebODE for the EON2002 workshop [3].

2 Interoperability experiment with WebODE and Protégé-2000

In order to analyze the interoperability between WebODE and Protégé-2000, we have carried out the following process:

1. Reuse the travel ontology built in WebODE for the EON2002 Workshop [3], and export such ontology to RDF(S) using the WebODE RDF(S) export service.

[1] http://protege.stanford.edu/
[2] http://webode.dia.fi.upm.es/

2. Import this RDF(S) ontology in Protégé-2000.
3. Export the ontology from Protégé-2000 to RDF(S).
4. Import the Protégé-2000 RDF(S) ontology in WebODE, and analyze the differences between the original ontology (reused ontology) and the ontology that results from this circular import/export process.

Figure 1 shows the circular import/export process that we have carried out in the first part of our interoperability experiment.

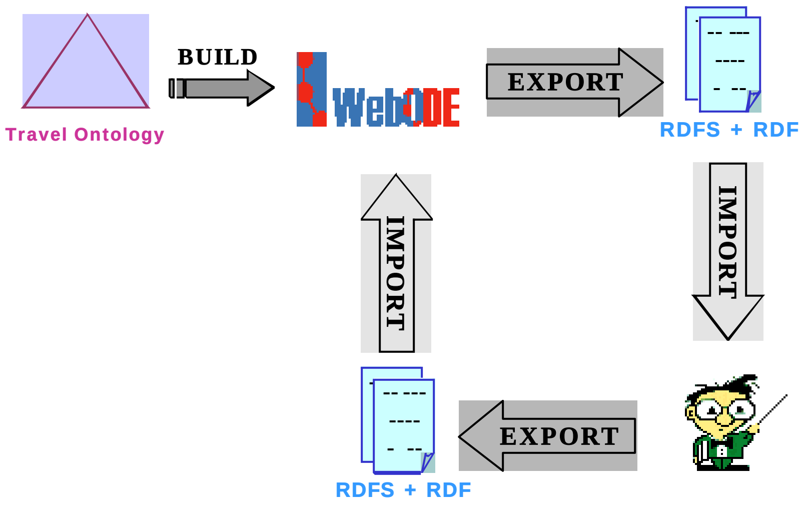


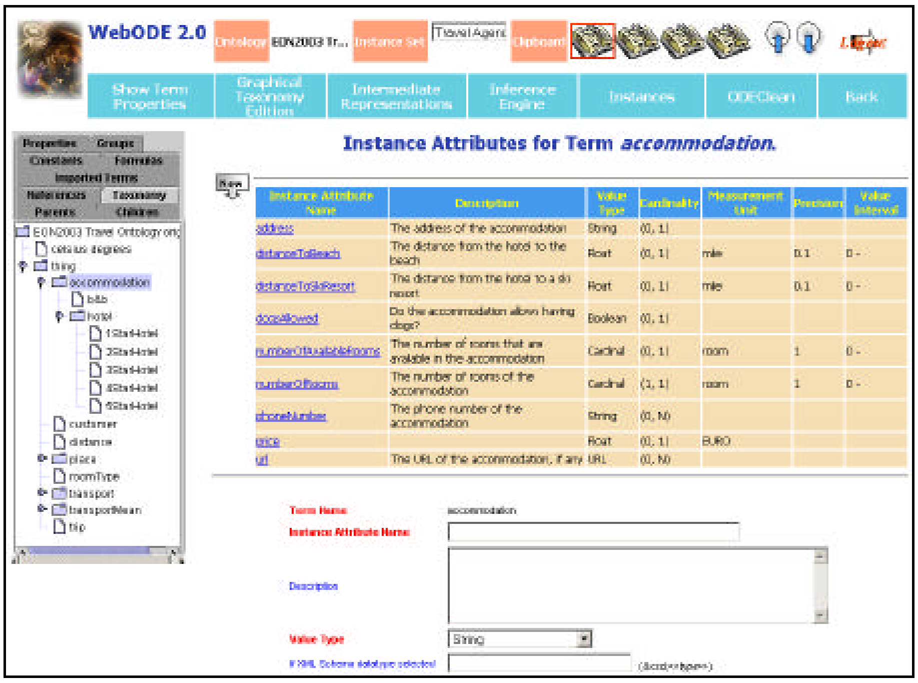
Figure 1. Circular import/export process using WebODE and Protégé-2000.

2.1 Step 1. Export to RDF(S) using WebODE

The WebODE ontology in the travel domain described in [3] and shown in figure 2 have been first exported automatically to RDF(S).
We have studied the generated RDF(S) files, and we can mention the following features:

- WebODE generates a ZIP file that contains:
  - One file for the conceptualization of the ontology (*travel_fromWebODE.rdfs* which contains the classes and properties of the ontology).
  - One file for each instance set that the user has decided to export (which contain the instances of that instance set). In our case, we have exported one of the instance sets, the one corresponding to the travel agency in New York (*travelAgencyNY_fromWebODE.rdf*).
- As a difference with the RDF(S) export function of other tools, such as Protégé-2000, WebODE does not export all the knowledge of the ontology as it is defined in the original ontology, but only those pieces of knowledge that can be directly represented with the standard knowledge model of RDF Schema. Consequently, axioms defined in the original ontology are not exported, disjoint and exhaustive

decompositions and partitions are not exported as such but as subclass-of relationships, etc.



Instance Attributes for Term *accommodation*.

Instance Attribute Name	Description	Value Type	Cardinality	Measurement Unit	Precision	Value Interval
address	The address of the accommodation	String	(0, 1)			
distanceToBeach	The distance from the hotel to the beach	Float	(0, 1)	mile	0.1	0 -
distanceToSkiResort	The distance from the hotel to a ski resort	Float	(0, 1)	mile	0.1	0 -
dogsAllowed	Do the accommodation allows having dogs?	Boolean	(0, 1)			
numberOfAvailableRooms	The number of rooms that are available in the accommodation	Cardinal	(0, 1)	room	1	0 -
numberOfRooms	The number of rooms of the accommodation	Cardinal	(1, 1)	room	1	0 -
phoneNumber	The phone number of the accommodation	String	(0, N)			
price		Float	(0, 1)	EURO		
url	The URL of the accommodation, if any	URL	(0, N)			

Term Name	accommodation
Instance Attribute Name	
Description	
Value Type	String
If XML Schema datatype selected	(&xsd;<type>)

Figure 2. Edition of instance attributes of the concept *accommodation* with the WebODE ontology editor.

- In the RDF(S) export process, the user is requested the namespace of the ontology to be exported. We have used the namespace: *http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#*. The files exported contain the following predefined namespaces for the RDF and RDFS prefixes:
  - *rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#*
  - *rdfs: http://www.w3.org/2000/01/rdf-schema#*

We have found the following problems in the exported RDF and RDFS files:

- The concept *b&b* has a different identifier than the one used in WebODE, as follows:

```
<rdf:Description rdf:about='#b&amp;b'>
```

- The relation *usesTransportMean*, which is defined in WebODE between the following pairs of concepts: *(carRented, car)*, *(cityBus, bus)*, *(flight, airTransportMean)*, *(undergroundTransport, underground)*, *(transport, transportMean)*, is defined only once in the generated RDFS file. This is due to the fact that RDF does not allow homonymous property names. Besides, in RDFS this property does not have its domain nor its range defined.

- The same applies to class and instance attributes, which are necessarily attached to a concept in WebODE, so that we can have different attributes with the same name in different concepts. For instance, the class attribute *numberOfStars* is defined once in the RDFS file, while it is defined for five classes in WebODE (*1StarHotel*, *2StarHotel*, etc.). In this case, neither the domain nor the range are specified in the RDFS file.
- Finally, since the RDF(S) export function was developed when the treatment of datatypes was not clear in the RDFS specification, the current RDF(S) export function converts all the types of WebODE instance and class attributes to *rdfs:Literal*.
- WebODE constants are transformed into concepts in RDF(S). For instance, the constant *celsius degrees* is transformed into the concept *celsius_degrees*. Consequently, it loses its value.

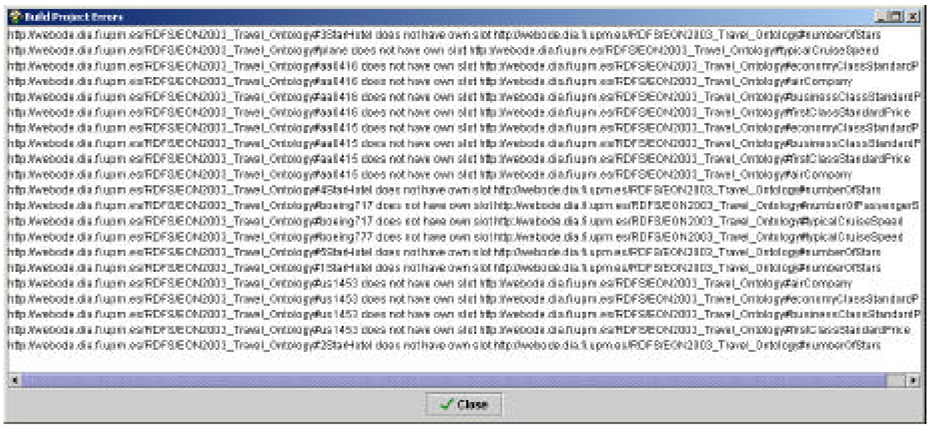
2.2 Step 2. Import the RDF(S) files generated by WebODE into Protégé-2000

We have imported into Protégé-2000 the RDF and RDFS files generated in the previous stage of our experiment. During the import process, the following comments have been provided by Protégé-2000:

- Protégé-2000 has recognized four namespaces in the ontologies imported:
  - *rdf*, *rdfs*, and the base namespace of the ontology
  - One additional namespace that appears as the value of a property for a hotel: http://holidayinn.com.

```
<NS0:url rdf:resource='http://holidayinn.com/13492'/>
```

- Besides, the values of class attributes that were exported from WebODE to RDF(S) are not correctly imported (e.g., the number of stars of a hotel, the air company in charge of a flight, etc.). Protégé-2000 shows a warning that alerts the user that this "own slot" has not been defined in a metaclass, as shown in figure 3. Consequently, this information is lost.

