13th International Conference on
Knowledge Engineering and Knowledge Management
In Cooperation with AAAI

October 1-4, 2002 Sigüenza, Spain

EON2002
Evaluation of Ontology-based Tools

EKAW02 Workshop (WS1)

Sigüenza, Spain, September 30th 2002

Organizing Committee

Jürgen Angele
York Sure

Sponsorship

This workshop is sponsored by the thematic network OntoWeb (EU IST-2000-29243).

Introduction

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

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

The workshop is divided in two parts: (i) presentations of accepted papers and (ii) discussions about the "EON2002 Experiment". The experiment was initiated during the OntWeb3 meeting by the participants of the Special Interest Group on Tools (SIG3)[1]. The general question was how to evaluate ontology related technologies. To brake down this rather complex task into a pragmatic one, the group decided to focus on ontology engineering environments (OEE) as a starting point. These tools are rather common and widely used by the Semantic Web Community and some of the participating members were even tool provider themselves. Submissions to this experiment should answer the following items with respect to the used OEE:

- What modeling decisions need to be considered during the design?
- What limitations occure? ... and why?
- What problems arise due to using different representation languages for export?
- What are the lessons learned from modelling this experiment?

The ontology should be exported into a common representation language. However, most OEEs were designed having specific design rationals from representation formalisms in mind. Therefore they typically have a strong support for their "home language". To make the results more comparable we encouraged people to provide not only an "home language" export, but also an RDF(S) export.

The results of the experiment (as well as further information about the workshop) can be found at: `http://km.aifb.uni-karlsruhe.de/eon2002/`

We thank all members of the program committee, additional reviewers, authors, experiment participants and local organizers for their efforts. This workshop is supported by OntoWeb (EU IST-2000-29243).

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

Jürgen Angele & York Sure

[1] Further information can be found in the OntoWeb deliverable 1.3 that is downloadable at http://www.ontoweb.org/.

Table of Contents

Evaluating Ontology-Mapping Tools: Requirements and Experience

Natalya F. Noy and Mark A. Musen

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

Abstract. The appearance of a large number of ontology tools may leave a user looking for an appropriate tool overwhelmed and uncertain on which tool to choose. Thus evaluation and comparison of these tools is important to help users determine which tool is best suited for their tasks. However, there is no "one size fits all" comparison framework for ontology tools: different classes of tools require very different comparison frameworks. For example, ontology-development tools can easily be compared to one another since they all serve the same task: define concepts, instances, and relations in a domain. Tools for ontology merging, mapping, and alignment however are so different from one another that direct comparison may not be possible. They differ in the type of input they require (e.g., instance data or no instance data), the type of output they produce (e.g., one merged ontology, pairs of related terms, articulation rules), modes of interaction and so on. This diversity makes comparing the performance of mapping tools to one another largely meaningless. We present criteria that partition the set of such tools in smaller groups allowing users to choose the set of tools that best fits their tasks. We discuss what resources we as a community need to develop in order to make performance comparisons within each group of merging and mapping tools useful and effective. These resources will most likely come as results of evaluation experiments of stand-alone tools. As an example of such an experiment, we discuss our experiences and results in evaluating PROMPT, an interactive ontology-merging tool. Our experiment produced some of the resources that we can use in more general evaluation. However, it has also shown that comparing the performance of different tools can be difficult since human experts do not agree on how ontologies should be merged, and we do not yet have a good enough metric for comparing ontologies.

1 Ontology-Mapping Tools Versus Ontology-Development Tools

Consider two types of ontology tools: (1) tools for developing ontologies and (2) tools for mapping, aligning, or merging ontologies. By **ontology-development tools** (which we will call *development tools* in the paper) we mean ontology editors that allow users to define new concepts, relations, and instances. These tools usually have capabilities for importing and extending existing ontologies. Development tools may include graphical browsers, search capabilities, and constraint checking. Protégé-2000 [17], OntoEdit [19], OilEd [2], WebODE [1], and Ontolingua [7] are some examples of development tools. **Tools for mapping, aligning, and merging ontologies** (which we will call *mapping tools*) are the tools that help users find similarities and differences between source ontologies. Mapping tools either identify potential correspondences automatically or provide the environment for the users to find and define these correspondences, or both. Mapping tools are often extensions of development tools. Mapping tool and algorithm examples include PROMPT[16], ONION [13], Chimaera [11], FCA-Merge [18], GLUE [5], and OBSERVER [12].