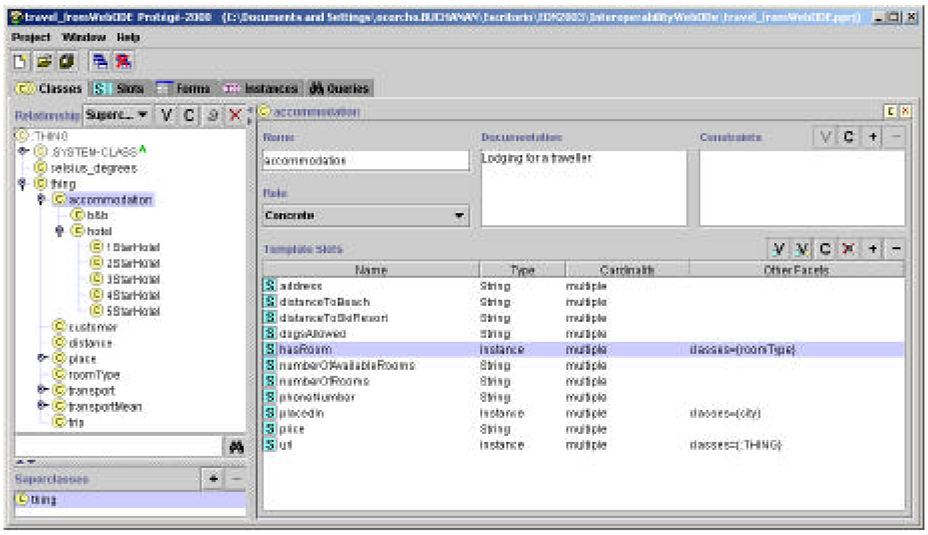


```
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#3StarHotel does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#numberOfStars
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#plane does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#typicalCruiseSpeed
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#aa1416 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#economyClassStandardP
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#aa1416 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#airCompany
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#aa1416 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#businessClassStandardP
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#aa1416 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#firstClassStandardPrice
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#aa1415 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#economyClassStandardP
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#aa1415 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#businessClassStandardP
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#aa1415 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#firstClassStandardPrice
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#aa1415 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#airCompany
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#4StarHotel does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#numberOfStars
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#boeing717 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#numberOfPassengerS
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#boeing717 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#typicalCruiseSpeed
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#boeing777 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#typicalCruiseSpeed
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#5StarHotel does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#numberOfStars
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#1StarHotel does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#numberOfStars
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#us1453 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#airCompany
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#us1453 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#economyClassStandardP
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#us1453 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#businessClassStandardP
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#us1453 does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#firstClassStandardPrice
http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#2StarHotel does not have own slot http://webode.dia.fi.upm.es/RDFS/EON2003_Travel_Ontology#numberOfStars
```

Figure 3. Own slots' import problem with Protégé-2000.

The result of the import process is shown in figure 4. There we can see the details of the concept *accommodation*, whose template slots are the same as those defined in WebODE (except for *hasRoom* and *placedIn*, which were defined as relations in WebODE). However there are some differences between these attributes and relations, which are related to their cardinalities and types. As a result of using RDFS as an exchange language, we have lost the cardinality information for template slots. Additionally, the types "integer", "Boolean", etc., have been transformed to "String" in Protégé-2000, since they were transformed by WebODE to *rdfs:Literal*. Finally, the type of the slot *ur*" is "Instance", of the class *:THING*, as it was transformed to a property whose range was *rdfs:Resource* by WebODE.

After the import process, we have compared the WebODE ontology and the Protégé-2000 ontology (shown in figure 4), finding the following differences:



accommodation

Name: accommodation

Documentation: Lodging for a traveller

Constraints

Role: Concrete

Template Slots

Name	Type	Cardinality	Other Facets
address	String	multiple	
distanceToBeach	String	multiple	
distanceToSkiResort	String	multiple	
dogsAllowed	String	multiple	
hasRoom	Instance	multiple	classes={roomType}
numberOfAvailableRooms	String	multiple	
numberOfRooms	String	multiple	
phoneNumber	String	multiple	
placedIn	Instance	multiple	classes={city}
price	String	multiple	
url	Instance	multiple	classes={:THING}

Figure 4. Travel ontology in Protégé-2000.

- Attributes whose type was "integer" or "Boolean" in WebODE have changed in Protégé-2000 to type "String". This is due to the fact that the RDFS file already contained a transformation of these basic types to *rdfs:Literal*.
- The cardinalities of attributes have changed. All of them have 0 as a minimum cardinality and N as a maximum cardinality (that is, they are defined as "multiple").
- The class attributes defined in WebODE have disappeared, because of the own slot problem described in the import process.
- The attributes with multiple documentations (multiple *rdfs:label* properties attached) have now one single documentation that joins all of them.
- The knowledge about disjoint and exhaustive decompositions, and partitions is lost in Protégé-2000, since it was not available in the RDF(S) files. The same applies to axioms, concept groups, constants, etc.

- The values of the attribute *url* for two of the instances have been transformed to instance themselves, as instances of the class *:THING*. In WebODE and RDF(S) they were just URIs.

Since Protégé-2000 is not able to work with different instance sets at the same time, we have been only able to import one of the instance sets that could be exported by WebODE.

2.3 Step 3. Export the Protégé-2000 ontology to RDF(S)

Finally, we have exported the Protégé-2000 ontology to RDF(S) and we have obtained two files, one for the classes and another one for the instances. There are many differences (mainly syntactic) between the original RDF(S) files and the target RDF(S) files generated, as can be seen by simply comparing the four files.

2.4 Step 4. Import the RDF(S) ontology generated by Protégé-2000 into WebODE

In order to import the ontology into WebODE, we have had to join the two files generated by Protégé-2000 into only one file that contains both the ontology conceptualization and the instances. This file is called *Travel_fromProtegetoWebODE.rdf_s*.

In this import process we have found the following problems:

- Protégé-2000 uses a namespace for the RDFS KR ontology that comes from an old specification: *http://www.w3.org/TR/1999/PR-rdf-schema-19990303#*. This causes the WebODE RDF(S) import function to not correctly detect the concepts defined in the ontology. Consequently, we have edited the file manually so as to change this namespace by the following: *http://www.w3.org/2000/01/rdf-schema#*.
- The concepts whose identifier starts with a digit have not been imported correctly. As a consequence, we had to rename manually the terms *1StarHotel*, *2StarHotel*, *3StarHotel*, *4StarHotel*, and *5StarHotel*.
- The same applies to the instances whose identifier starts with a digit. In this case, the WebODE import function notifies the following error:

  "Error importing RDFS ontology: Error occurred in server thread; nested exception is: com.hp.hpl.mesa.rdf.jena.model.RDFError: org.xml.sax.SAXParseException: An invalid second ':' was found in the element type or attribute name."

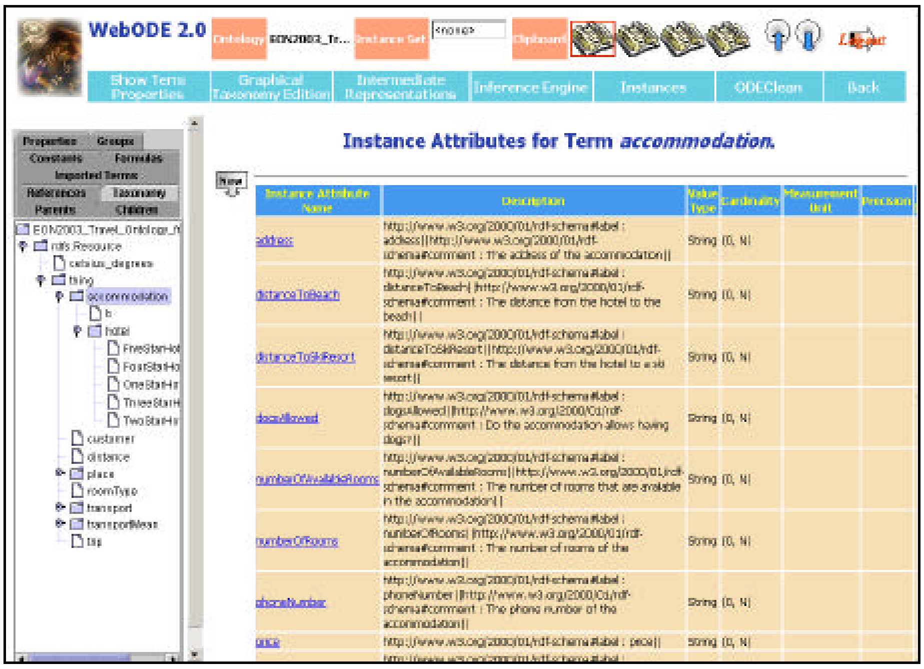
  which is not much descriptive about the problem in the source RDF(S) file.

In this case, we have compared the original ontology built in WebODE and the resulting ontology of importing the RDF(S) of Protégé-2000 in WebODE (shown in figure 5).

We have found the following differences in our comparison:

- A new concept is generated in WebODE (*rdfs:Resource*) which is used as the root concept of the ontology.

• New relations, which did not exist in the original ontology, appear in the imported ontology. These relations were represented as attributes of type *URL* in the original ontology. Since they were transformed into slots with range *:THING*, and transformed back to RDFS as properties with range *rdfs:Resource*, they have not been recovered as originally during the last imp ort process.



Instance Attributes for Term *accommodation*.