Even though theories on how to evaluate either type of tools are not well articulated at this point, there are already several frameworks for evaluating ontology-development tools. For example, Duineveld and colleagues [6] in their comparison experiment used different development tools to represent the same domain ontology. Members of the Ontology-environments SIG in the OntoWeb initiative[1] designed an extensive set of criteria for evaluating ontology-development tools and applied these criteria to compare a number of projects. Some of the aspects that these frameworks compare include:

– interoperability with other tools and the ability to import and export ontologies in different representation languages;
– expressiveness of the knowledge model;
– scalability and extensibility;
– availability and capabilities of inference services;
– usability of the tools.

Let us turn to the second class of ontology tools: tools for mapping, aligning, or merging ontologies. It is tempting to reuse many of the criteria from evaluation of development tools. For example, expressiveness of the underlying language is important and so is scalability and extensibility. We need to know if a mapping tool can work with ontologies from different languages. However, if we look at the mapping tools more closely, we see that their comparison and evaluation must be very different from the comparison and evaluation of development tools. All the ontology-development tools have very similar *inputs and the desired outputs*: we have a domain, possibly a set of ontologies to reuse, and a set of requirements for the ontology, and we need to use a tool to produce an ontology of that domain satisfying the requirements. Unlike the ontology-development tools, the

[1] http://delicias.dia.fi.upm.es/ontoweb/sig-tools/

ontology-mapping tools *vary with respect to the precise task that they perform, the inputs on which they operate and the outputs that they produce.*

First, the *tasks* for which the mapping tools are designed, differ greatly. On the one hand, all the tools are designed to find similarities and differences between source ontologies in one way or another. In fact, researchers have suggested a uniform framework for describing and analyzing this information regardless of what the final task is [3, 10]. On the other hand, from the user's point of view the tools differ greatly in what tasks this analysis of similarities and differences supports. For example, Chimaera and PROMPT allow users to merge source ontologies into a new ontology that includes concepts from both sources. The output of ONION is a set of articulation rules between two ontologies; these rules define what the similarities and differences are. The articulation rules can later be used for querying and other tasks. The task of GLUE, AnchorPROMPT [14] and FCA-Merge is to provide a set of pairs of related concepts with some certainty factor associated with each pair.

Second, different mapping tools rely on *different inputs*: Some tools deal only with class hierarchies of the sources and are agnostic in their merging algorithms about slots or instances (e.g., Chimaera). Other tools use not only classes but also slots and value restrictions in their analysis (e.g., PROMPT). Other tools rely in their algorithms on the existence of instances in each of the source ontologies (e.g., GLUE). Yet another set of tools require not only that instances are present, but also that source ontologies share a set of instances (e.g., FCA-Merge). Some tools work independently and produce suggestions to the user at the end, allowing the user to analyze the suggestions (e.g., GLUE, FCA-Merge). Some tools expect that the source ontologies follow a specific knowledge-representation paradigm (e.g., Description Logic for OBSERVER). Other tools rely heavily on interaction with the user and base their analysis not only on the source ontologies themselves but also on the merging or alignment steps that the user performs (e.g., PROMPT, Chimaera).

Third, since the tasks that the mapping tools support differ greatly, the *interaction between a user and a tool* is very different from one tool to another. Some tools provide a graphical interface which allows users to compare the source ontologies visually, and accept or reject the results of the tool analysis (e.g., PROMPT, Chimaera, ONION), the goal of other tools is to run the algorithms which find correlations between the source ontologies and output the results to the user in a text file or on the terminal–the users must then use the results outside the tool itself.

The goal of this paper is to start a discussion on a framework for evaluating ontology-mapping tools that would account for this great variety in underlying assumptions and requirements. We argue that many of the tools cannot be compared directly with one another because they are so different in the tasks that they support. We identify the criteria for determining the groups of tools that *can* be compared directly, define what resources we need to develop to make such comparison possible and discuss our experiences in evaluating our merging tool, PROMPT, as well as the results of this evaluation.

2 Requirements for Evaluating Mapping Tools

Before we discuss the evaluation requirements for mapping tools, we must answer the following question which will certainly affect the requirements: what is the goal of such potential evaluation? It is tempting to say "find the best tool." However, as we have just discussed, given the diversity in the tasks that the tools support, their modes of interaction, the input data they rely on, it is impossible to compare the tools to one another and to find one or even several measures to identify the "best" tool.

Therefore, we suggest that the questions driving such evaluation must be user-oriented. A user may ask either *what is the best tool for his task* or *whether a particular tool is good enough for his task*. Depending on what the user's source ontologies are, how much manual work he is willing to put in, how important the precision of the results is, one or another tool will be more appropriate. Therefore, the first set of evaluation criteria are **pragmatic criteria**. These criteria include but are not limited to the following:[2]

Input requirements What elements from the source ontologies does the tool use? Which of these elements does the tool require? This information may include: concept names, class hierarchy, slot definitions, facet values, slot values, instances. Does the tool require that source ontologies use a particular knowledge-representation paradigm?

Level of user interaction Does the tool perform the comparison in a "batch mode," presenting the results at the end, or is it an interactive tool where intermediate results are analyzed by the user, and the tool uses the feedback for further analysis?

Type of output What is the result of working with the tool? Is it a set of articulation rules? Is it a merged ontology? Is it an (instantiated) ontology representing the mappings? Is it a list of pairs of related concepts (possibly with a certainty factor associated with them)?

Content of output Which elements of the source ontologies are correlated in the output? These elements can include relations between classes, slots, values, or instances.

There is no single "best" set of answers to these questions. If the user's ontologies include instance data, the tools that use this data in their analysis will provide more precise suggestions. However, if the instance data is not available, these tools are inappropriate. Similarly, if the user needs only approximate mappings between the sources, the tools that provide less precise mappings but require less interaction may be what the user is looking for. In other words, if we create a comparison matrix of tools and their features (as most current comparisons of the tools do), the best tool is not the tool that gets the largest number of checkmarks in the matrix. Rather, the best tool is the tool whose set of checkmarks best matches the user's conditions. From a practitioner's point of view,

[2] This list is a summary of many of the parameters used in the OntoWeb initiative for comparing mapping tools.

existing comparison frameworks have perhaps over-emphasized the importance of getting as many features in a single tool as possible (e.g., [9]).

These pragmatic criteria will help a user identify a group of tools that will be useful to him. Within a group of tools that use the same type of input data and produce similar types of outputs with similar level of interaction, which one should the user choose? At this point the quality of comparison algorithms comes into play. All other things being equal, it is the tool that produces "better" suggestions that will be most beneficial to us. Therefore, we need to define what "better" is in this context and how to find which tool is indeed better according to this **performance criterion**. Defining what "better" is, is fairly easy: the tool with better recall and precision wins. We can define **recall** as the fraction of correct matches that the algorithm identifies. We can define **precision** as the fraction of correct matches among the matches that the tool identifies. These notions of recall and precision are similar to recall and precision used in information retrieval: recall measures how much of the useful information the tools finds; precision measures how much of the information that the tool finds is useful. Therefore, we can perform experiments comparing the tools in the same group directly to one another to determine the quality of the comparison algorithms. Ontology-mapping tools employ a variety of techniques to compare source ontologies. Some tools use machine learning (e.g., GLUE and FCA-Merge), others analyze graph structure (e.g., AnchorPROMPT, ONION), yet other tools use heuristic-based analyzers (e.g., ONION, PROMPT).

To compare the tools within one group, we as a community need to develop the following resources:

Source ontologies We need (preferably several sets of) pairs of ontologies covering similar domains. We would like to have sets of ontologies of different sizes, with different levels of overlap, some of them complicated and some of them close to simple hierarchies.

Benchmark results For each pair of the source ontologies, we need human-generated correspondences between them. Again, we would like to have these correspondences at different levels and in different forms: pairs of related concepts, logic rules showing more complex transformations, ontologies representing the mappings.

Metrics for comparing the performance of the tools For the tools that produce a list of correspondences between concepts in the source ontologies, we can use the measures of recall and precision that we defined earlier. For the tools that result in new merged ontologies, we must compare the resulting ontologies with the benchmark ones. Therefore, we need some precise measure of the "distance" between ontologies. There are several proposals on measuring distance between individual concepts. However, we need distance between ontologies as a whole. As an alternative to measuring distance between ontologies, we can consider evaluating the ontologies resulting from experiments based on analysis of taxonomic relationships proposed by Guarino and Welty [8]. We can use the information on essence, rigidity, and other properties defined in the benchmark results to determine whether these properties hold in the ontologies resulting from the experiments.

Last but not least, when we perform the comparison experiments, we must be careful to maintain a set of **experiment controls**: level of the users' expertise with a particular tool and with the mapping process in general, the amount of training the users get, the documentation that is available, and so on. The more uniform these controls are, the more valid the experiments will be.

Ideally, researchers that do not have a vested interest in any of the tools should create the resources that we have listed. Otherwise, the selection of resources and the benchmarks will inevitably be skewed towards one or the other tool. However, in practice, this approach may not be possible, unless there is specific funding to create the resources. Therefore, realistically, these resources are most likely to come initially as results of stand-alone evaluation experiments of specific tools. These experiments evaluate whether a particular tool is "good enough" for the user's task. To perform these individual experiments, researchers will need to find source ontologies covering the same domain. They will need to create manually a gold-standard ontology to serve as a benchmark. Alternatively, the merged ontologies that the experiment participants will produce could also serve as benchmarks. In the experiment, the researchers will likely need to compare the resulting ontologies, thus developing some metric of the distance. Therefore, many of the resources for future comparative evaluation of different tools can come as results of such stand-alone experiment. We performed such an experiment evaluating PROMPT—an ontology-merging tool developed in our laboratory [16]. We describe the experiment in the rest of this paper.