Instance Attribute Name	Description	Value Type	Cardinality	Measurement Unit	Precision
address	http://www.w3.org/2000/01/rdf-schema#label : address||http://www.w3.org/2000/01/rdf-schema#comment : The address of the accomodation||	String	(0, N)		
distanceToBeach	http://www.w3.org/2000/01/rdf-schema#label : distanceToBeach||http://www.w3.org/2000/01/rdf-schema#comment : The distance from the hotel to the beach||	String	(0, N)		
distanceToSkiResort	http://www.w3.org/2000/01/rdf-schema#label : distanceToSkiResort||http://www.w3.org/2000/01/rdf-schema#comment : The distance from the hotel to a ski resort||	String	(0, N)		
dogsAllowed	http://www.w3.org/2000/01/rdf-schema#label : dogsAllowed||http://www.w3.org/2000/01/rdf-schema#comment : Do the accommodation allows having dogs?||	String	(0, N)		
numberOfAvailableRooms	http://www.w3.org/2000/01/rdf-schema#label : numberOfAvailableRooms||http://www.w3.org/2000/01/rdf-schema#comment : The number of rooms that are available in the accommodation||	String	(0, N)		
numberOfRooms	http://www.w3.org/2000/01/rdf-schema#label : numberOfRooms||http://www.w3.org/2000/01/rdf-schema#comment : The number of rooms of the accommodation||	String	(0, N)		
phoneNumber	http://www.w3.org/2000/01/rdf-schema#label : phoneNumber||http://www.w3.org/2000/01/rdf-schema#comment : The phone number of the accommodation||	String	(0, N)		
price	http://www.w3.org/2000/01/rdf-schema#label : price||	String	(0, N)		
	http://www.w3.org/2000/01/rdf-schema#label :				

Figure 5. Travel ontology imported from Protégé-2000 RDF(S).

• The concept *b&b* (bed and breakfast) has been transformed to *b*, because of the symbol &.
• The documentation of concepts, attributes, relations, etc., now have more text: they include the term label (as defined in the Protégé-2000 RDF(S) files) and the comment, which was the original documentation.
• The cardinalities and types of the instance and class attributes are different from those that were originally present in WebODE. This knowledge was lost in the first step.
• All the information that was already lost in the first export process is, of course, missing: disjoint and exhaustive decompositions, partitions, axioms, etc.
• Relations with the same name represented in WebODE (e.g., *usesTransportMean*) are now transformed into a unique relation whose domain is *rdfs:Resource*.
• Class and instance attributes with the same name represented in WebODE (e.g., *airCompany*) are now transformed into a unique relation whose domain and range is *rdfs:Resource*. This is due to the fact that their domain was not exported to RDF(S) in step 1.

3 Conclusions

The table 1 summarizes the main conclusions of this circular import/export processes, with the number of ontology components that can be found in each of the ontologies generated during the process. We do not care about other issues, such as differences in the domains, ranges, cardinalities, term names, etc.

	WebODE	**RDF(S) (step 1)**	**Protégé-2000 (step 2)**	**RDF(S) (step 3)**	**WebODE (step 4)**
#concepts	62	62	62[3]	62	**63**
#subclass of	24	**61**	63	63	63
#disjoint decompositions	6	**0**	0	0	0
#exhaustive decompositions	0	**0**	0	0	0
#partitions	3	**0**	0	0	0
#attributes/relations	69	**43**	43	43	**44**
#axioms	8	**0**	0	0	0
#constants	1	**0**	0	0	0
#instances	20	20	**22**	22	**20**

Table 1. Summary of knowledge preserved and lost during the circular import/export process

The most relevant comments that can be extracted from the previous table are the following:

- WebODE creates a new concept when importing ontologies from RDF(S). This class is *rdfs:Resource*, which is used as the root concept of all the ontology concepts, and is also used as the domain and/or range of several ad hoc relations for which the domain/range has not been defined explicitly in the RDFS file.
- With regard to the taxonomic relationships between concepts, we have two comments:
  - WebODE is able to represent disjoint and exhaustive knowledge in its concept taxonomies. However, with RDFS we cannot represent this kind of knowledge, and consequently it is transformed into simple subclass of relationships. This is the reason why there are 24 subclass of relationships defined in the original ontology, and they are transformed into 61 in the RDF(S) file and successive transformations.
  - Besides, when importing the ontology from RDF(S) to Protégé-2000 two new subclass of relationships appear. These are related to the use of the class *:THING* as the root class of any Protégé-2000 ontology. As a consequence, the classes *thing* and *celsius_degrees* from the original ontology are explicitly declared as subclasses of *:THING*.

[3] This figure does not include the system classes that are always generated by Protégé-2000

- The number of attributes and relations that are present in the original ontology is quite different than that of the ontology generated in RDF(S) and obtained in the subsequent processes. The reason for this is that WebODE allows representing different attributes and relations for different concepts with the same name. This is not allowed neither in RDF(S) nor in Protégé-2000. Consequently, in the transformation, attributes and relations with the same name are transformed into only one attribute/relation.
- We have discovered an error in the import process of WebODE with the RDF(S) property *url*, whose range is *rdfs:Resource*. This property is transformed into an attribute of type *URL* and a relation between the concept accommodation and the concept *rdfs:Resource*.
- Axioms and constants are lost in the transformation to RDF(S), since they cannot be represented in this language.
- Finally, the number of instances is constant, except for the import to Protégé-2000, in which instances are created for two resources that appear as the range of the property *url* (holidayInn hotels' URLs), and except for the import to WebODE, where these instances are lost since they are instances of *rdfs:Resource*.

Acknowledgments

This work has been supported by the OntoWeb thematic network (IST-2000-29243), by the research grant AP2002-3838 from MEC, and by a research grant from UPM ("Beca asociada a proyectos modalidad B").

References

1. Arpírez JC, Corcho O, Fernández-López M, Gómez-Pérez A (2003) *WebODE in a nutshell*. AI Magazine 24(3):37:48
2. Brickley D, Guha RV (2003) *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Working Draft. http://www.w3.org/TR/PR-rdf-schema
3. Corcho O, Fernández-López M, Gómez-Pérez A (2002) *Evaluation experiment for the editor of the WebODE ontology workbench*. In: Angele J, Sure Y (eds) EKAW02 Workshop on Evaluation of ontology-based tools (EON2002). Sigüenza, Spain
4. Corcho O, Fernández-López M, Gómez-Pérez A, Vicente O (2002) *WebODE: an Integrated Workbench for Ontology Representation, Reasoning and Exchange*. In: Gómez-Pérez A, Benjamins VR (eds) 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02). Sigüenza, Spain. (Lecture Notes in Artificial Intelligence LNAI 2473) Springer-Verlag, Berlin, Germany, pp 138–153
5. Lassila O, Swick R (1999) *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. http://www.w3.org/TR/REC-rdf-syntax/
6. Noy NF, Fergerson RW, Musen MA (2000) *The knowledge model of Protege-2000: Combining interoperability and flexibility*. In: Dieng R, Corby O (eds) 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00). Juan-Les-Pins, France. (Lecture Notes in Artificial Intelligence LNAI 1937) Springer-Verlag, Berlin, Germany, pp 17–32

Case Study: Using Protégé to Convert the Travel Ontology to UML and OWL

Holger Knublauch

Stanford Medical Informatics
Stanford University
MSOB x-215
251 Campus Drive
Stanford, CA 94305-5479
holger@smi.stanford.edu
WWW home page: http://www.knublauch.com

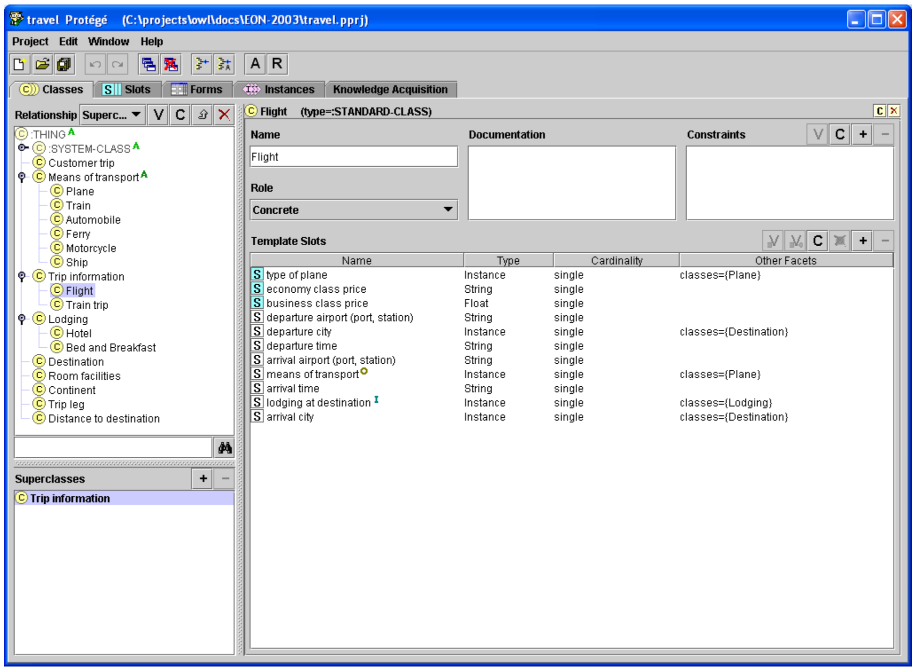
Abstract. Our goal was to evaluate the import/export capabilities of Protégé between various ontology file formats. As a starting point, we chose the Travel ontology used for the Protégé experiment from the previous EON workshop. We exported this into UML, from where we could import most of the ontology into the mainstream software development tool Poseidon. Furthermore, we exported the ontology into OWL. The resulting OWL file could be processed by the OWL Species Validator. All transformations maintained the structure of the ontology without problems but could not handle all of the model semantics correctly.