3 PROMPT Evaluation

PROMPT [16] is a tool for interactive ontology merging. It is a plugin for Protégé-2000.[3] PROMPT leads the user through the ontology-merging process, identifying possible points of integration, and making suggestions for operations that should be done next, what conflicts need to be resolved, and how to resolve them. The tool compares names of concepts, relations among them, constraints on slot values, and instances of concepts to make its suggestions.

We evaluated the quality of the suggestions that the tool provides by asking several users to merge two source ontologies using PROMPT. We recorded their steps, which suggestions they followed, which suggestions they did not follow, and what the resulting ontology looked like.

3.1 Source ontologies

In order to evaluate the performance of the PROMPT merging tool, we chose two ontologies that were developed independently by two teams in the DAML project.[4] We imported two ontologies from the DAML ontology library [4]:

[3] http://protege.stanford.edu
[4] http://www.daml.org

1. An ontology for describing individuals, computer-science academic departments, universities, and activities that occur at them developed at the University of Maryland (UMD), and
2. An ontology for describing employees in an academic institutions, publications, and relationships among research groups and projects developed at Carnegie Mellon University (CMU).

These two ontologies constituted a good target for the merging experiment because on the one hand, they covered similar subject domains (research organizations and projects, publications, etc.) and on the other hand, their developers worked completely independent of one another and therefore there was no intensional correlation among terms in the ontologies. In addition, the domain is easy to understand for everyone.

Figure 1 presents snapshots of the two hierarchies. Note that many of the concepts in the two ontologies are similar, but they are represented by different terms: *Industrial_org* versus *CommercialOrganization*, *Governmental_org* versus *GovernmentOrganization*, *Student* versus *Students*, *Organisation* versus *Organization*. The structure of the hierarchy is also different: In the CMU hierarchy, for example, *Students*, *Faculty*, *Management* are subclasses of the class *Employment_Categories*, whereas in the UMD hierarchy these types of classes are subclasses of *Person*. Even though the CMU hierarchy has a class *Person*, its only subclass is *Employee*.

Given these differences in the sources, the merged ontologies produced by different users will inevitably be different: There are many design decisions that could go either way. For example, will the *Organization* class in the merged ontology be at the top level, as it is in the CMU hierarchy, or will it be a subclass of *SocialGroup*, as it is in the UMD hierarchy? Are the classes *Employment_Categories* from the CMU ontology and *Employee* from the UMD ontology essentially the same classes? The easiest way to answer these questions is to have the designers of the two original ontologies get together and merge them. However, in practice this scenario is unrealistic. Therefore, if our task requires that we merge the two ontologies producing one uniform ontology, the user performing the merge will have to make these decisions and different users may make different decisions.

3.2 Experiment setup

We asked users to use the PROMPT tool to merge the two ontologies described in the previous section. All the users were previously familiar with Protégé, but have not tried to use PROMPT before. None of the users has addressed the problem of merging ontologies prior to the experiment. There were four participants in the experiment—all of them students at the Stanford Medical Informatics who answered our call for participation. Each participant received a package containing:

– the PROMPT software

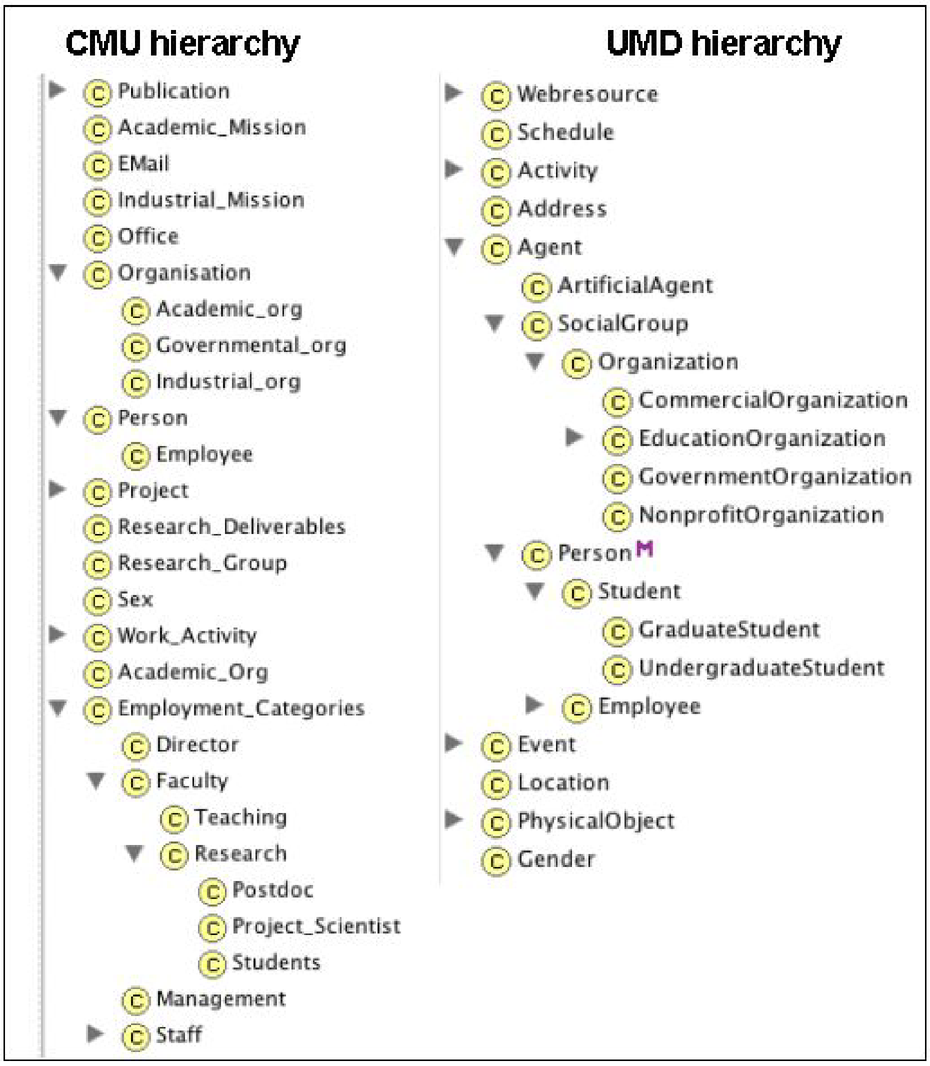


Fig. 1. Snapshots of the class hierarchies in the two source ontologies for the experiment.

– documentation of the tool
– a detailed tutorial
– a tutorial example
– materials for the evaluation

The users performed the evaluation at their convenience on their own computers. We asked each participant to install the tool, to read the tutorial and to follow the examples in the tutorial using the tutorial ontologies. We suggested that they may then look through the documentation to get additional insights. After that the participants were to perform the evaluation itself by merging the two source ontologies. After they were done, they sent back the resulting ontology and the log files for our analysis.

In order to minimize differences in the results, we arbitrarily set up the CMU ontology to be the preferred one. That is, if the users were merging two classes with different names, the name from the CMU ontology was used.

3.3 Using PROMPT: Experiment Results

The primary goal of our experiment was to evaluate the *quality of PROMPT's suggestions*. In addition, to experiment with metrics for finding a distance between ontologies, we compared the ontologies that the participants have produced.

Quality of PROMPT's suggestions To evaluate the quality of PROMPT's suggestions, we evaluated the precision and recall measures that we have described in Section 2: Precision is the fraction of the tool's suggestions that the users decided to follow. Recall is the fraction of the operations that the users performed that were suggested by the tool. In our experiments, the average precision was 96.9% and the average recall was 88.6%. The precision was in fact a lot higher than we expected. There are several possible explanations for this remarkable result. First, the users were not experts in ontology merging and did not have any particular task in mind when merging the ontologies. As a result, they found it easier simply to follow the tools suggestions, as long as they seemed reasonable rather than explore the ontologies deeper and come up with their own operations. Second, there was a significant overlap in the content and structure of the source ontologies, which made automatic generation of correct suggestions easier.

Some of the lower recall figures (it was 49% for *merge* operations in one of the experiments) result from questionable choices that the users made. For example, one user merged *Publication* and *DocumentRepresentation* classes, *EMail* and *ElectronicDocument*, *ProjectScientist* and *Research_Assistant*; slots *publisher* and *publishDate*. PROMPT also did not identify such pairs of related classes as *Academic_org* and *EducationOrganization* or *Industrial_org* and *CommercialOrganization*.

Even though we did not formally evaluate usability, the fact that all the users were able to use the tool and completely merge the source ontologies after a brief handout tutorial, indicates that the tool is fairly easy to use. Even though we told the participants that while they were still going through the tutorial, they could ask questions about the tool and about the process, only one of them ended up asking a question.

Resulting ontologies In addition to comparing the recall and precision of PROMPT's suggestions, we looked at the resulting ontologies. Since we did not have a benchmark ontology (we pulled both ontologies from the Internet from an ontology library), we could compare the ontologies resulting from the experiment only to one another.