1 Introduction

Protégé (http://protege.stanford.edu) is one of the most widely used ontology editors with currently about 10,000 registered users. Its extensible open-source platform supports several ontology file formats including CLIPS (Protégé's native format), various XML dialects, databases, DAML+OIL and RDF(S). Very recently, storage plugins for the Unified Modeling Language (UML) and the Web Ontology Language (OWL) have been added. Both plugins are not complete yet and will evolve during the following months.

This document reports on a simple experiment with the UML and OWL Plugins. We wanted to test whether Protégé can convert a given ontology into these formats and to get an idea of which information are getting lost during conversion. Our starting point is the Travel Ontology developed by Natasha F. Noy as described in her contribution to the previous EON workshop. A screenshot of this ontology (displayed in Protégé) is shown in figure **??**.

The experiment was performed using the most recent alpha release of Protégé 2.0 (build 42). Older versions (starting with version 1.8) would expose the same behavior for the UML conversion. However, these versions do not support the OWL Plugin.



:THINGA
:SYSTEM-CLASSA
Customer trip
Means of transportA
Plane
Train
Automobile
Ferry
Motorcycle
Ship
Trip information
Flight
Train trip
Lodging
Hotel
Bed and Breakfast
Destination
Room facilities
Continent
Trip leg
Distance to destination

Superclasses: Trip information

Flight (type=:STANDARD-CLASS)

Name: Flight

Role: Concrete

Template Slots

Name	Type	Cardinality	Other Facets
type of plane	Instance	single	classes={Plane}
economy class price	String	single	
business class price	Float	single	
departure airport (port, station)	String	single	
departure city	Instance	single	classes={Destination}
departure time	String	single	
arrival airport (port, station)	String	single	
means of transport	Instance	single	classes={Plane}
arrival time	String	single	
lodging at destination	Instance	single	classes={Lodging}
arrival city	Instance	single	classes={Destination}

Fig. 1. The original ontology (CLIPS format) edited with Protégé.

2 UML Export and Import

UML is one of the best known modeling languages for real-world projects. There have been several attempts to exploit UML for ontology modeling so that mainstream tools can be used for knowledge modeling. The Object Management Group (OMG) has recently issued a call for proposals for a UML-based ontology language which will boost interest in ontology design among software developers. In order to provide some interoperability between Protégé and UML tools, the UML Plugin has been developed in February 2003. Since then, it has been adopted into routine use by many users.

The Protégé knowledge model (OKBC) and UML allow very similar constructs. Most obviously, the following conversions exist:

– UML classes can be compared to OKBC classes
– UML objects are similar to OKBC instances
– UML attributes and relationships are comparable to OKBC slots

However, there is a significant area of language elements that are incompatible. Most notably, Protégé supports a native constraint language called PAL, whereas UML uses its Object Constraint Language (OCL). Both have a similar structure but a converter does not exist yet.

Another difference is that Protégé supports generic facet overloading, which means that you can redefine slot properties (such as value type, cardinality and default values) for certain classes. This reflects a major difference between UML and OKBC, namely that in OKBC, slots are first-class elements and can exist without being assigned to a class, whereas UML attributes and relationships must be assigned to classes. Protégé's UML Plugin is able to handle this difference. For example, it creates multiple copies of an attribute if a slot is attached to more than one class. It fails however with complex facet overloads, because a comparable concept does not exist in UML.

Other differences between UML and OKBC include the handling of metaclasses (which is much more flexible in Protégé) and support for instances. Although UML officially has the concept of Object Diagrams, few tools support it properly, and so the UML Plugin does not export instances. There is however no reason why this should not be supported in future versions.

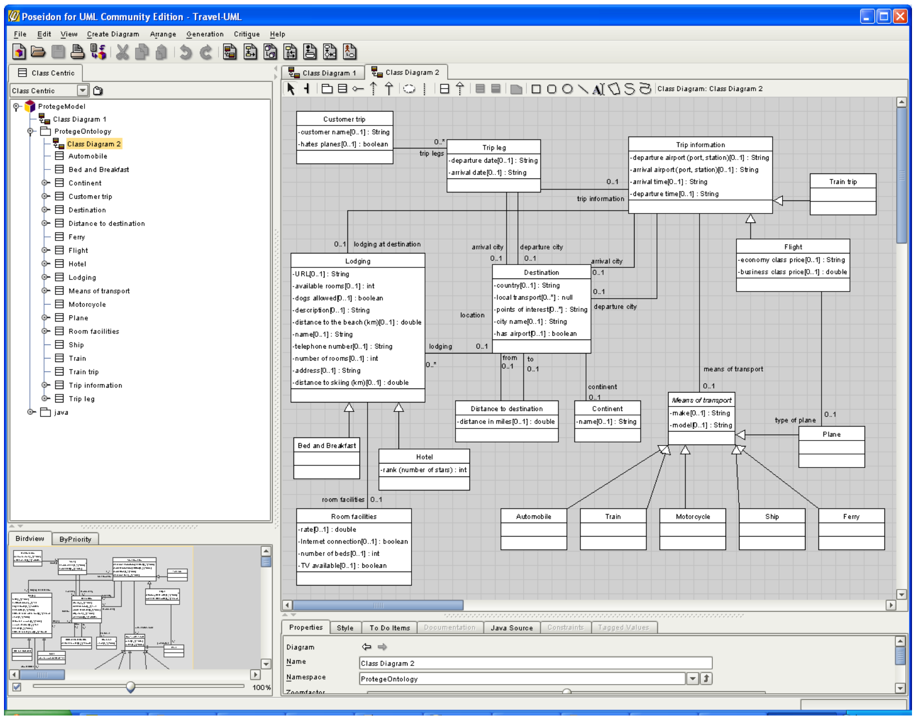

Poseidon for UML Community Edition - Travel-UML
File Edit View Create Diagram Arrange Generation Critique Help
Class Centric
ProtegeModel
Class Diagram 1
ProtegeOntology
Class Diagram 2
Automobile
Bed and Breakfast
Continent
Customer trip
Destination
Distance to destination
Ferry
Flight
Hotel
Lodging
Means of transport
Motorcycle
Plane
Room facilities
Ship
Train
Train trip
Trip information
Trip leg
java
Class Diagram 1
Class Diagram 2
Class Diagram: Class Diagram 2
Customer trip
-customer name[0..1] : String
-hates planes[0..1] : boolean
Trip leg
-departure date[0..1] : String
-arrival date[0..1] : String
Trip information
-departure airport (port, station)[0..1] : String
-arrival airport (port, station)[0..1] : String
-arrival time[0..1] : String
-departure time[0..1] : String
Train trip
trip legs
trip information
lodging at destination
arrival city
departure city
Lodging
-URL[0..1] : String
-available rooms[0..1] : int
-dogs allowed[0..1] : boolean
-description[0..1] : String
-distance to the beach (km)[0..1] : double
-name[0..1] : String
-telephone number[0..1] : String
-number of rooms[0..1] : int
-address[0..1] : String
-distance to skiing (km)[0..1] : double
Destination
-country[0..1] : String
-local transport[0..*] : null
-points of interest[0..*] : String
-city name[0..1] : String
-has airport[0..1] : boolean
Flight
-economy class price[0..1] : String
-business class price[0..1] : double
location
lodging
from
to
continent
means of transport
type of plane
Distance to destination
-distance in miles[0..1] : double
Continent
-name[0..1] : String
Means of transport
-make[0..1] : String
-model[0..1] : String
Plane
Bed and Breakfast
Hotel
-rank (number of stars) : int
room facilities
Room facilities
-rate[0..1] : double
-Internet connection[0..1] : boolean
-number of beds[0..1] : int
-TV available[0..1] : boolean
Automobile
Train
Motorcycle
Ship
Ferry
Birdview
ByPriority
Properties
Style
To Do Items
Documentation
Java Source
Constraints
Tagged Values
Diagram
Name
Class Diagram 2
Namespace
ProtegeOntology
Zoomfactor
100%

Fig. 2. The ontology exported with Protégé in UML format opened with Poseidon for UML.

For the given travel ontology, most of the structural information from the ontology could be preserved. As shown in figure **??**, the resulting UML file (in

XMI format) could be loaded with the well-known UML modeling tool Poseidon. Since not all CASE tools support the XMI standard equally well, it might not be possible to load UML files generated with Protégé into all tools. This shortcoming is however due to different interpretations of UML/XMI standards by third-party tools, while Protégé supports the official UML specification.

Note that Protégé can also re-import UML files that have been changed with an external tool. In this step it will also combine multiple namesake attributes into a single slot, etc.

The following information got lost during the translation:

– PAL Constraints
– Facet overloads (there were 4 of them in the original ontology)

The allowed values of symbol slots are exported correctly in the XMI file, but not displayed by the UML tool so that the datatype of some attributes is "null". This is a bug in Poseidon.

While UML and OKBC each provide different modeling elements, they are both extensible and thus allow for a complete round-trip mapping. Protégé's generic metamodeling architecture can be used to define new metaclasses which capture UML-specific items such as methods and OCL expressions. This has been partially implemented so that Protégé can also be used to define class methods. UML has a number of extension mechanisms, such as stereotypes and tagged values, which can be used to store Protégé-specific data for round-tripping.

The rather awful problem with the current UML specification (before 2.0) is that there is no standard exchange format for diagrams. This means that users need to re-layout their class diagrams each time when it has been changed.

3 OWL Export and Import

Work on the OWL Plugin for Protégé started in April 2003 and is not finished yet. Therefore the following results are preliminary (and might have changed at the workshop time). Protégé relies on the Jena API, a leading Java-based API for OWL and RDF. Since this software is also still in alpha state, not all features are implemented yet.

As shown in figure **??**, Protégé and OWL each support constructs that are not available in the other. A major difference is that OWL supports arbitrary class descriptions, whereas Protégé only knows primitive named classes. We have extended Protégé's metamodel to express these additional language elements. More details on this mapping can be found on the OWL Plugin web site (http://protege.stanford.edu/plugins/owl).

The current version of the OWL plugin allows to load arbitrary OWL (DL) files into Protégé. Some elements of OWL Full, especially metaclasses, can be represented. Protégé maintains a copy of the OWL model using the Jena API, and changes in the Protégé model are synchronized with the OWL objects. This technology ensures that all language elements that Protégé does not support in

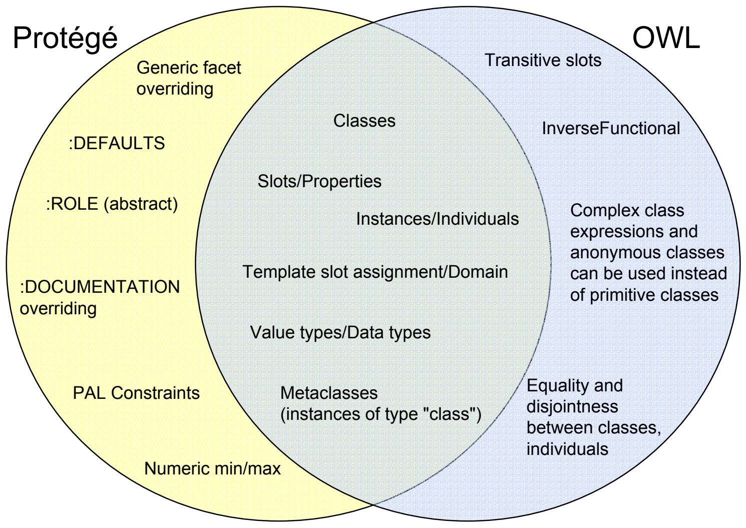

Protégé
OWL
Generic facet overriding
:DEFAULTS
:ROLE (abstract)
:DOCUMENTATION overriding
PAL Constraints
Numeric min/max
Classes
Slots/Properties
Instances/Individuals
Template slot assignment/Domain
Value types/Data types
Metaclasses (instances of type "class")
Transitive slots
InverseFunctional
Complex class expressions and anonymous classes can be used instead of primitive classes
Equality and disjointness between classes, individuals

Fig. 3. The language elements of Protégé and OWL in comparison.

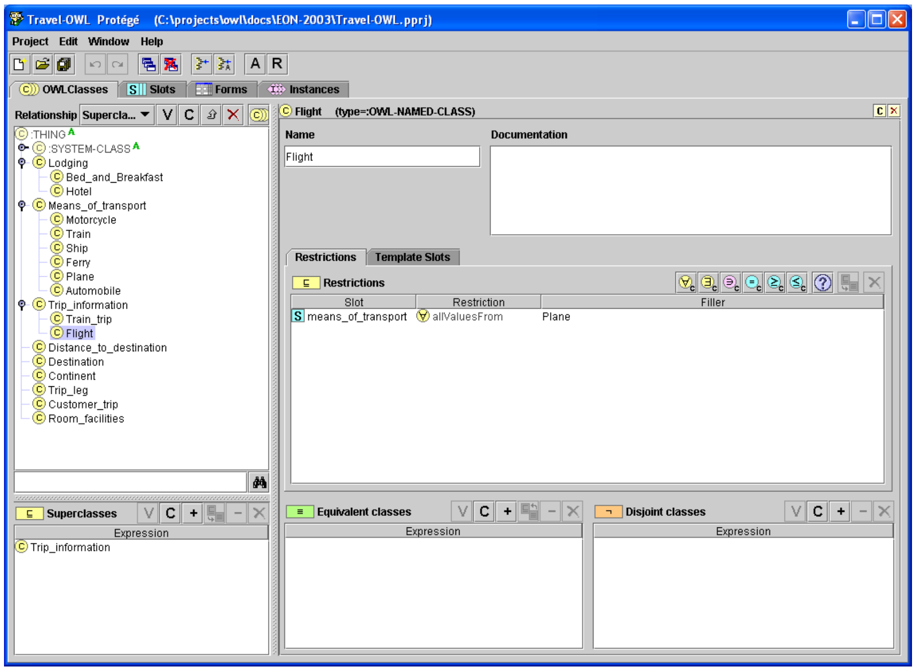
its own metaclass hierarchy at least remain untouched when saved back to a file. Editing OWL files with Protégé is therefore lossless.

The example travel ontology could be converted into Protégé without problems. As shown in figure **??**, facet overloads are automatically converted into OWL restrictions (here: An allValuesFrom restriction). The only information that currently gets lost is Protégé-specific elements such as PAL constraints.

The OWL files created by Protégé obey the recent OWL standard specification and can be loaded by external OWL tools such as the OWL Species Validator. However, due to the lack of other ontology tools with OWL support, we could not seriously test advanced issues such as round-tripping between tools.

4 Discussion and Future Work

The simple case studies show that Protégé is a suitable platform for interchanging models in standard languages such as UML and OWL. Both languages play a central role in two huge communities that are traditionally not counted as ontology builders: Mainstream Software Engineering and the Semantic Web, respectively. The wide adoption of Protégé's support for these languages has



Name: Flight

Slot	Restriction	Filler
means_of_transport	allValuesFrom	Plane

Superclasses: Trip_information

Fig. 4. The ontology in OWL format edited with Protégé.

shown us how important they are and that ontology construction could play a much more important role in these communities.

Both examples also demonstrate the flexibility of the OKBC knowledge model. OKBC provides a very flexible metamodeling architecture that can be easily extended to capture other languages than those natively supported by Protégé. With an extended metamodel in place, one only needs to adapt the user interface to get a custom-tailored modeling tool for almost any language. Several specific editor components have been implemented for OWL.

In support of true round-trip engineering – which is crucial for real world projects – the tools should make sure that one tool's language specific data is not lost when opened with another tool. We have not fully implemented these capabilities due to lack of time. Currently, Protégé-specific information that does not have a direct counterpart in OWL or UML is getting lost. There are however no reasons why this should not be possible in the future.

EXPERIMENT:

Interoperability of Protégé 2.0 beta and OilEd 3.5 in the Domain Knowledge of Osteoporosis

Franz Calvo, MD fcalvo@u.washington.edu and John H. Gennari, PhD gennari@u.washington.edu
Department of Medical Education and Biomedical Informatics, School of Medicine. University of Washington, Seattle, WA, USA

I. Introduction

Because the idea of building a single, overall ontology for the entire Semantic Web seems impossible, we believe that integration of the various standards and ontology building tools is an important goal. However, the lack of interoperability between the different knowledge engineering tools currently available constitutes one of the bottlenecks of the Semantic Web [1]. Yet shared ontologies, ontology extension, and most ontology tools exhibit a certain degree of interoperability. In this experiment, we evaluate the capabilities of two ontology tools—Protégé and OilEd—to successfully import an ontology originally developed using the other tool. In Figure 1, we show the tools we have evaluated, indicating relationships among the tools. The arrows show the sorts of output formats (languages) that each tool can produce.

OilEd (version 3.5) http://oiled.man.ac.uk is the *de facto* standard environment for the language which grew out of the combination of DAML and OIL and has been variously known as DAML+OIL and OWL. The Web Ontology Language (OWL) has recently been advanced to a W3C Candidate Recommendation status. Details are available under the Semantic Web activity of W3C at http://www.w3.org/2001/sw . OWL is based on description logics but has many of the syntactic and other features of Frame languages. As DAML+OIL, the native format for OilEd ontologies, is not readable by Protégé 2.0 beta, ontologies created with OilEd should be exported in RDFS format to be readable by Protégé. OilEd can export in OWL format but is unable to import ontologies in this format, so OWL will not be evaluated.

Protégé 2.0 beta http://protege.stanford.edu is an extensible ontology editor and a knowledge base editor. Protégé uses the Open Knowledge-Base Connectivity protocol (OKBC) model as the basis for its own knowledge model. OKBC is a common query and construction interface for frame-based systems. As an effort to be compatible with other ontology tools, Protégé can export its ontologies in RDFS format. The current version provides beta level support for editing Semantic Web ontologies in OWL. The PAL constraints and Queries Tabs, a plug-in to represent axioms, is not compiled for Protégé 2.0 yet.

DAML+OIL plug-in for Protégé (alpha version) http://www.ai.sri.com/daml/DAML+OIL-plugin. is developed at SRI. The plug-in generates ontologies in two formats simultaneously, PPRJ and DAML, which are readable by Protégé and OilEd respectively. The OWL format is not supported.

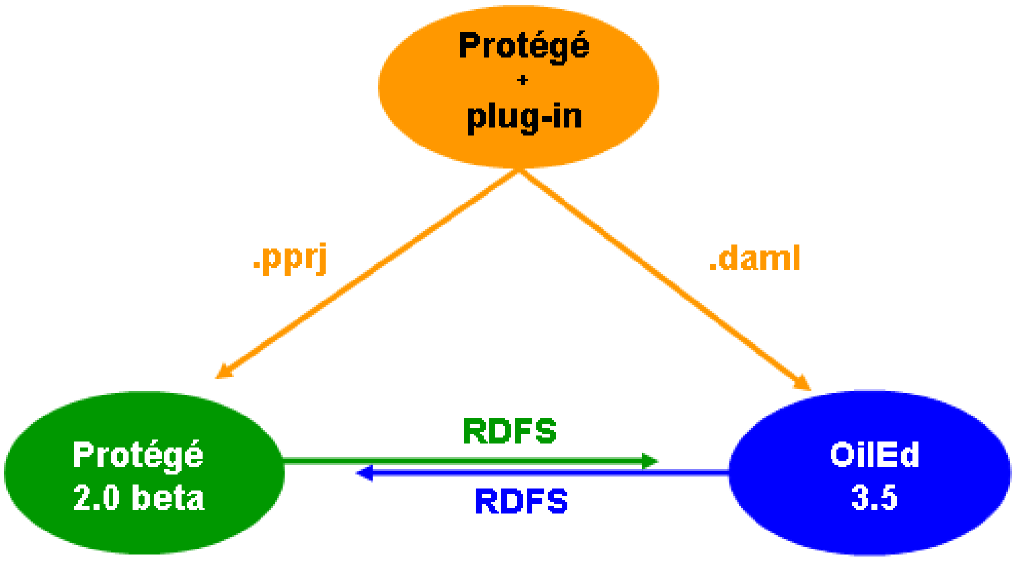


Figure 1. Relevant file formats for the two ontology tools being evaluated in this experiment.