We have noted in Section 2 that we need a distance measure between ontologies. We can treat the ontologies resulting from the experiment as versions of the same ontology—after all, they all result from merging the same ontologies. Therefore, we can use the notion of diff between versions to find a distance between two ontologies. In our earlier work [15], we defined the notion of a **structural diff** between two versions of the same ontology.

Definition 1 (Structural diff). *Given two versions of an ontology O, V_1 and V_2, a* **structural diff** *between V_1 and V_2, $D(V_1, V_2)$, is a set of frame pairs $\langle F_1, F_2 \rangle$ where:*

- *$F_1 \in V_1$ or $F_1 = null$; $F_2 \in V_2$ or $F_2 = null$*
- *F_2 is an* **image** *of F_1 (***matches** *F_1), that is, F_1 became F_2. If F_1 or F_2 is null, then we say that F_2 or F_1 respectively does not have a match.*
- *Each frame from V_1 and V_2 appears in at least one pair.*
- *For any frame F_1, if there is at least one pair containing F_1, where $F_2 \neq null$, then there is no pair containing F_1 where $F_2 = null$ (if we found at least one match for F_1, we do not have a pair that says that F_1 is unmatched). The same is true for F_2.*

Note that the definition implies that for any pair of frames F_1 and F_2, there is at most one entry $\langle F_1, F_2 \rangle$.

The structural diff describes which frames have changed from one version to another. However, for a diff to be more useful to the user, it should include not only *what* has changed but also some information on *how* the frames have changed. A PROMPTDIFF table provides this more detailed information [15].

Definition 2 (PROMPTDIFF table). *Given two versions of an ontology O, V_1 and V_2, the* PROMPTDIFF **table** *is a set of tuples $\langle F_1, F_2, rename\_value, operation\_value, mapping\_level \rangle$ where:*

- *There is a tuple $\langle F_1, F_2, rename\_value, operation\_value, mapping\_level \rangle$ in the table iff there is a pair $\langle F_1, F_2 \rangle$ in the structural diff $D(V_1, V_2)$.*
- *rename_value is true if frame names for F_1 and F_2 are the same; rename_value is $false$ otherwise.*
- *$operation\_value \in OpS$, where $OpS = \{add, delete, split, merge, map\}$*
- *$mapping\_level \in MapS$, where $MapS = \{unchanged, isomorphic, changed\}$.*

	u1-u2	u1-u3	u1-u4	u2-u3	u2-u4	u3-u4
Frames in ontology 1	251	251	251	253	253	216
Frames in ontology 2	253	216	232	216	232	232
Unmatched entries from ontology 1:	3	37	19	39	22	11
Unmatched entries from ontology 2:	5	2	0	2	1	27
Changed rows in the table:	30	50	46	48	54	45
Difference (in number of frames)	38	89	65	89	77	83
Difference (in %)	14.8%	35.2%	25.9%	34.9%	30.3%	34.2%

Table 1. The difference between pairs of ontologies in the experiment. There were four users, u1, u2, u3, and u4. Each column represents a comparison of ontologies that each pair of users produced.

The operations in the operation set *OpS* indicate to the user how a frame has changed from one version to the other: whether it was added or deleted, whether it was split in two frames, or whether two frames were merged. We assign a *map* operation to a pair of frames if none of the other operations applies. The *mapping_level* indicates how different the two frames are from each other. If the *mapping_level* is *unchanged*, then the user can safely ignore the frames—nothing has changed in their definitions. If two frames are *isomorphic*, then their corresponding slots and facet values are images of each other, but not necessarily identical images. The *mapping_level* is *changed* if the frames have slots or facet values that are not images of each other.

Therefore, we can measure the distance between two ontologies by considering the number of frames in each ontology and the number of rows in the PROMPTDIFF table that have *add*, *delete*, or *changed* in their operation or mapping-level column. In other words, the "difference" between two ontologies is comprised by the frames that either do not have matches or have changed significantly: classes have different superclasses, metaclasses, or slots; slots are attached to different classes or have different facet values. Table 1 presents the results of this comparison. For the four users in the experiment, there are six pairs of ontologies. It is interesting that even with the tool that may have been "steering" the users in a certain direction, the resulting ontologies differed by about 30%. This result indicates that even when human experts are merging ontologies, there is very little agreement and users make very different design decisions. This observation has serious implication for the possibility of even having a benchmark ontology (something that we said is needed to compare merging tools fairly). Consider the *Employment_Categories* and *Employee* classes from Figure 1. Some users decided to merge the class *Employment_Categories* with the *Employee* classes in both ontologies, creating one class out of three. Others kept the distinction that was present in the CMU hierarchy. In one ontology, *Proceedings* is a subclass of *Book* and in the other it is a subclass of *Publication*. Even though most users merged the two *Publication* classes, the two *Proceedings* classes, and the two *Book* classes, they made different decisions on where to place the *Proceedings* class in the merged ontology.