II. Building the model

In order to test the interoperability of OilEd and Protégé, we have developed an ontology in the domain knowledge of osteoporosis, a common medical disorder. Our high level ontology has been modeled after the NLM's Unified Medical Language System (UMLS) Semantic Network http://www.nlm.nih.gov/research/umls , a freely available knowledge source which has been subject of numerous publications.

Our ontology contains over 200 concepts representing clinically-relevant aspects of osteoporosis, such as physical signs, symptoms, diagnostic tests and management options. Salient characteristics of knowledge to be represented in a biomedical ontology include:

- Preferred name. Several biomedical concepts are referenced by more than one name, and one of them is usually preferred over the others. For example "Postmenopausal osteoporosis" is also known as "Type I osteoporosis", but the former is the preferred one.
- Synonymy. There are biomedical concepts which have up to six synonymous (e.g. "Disease of hematopoietic system" has as synonymous "Blood dyscrasia", "Hematologic disease", "Disorder of hematopoietic system", "Hematopoietic disease", "Blood disorder", and "Hematopoietic disorder ").
- Disjoint concepts. Examples of mutually exclusive but not exhaustive concepts include "Medical device" and "Clinical drug", both subclasses of "Manufactured object".

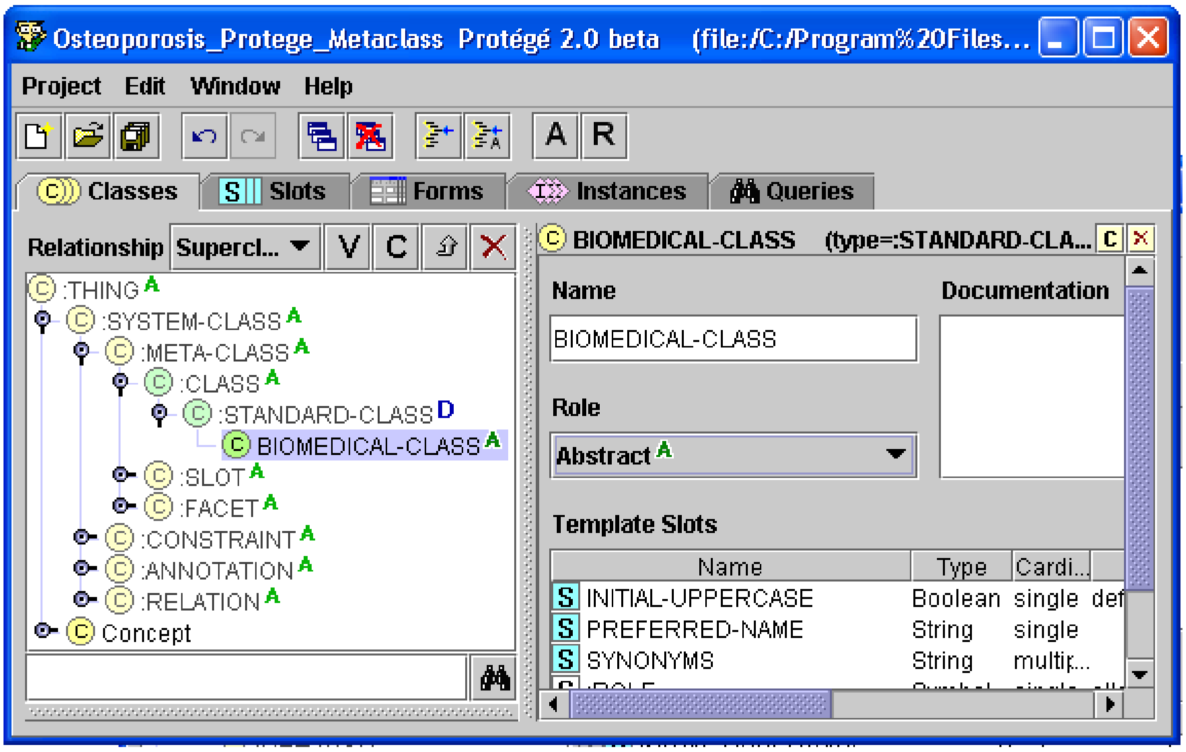
- Partition. Examples of mutually exclusive and exhaustive concepts include “Organic chemical”, “Inorganic chemical”, and “Element, ion, or isotope”, all of them subclasses of “Chemical viewed structurally”.
- Defined and primitive classes. We have a defined class, when we are able to assign *sufficient* as well as *necessary* conditions for the class (e.g. “metabolic disease” and “disease of bone” for the defined class “metabolic bone disease”). In the case of most biomedical concepts, we can only assign them some *necessary* conditions. These classes are so-called primitive classes. (e.g. the class “metabolic bone disease” and some *necessary* but not *sufficient* properties build up the primitive class “osteoporosis”).
- Multiple inheritance (polyhierarchy). Most biomedical concepts have more than one parent class.
- Abstract concepts. Some concepts, such as “Element, Ion, or Isotope” are used only for classification purposes. These abstract concepts can have subclasses, but not instances.
- Inverse relations. In some cases it is useful to represent relations that have inverse meanings because both are useful. (e.g. “causes” and “has_etiology”). The ontology should be able to automatically assign values to the other relation when one of them is used.
- Relation hierarchies. The UMLS Semantic Network associates all its 54 relations in a hierarchy. For example, the relation “spatially_related_to” and “temporally_related_to” are both subclasses of the relation “associated with”.

In Table 1, we show how each of these three tools represents each of these ontological characteristics.

	Protégé 2.0 beta	**Protégé + plug-in**	**OilEd 3.5**
Preferred name	Represented as a metaclass	Represented as the class name	Represented as the class name
Synonymy	Represented as a meta-class with multiple cardinality	Not satisfactory because custom-built metaclasses are not allowed.	Not satisfactory because custom-built metaclasses are not allowed.
Disjoint concepts	Not possible	Possible, as a "LogicalDefinition".	Possible, as axioms
Partition	Not possible	Not possible	Initially possible but a bug converts a partition into disjunctions when the ontology is saved to disk (Figure 6)
Defined and primitive classes	Not possible	Possible	Possible (Figure 5)
Polyhierarchy	Possible	Initially possible but one of the parent classes disappears when imported by OilEd (Figure 3)	Possible
Abstract concepts	Possible	Not possible	Not implemented; metaclasses are not supported
Inverse relations	Implemented in the tool but not useful in this ontology because slots are used in override mode.	Implemented in the tool but not useful in this ontology because slots are used in override mode.	Yes (Figure 4).
Relation hierarchies	Yes, hierarchies are graphically displayed	Yes, hierarchies are graphically displayed	Yes, but the hierarchy is not graphically displayed.

Table 1. Comparison of the ontology-building capabilities of Protégé, Protégé+plug-in, and OilEd.

Ontologies generated using the alpha DAML+OIL plug-in for Protégé can not represent properly multiple inheritance (see Figure 3). The ability to represent polyhierarchies is crucial for a biomedical ontology, so we choose not to further test this plug-in tool in our experiment.


Osteoporosis_Protege_Metaclass Protégé 2.0 beta (file:/C:/Program%20Files...
Project Edit Window Help
A R
Classes Slots Forms Instances Queries
Relationship Supercl... V C

```
:THING
  :SYSTEM-CLASS
    :META-CLASS
      :CLASS
        :STANDARD-CLASS
          BIOMEDICAL-CLASS
      :SLOT
      :FACET
    :CONSTRAINT
    :ANNOTATION
    :RELATION
  Concept
```

BIOMEDICAL-CLASS (type=:STANDARD-CLA...

Name: BIOMEDICAL-CLASS

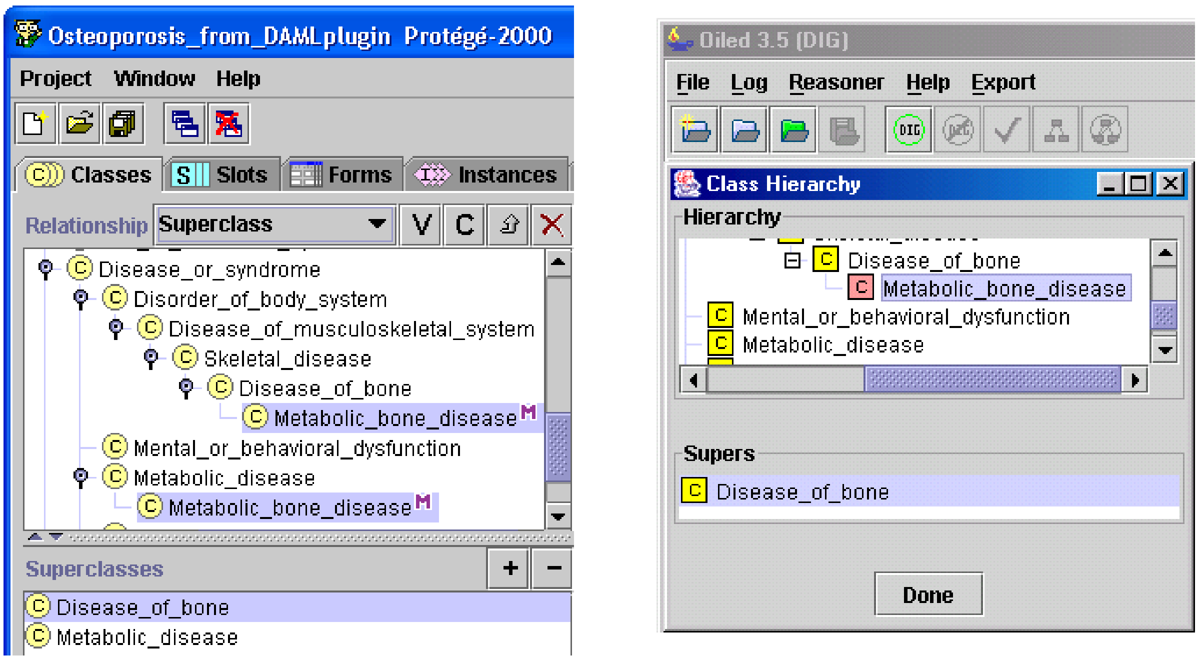
Documentation

Role: Abstract

Template Slots

Name	Type	Cardi...
INITIAL-UPPERCASE	Boolean	single def
PREFERRED-NAME	String	single
SYNONYMS	String	multip...