Our approach to measuring the distance may also be inflating the actual distance. For example, if two users both merged the classes *GraduateStudent* and *UndergraduateStudent* but did not merge their superclasses, then both *GraduateStudent* and *UndergraduateStudent* will appear as "changed" when we compare the merged ontologies: their superclasses are different. Therefore, a better measure may be some sort of weighted measure reflecting how much the concepts have changed.

4 Concluding Remarks

Our evaluation experiment was not ideal. We were limited by available resources and, in some cases, by circumstances. We had four participants in the experiment. The number of users was still too small and the variability in user's expertise with Protégé too large to get meaningful estimates on whether the tool really saves time. If we had more users performing the experiment, the time data would have been one of the interesting points to compare.

Such an experiment however would still have answered only one of the possible user's questions that we discussed in Section 2: is the PROMPT tool good enough. And even that answer assumes that recall and precision figures taken in isolation from other tools, are meaningful. What we really need is a larger-scale experiment that compares tools with similar sets of pragmatic criteria. For example, we would then compare PROMPT with other tools that use classes, slots, facets, and instances in their analysis and that produce suggestions about merging all these knowledge-base elements. In general, it is pointless to compare performance of PROMPT and FCA-Merge for example: FCA-Merge requires that source ontologies not only have instance data but also share the instances. PROMPT does not have such a requirement. Therefore, if a user's ontology does not have instance data, FCA-Merge will be unusable.

In order to help users sort through existing tools and find the right ones for his task, we as a community need to develop the resources that would allow us to perform meaningful experiments comparing different tools:

- Create a library of ontologies covering similar domains. Many ontologies in the DAML ontology library can serve this purpose.
- Manually define mapping between concepts in these ontologies to create consensus benchmark ontologies. Ideally, get the authors of the original ontologies involved in the process of creating mappings.
- Define formal metrics for comparing the distance between ontologies, allowing experimenters to compare the ontologies produced by participants to one another and to the benchmark ontology.
- Define experimental protocols with fixed controls that will make the results of different evaluations comparable.

Our evaluation of PROMPT produced one pair of ontologies that could be used in the library of ontologies for other experiments. Also, since PROMPT is an interactive tool and human experts validate the merged ontologies, we have a set of benchmark ontologies that result from merging the two source ontologies. However, these ontologies differ significantly, which means that there may not be a single "correct" merged ontology. Our evaluation has also produced an initial metric for comparing ontologies resulting from different experiments. All these resources will be useful in a more general evaluation comparing the performance of different ontology-mapping tools.

In addition to developing these resources, we need to answer many research questions. These questions include but are not limited to the following questions:

– Which pragmatic criteria are most helpful to users in finding the best tool for their task?
– For the ontologies in the repository, how to we develop a benchmark ontology? Does this "gold standard" mapping even exist for many of the ontologies?
– How do we measure how close to the gold standard is the analysis that the tools produce?
– Can we use some of the metric and analysis approaches that are being developed for evaluating ontologies themselves in our evaluation of ontologies resulting from the tool's analyses?

5 Acknowledgments

We would like to thank Monica Crubézy for her thoughtful comments on the paper. We are very grateful to the students at Stanford Medical Informatics who participated in the experiment. The National Cancer Institute provided the funding for our work on ontology merging and management.

References

1. J.C. Arpírez, O. Corcho, M. Fernández-López, and A. Gómez-Pérez. WebODE: a scalable workbench for ontological engineering. In *KCAP-01*, Victoria, Canada, 2001.
2. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OILEd: a reason-able ontology editor for the semantic web. In *KI2001, Joint German/Austrian conference on Artificial Intelligence*, volume LNAI Vol. 2174, pages 396–408, Vienna, 2001. Springer-Verlag LNAI Vol. 2174.
3. P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. Model management: Managing complex information structures. *SIGMOD Record*, 29(4):55–63, 2000.
4. DAML. DAML ontology library, 2001.
5. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *The Eleventh International WWW Conference*, Hawaii, US, 2002.
6. A. J. Duineveld, R. Stoter, M. R. Weiden, B. Kenepa, and V. R. Benjamins. Wondertools? a comparative study of ontological engineering tools. *International Journal of Human-Computer Studies*, 52(6):1111–1133, 2000.
7. A. Farquhar, R. Fikes, and J. Rice. The Ontolingua server: a tool for collaborative ontology construction. In *Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1996.