Figure 2. All concepts in our Protégé ontology have been modeled as subclasses of the *Biomedical-class* metaclass.


Osteoporosis_from_DAMLplugin Protégé-2000
Project Window Help
Classes Slots Forms Instances
Relationship Superclass V C
Oiled 3.5 (DIG)
File Log Reasoner Help Export
Class Hierarchy
Done

```
Disease_or_syndrome
  Disorder_of_body_system
    Disease_of_musculoskeletal_system
      Skeletal_disease
        Disease_of_bone
          Metabolic_bone_disease
  Mental_or_behavioral_dysfunction
  Metabolic_disease
    Metabolic_bone_disease
```

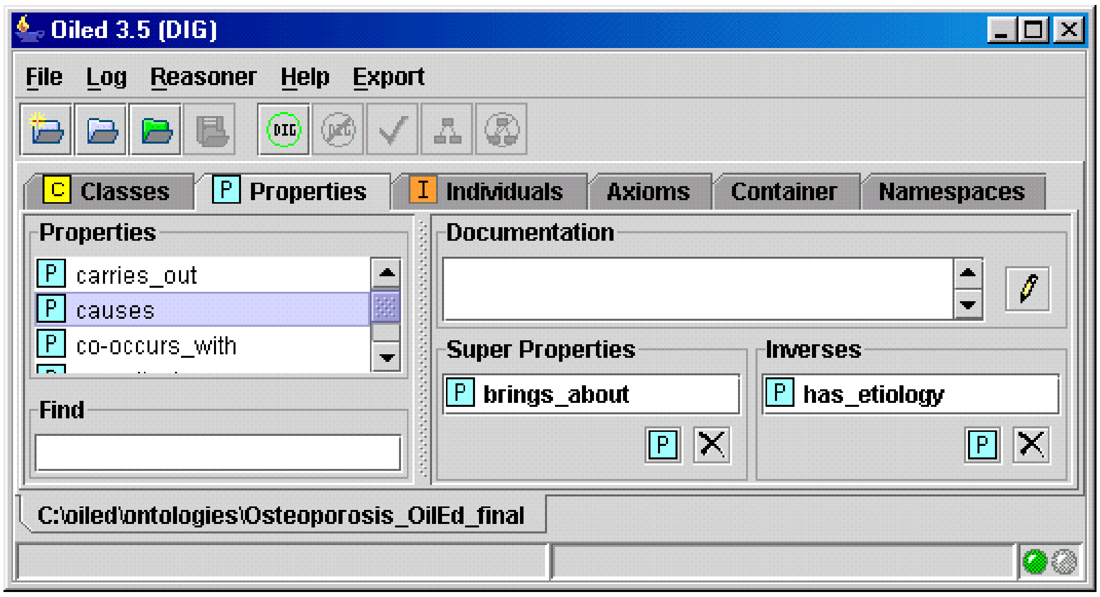
Superclasses
- Disease_of_bone
- Metabolic_disease

Hierarchy

```
Disease_of_bone
  Metabolic_bone_disease
Mental_or_behavioral_dysfunction
Metabolic_disease
```

Supers
- Disease_of_bone

Figure 3. Polyhierarchies modeled with DAML+OIL plug-in for Protégé (left pane) disappear when imported by OilEd (right pane).



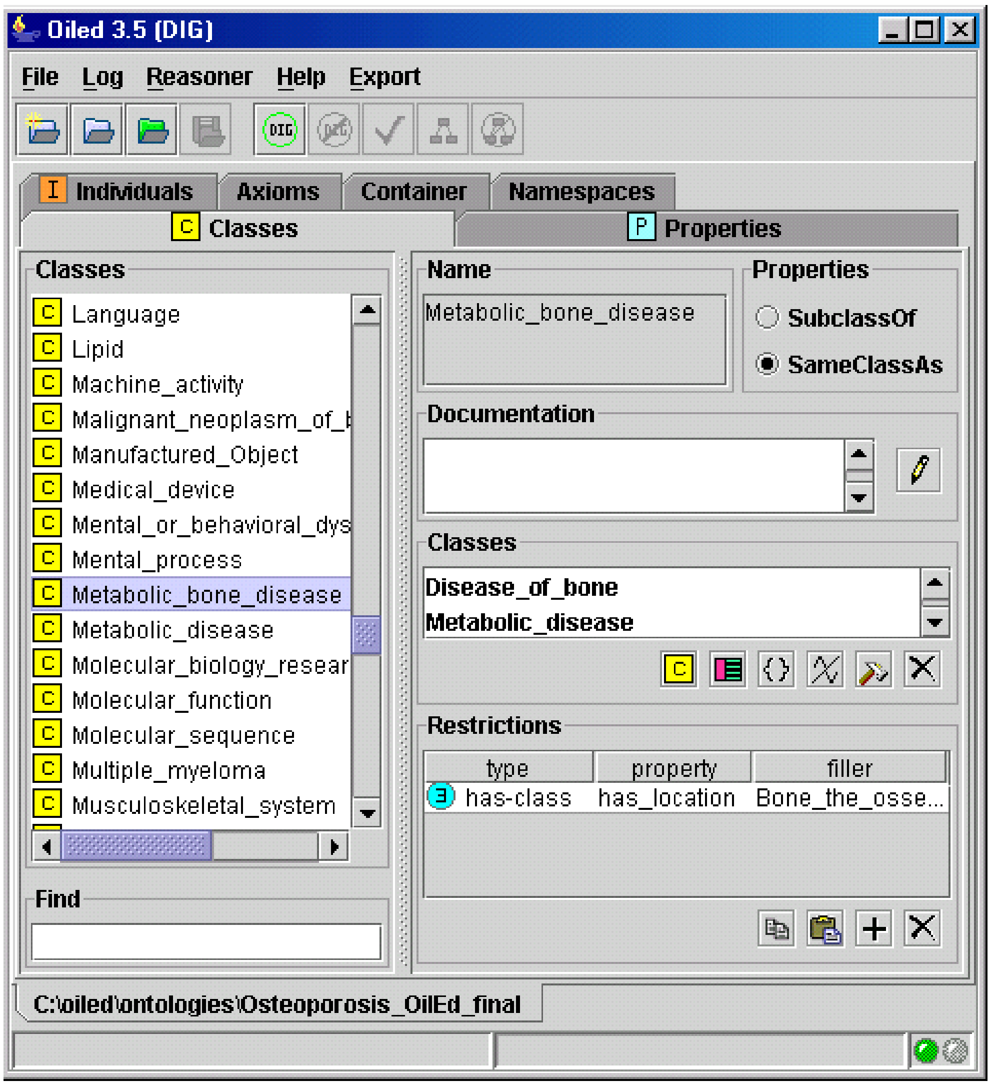
carries_out
causes
co-occurs_with

brings_about

has_etiology

C:\oiled\ontologies\Osteoporosis_OilEd_final

Figure 4. OilEd easily represents inverse properties (called slots in Protégé).



Classes
Language
Lipid
Machine_activity
Malignant_neoplasm_of_I
Manufactured_Object
Medical_device
Mental_or_behavioral_dys
Mental_process
Metabolic_bone_disease
Metabolic_disease
Molecular_biology_resear
Molecular_function
Molecular_sequence
Multiple_myeloma
Musculoskeletal_system

Find

Name: Metabolic_bone_disease

Properties: SubclassOf [✓ SameClassAs]

Documentation

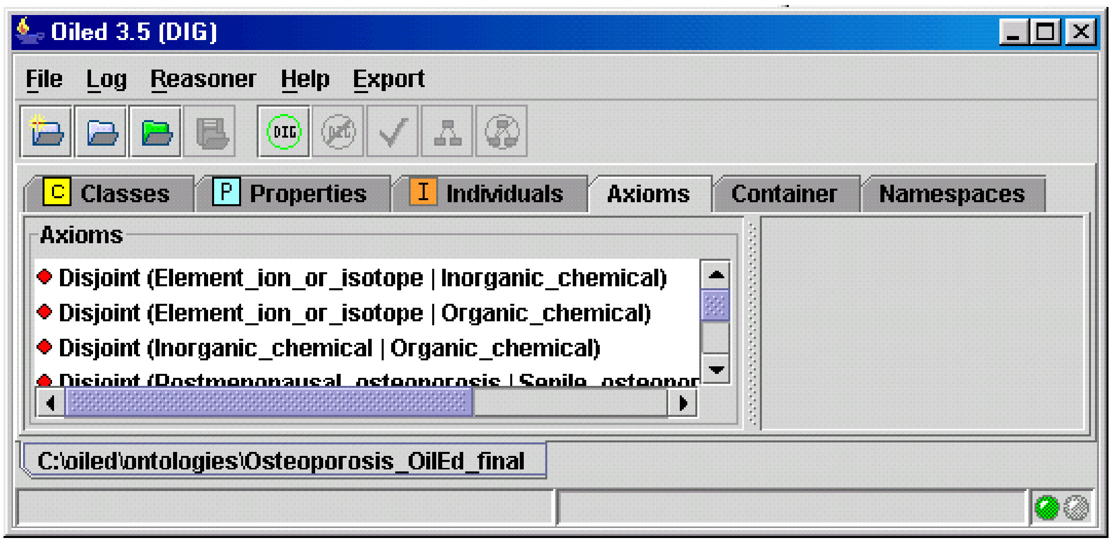
Classes
Disease_of_bone
Metabolic_disease

Restrictions

type	property	filler
∃ has-class	has_location	Bone_the_osse...

C:\oiled\ontologies\Osteoporosis_OilEd_final

Figure 5. Example of a defined class (Metabolic_bone_disease) in OilEd.


Oiled 3.5 (DIG)
File Log Reasoner Help Export
Classes Properties Individuals Axioms Container Namespaces
Axioms

Disjoint (Element_ion_or_isotope | Inorganic_chemical)
Disjoint (Element_ion_or_isotope | Organic_chemical)
Disjoint (Inorganic_chemical | Organic_chemical)
Disjoint (Postmenopausal_osteoporosis | Senile_osteopor

C:\oiled\ontologies\Osteoporosis_OilEd_final

Figure 6. Representation of disjoint classes in OilEd. A bug prevents the definitive representation of partitions.

The ontologies created with Protégé, Protégé with the DAML+OIL plug-in, and OilEd are all available from: http://www.galenonet.com/Osteoporosis_Interoperability_experiment_Oct16.zip

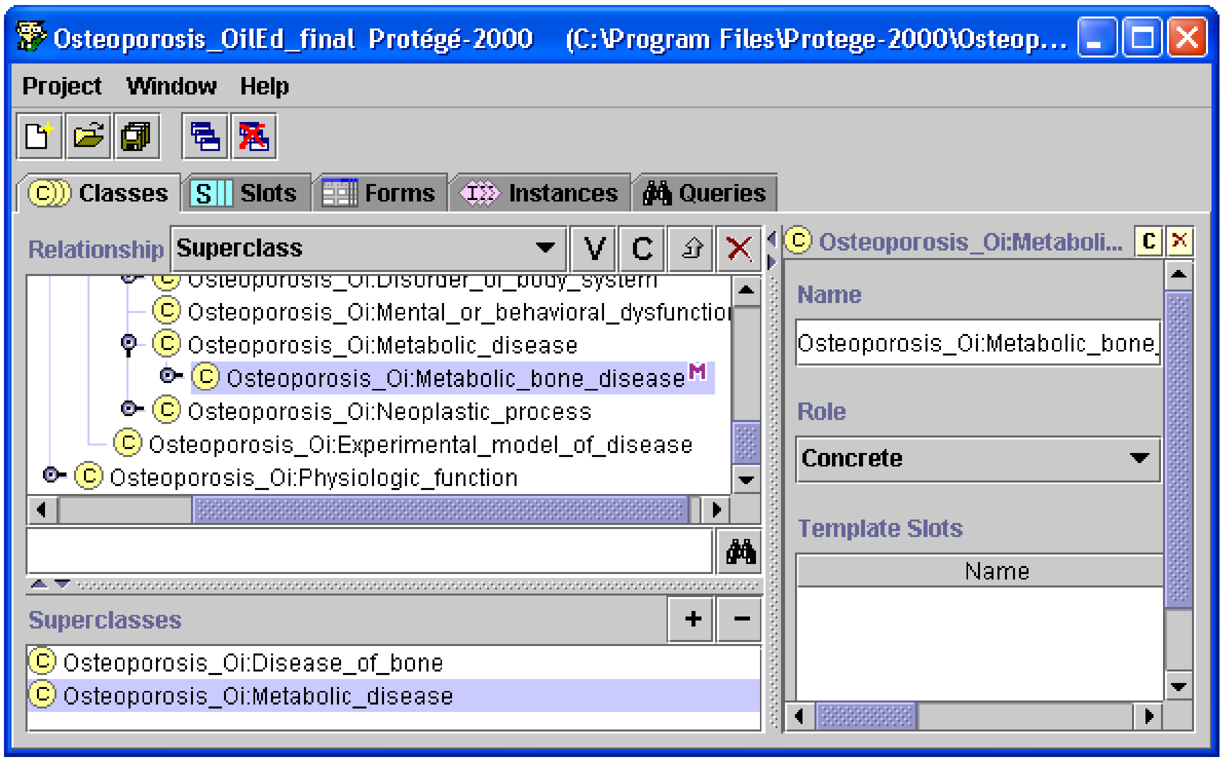
III. From Protégé to OilEd

The ontology was exported from Protégé 2.0 beta in RDFS format. When opened with OilEd 3.5, all properties (slots in Protégé) were present but none of the classes. In additions, properties had lost its hierarchy. As classes were not present, we did not perform any further interoperability tests.

IV. From OilEd to Protégé and back

When the ontology—created with OilEd—is later imported by Protégé, all the class names are displayed with a prefix and we could not find a way to get rid of them (Figure 1.). However, the main limitation we found in this step of the interoperability evaluation of the tools is the disappearance of the restrictions modeled with OilEd. For example, Protégé's Template Slots window does not contain any representation of OilEd's Restriction "Metabolic_bone_disease" "has-class" "has_location" "Bone" (compare Figure 5 and Figure 7). Table 2 summarizes these changes.

We then saved to disk the imported ontology (originally created with OilEd), using the RDFS format. When this ontology (saved by Protégé, originally created with OilEd) was opened by OilEd, the classes pane was empty. The situation was similar to the one described above in section III. OilEd is not capable of successfully importing classes from ontologies saved with Protégé.


Osteoporosis_OilEd_final Protégé-2000 (C:\Program Files\Protege-2000\Osteop...
Project Window Help
Classes
Slots
Forms
Instances
Queries
Superclasses
Template Slots

Relationship: Superclass

Osteoporosis_Oi:Disorder_of_body_system
Osteoporosis_Oi:Mental_or_behavioral_dysfunctio
Osteoporosis_Oi:Metabolic_disease
Osteoporosis_Oi:Metabolic_bone_disease
Osteoporosis_Oi:Neoplastic_process
Osteoporosis_Oi:Experimental_model_of_disease
Osteoporosis_Oi:Physiologic_function

Superclasses:
Osteoporosis_Oi:Disease_of_bone
Osteoporosis_Oi:Metabolic_disease

Osteoporosis_Oi:Metaboli...

Name: Osteoporosis_Oi:Metabolic_bone_

Role: Concrete

Template Slots: Name

Figure 7. The ontology created with OilEd loses its restrictions (Template slots) when imported by Protégé.

	Ontology as imported by Protégé
Disjoint concepts	Not applicable
Defined and primitive classes	Not applicable
Polyhierarchy	Yes
Inverse relations	Disappear
Relation hierarchies	Conserved

Table 2. Changes found when the RDFS ontology is imported by Protégé.

V. Discussion

We designed an experiment to specifically evaluate the interoperability of Protégé 2.0 beta and OilEd 3.5, two promising tools to create ontologies. Our results demonstrate that interoperability is not possible between these tools, by way of the RDFS format.

The work here does not investigate the *causes* of these interoperation problems. In some cases, the problems we report may simply be due to immature tool development. The semantic web languages in particular are quite new, and it may take some time before robust and well-tested tools are available for these languages. However, in other cases, interoperation problems may be more fundamental, indicating a gap or discrepancy in the underlying knowledge models. For example, the inability of Protégé to understand and use disjoint concepts and defined concepts (see Table 2) may fall into this category.

Each one of the two ontology engineering tools analyzed in this experiment offer special capabilities to represent biomedical knowledge that the other tool cannot offer. Protégé's advantage over OilEd include the representation of preferred names, synonymous, and abstract concepts. On the other way, OilEd uniquely allows the representation of disjoint concepts, defined and primitive classes, and to more easily represent inverse relations.

Interoperability of ontology engineering tools is highly desired, in order to integrate the different knowledge representations developed by different groups and organizations. However, knowledge representation is an area so complex that, in general, the different tools available lack interoperability. The ongoing CO-ODE project, which will merge the best of Protégé and OilEd, promises to enable interoperability between knowledge representation tools by using OWL.

Acknowledgement

To Richard Phillips, MD, for his useful comments and review of the manuscript.

This work was supported by NIH grant D43 TW001286-04S1 Fogarty International Training in Medical Informatics.

Bibliography

Fensel D. Ontologies: Their Glory and the New Bottlenecks They Create. Workshop Semantic Web Action Line 2001. http://www.ontoweb.org/workshop/amsterdamdec8/ShortPresentation/swt_dieter.pdf

Noy N. F., Fergerson R. W., Musen M. A. The knowledge model of Protege-2000: Combining interoperability and flexibility. 2nd International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France. 2000. http://protege.stanford.edu/publications/Knowledge_Model/protege-knowledge-model.html

Rector A. and colleagues. OilEd Normalised Ontology Tutorial – Biomedical version (for OilEd version 3.4). October 2002. http://www.cs.man.ac.uk/mig/ontology-tutorial/oiled-biomedical-ontology-tutorial.zip

Rector A. CO-ODE/HyOntUse. Sixth International Protégé Workshop. July 2003. Manchester, England. http://protege.stanford.edu/workshop_vi/Alan_Rector_Co-Ode-Southampton-at-Manchester-2003-07-061.pdf