8. N. Guarino and C. Welty. Ontological analysis of taxonomic relationships. In A. Laender and V. Storey, editors, *ER-2000: The 19th International Conference on Conceptual Modeling*. Springer-Verlag, 2000.
9. M. Klein. Combining and relating ontologies: an analysis of problems and solutions. In *IJCAI-2001 Workshop on Ontologies and Information Sharing*, pages 53–62, Seattle, WA, 2001.
10. J. Madhavan, P. A. Bernstein, P. Domingos, and A. Halevy. Representing and reasoning about mappings between domain models. In *Eighteenth National Conference on Artificial Intelligence (AAAI'2002)*, Edmonton, Canada., 2002.
11. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000)*. Morgan Kaufmann Publishers, San Francisco, CA, 2000.
12. E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Distributed and Parallel Databases—An International Journal*, 8(2), 2000.
13. P. Mitra, G. Wiederhold, and M. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings Conference on Extending Database Technology 2000 (EDBT'2000)*, Konstanz, Germany, 2000.
14. N. F. Noy and M. A. Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, WA, 2001.
15. N. F. Noy and M. A. Musen. PromptDiff: A fixed-point algorithm for comparing ontology versions. In *Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, Edmonton, Alberta, 2002.
16. N.F. Noy and M.A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX, 2000.
17. Protege. The Protégé project, http://protege.stanford.edu, 2002.
18. G. Stumme and A. Mädche. FCA-Merge: Bottom-up merging of ontologies. In *7th Intl. Conf. on Artificial Intelligence (IJCAI '01)*, pages 225–230, Seattle, WA, 2001.
19. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. OntoEdit: Collaborative ontology engineering for the semantic web. In *International Semantic Web Conference 2002 (ISWC 2002)*, Sardinia, Italia, 2002.

Using Protégé-2000 in Reuse Processes

H. Sofia Pinto[1], Duarte Nuno Peralta[1], and Nuno J. Mamede[2]

[1] Grupo de Inteligência Artificial, Departamento de Eng. Informática
Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa, Portugal
{sofia,duarte}@gia.ist.utl.pt

[2] L^{2}F INESC-ID/IST - Spoken Language Systems Laboratory
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{Nuno.Mamede}@inesc-id.pt

Abstract. There is already a considerable number of ontology-based tools. In order to better choose the appropriate tool and to better use its capabilities, tools need to be compared and the experiences in using them should be shared. We have been using Protégé-2000 in an ontology reuse experience. In this case, we reused an ontology kept in the Ontolingua Server library. In this article, we report our experience in using the import translators for OKBC and the support provided by the tool in revision/extension processes. The analysis provided in this article is from an user point of view.

1 Introduction and Motivation

There is already a considerable number of ontology-based tools. At this moment these tools are attracting an increasing number of new users of all kinds, from naive users to power users. As more tools become available, the problem of interoperability between different tools becomes more important. Different tools will be combined in different ways to support varied and increasingly more complex ontology related processes. For instance, one may import an ontology available in the library of a particular ontology building environment into another ontology building environment and extend the imported ontology with more knowledge. The resulting ontology may then be translated into another knowledge representation language and introduced in an evaluation tool.

In order to better choose the appropriate tool and to better use its capabilities, tools need to be compared and the experiences in using them should be shared. This analysis has got to be performed in much more detail than a brochure-like collection of features, specially for power users. That is, we must know the limits of available technology.

We have been using Protégé-2000 [8] to build an ontology. In this case, the ontology was built through a reuse process. We reused an ontology kept in the Ontolingua Server library [4]. In this article, we report our experience in using the import translators for OKBC [2] of Protégé-2000 and the support provided by this tool in reuse processes.

In this article, we begin by referring existing evaluation studies about ontology-based tools and their shortcomings. Then we describe the context of our experience and the actual reuse process that was performed using Protégé-2000. We describe the problems and strong points of Protégé-2000 in our case-study. Finally, we explain the reasons underlying these problems and analyze related work.

2 Evaluation of Ontology-Based Tools

There are already some studies evaluating ontology-based tools in the literature, like WonderTools [3] and the survey on ontology-based tools of OntoWeb [9].

In [3] an evaluation framework is proposed and a comparative study of several ontology-building tools was made. This was the first systematic evaluation/comparison study of ontology-based tools, more precisely of ontology building tools. However, one important dimension that is not analyzed by this framework is the study of the interoperability between the different tools. For instance, the existence of import/export translators is not analyzed.

In [9] a general survey of ontology-based tools is presented, namely for ontology building, merge, evaluation, annotation, and storage and query tools. This survey proposes an evaluation framework for each kind of tool, and compares each tool against the corresponding framework. Although this survey provides an evaluation of each kind of tool, the interoperability between different tools is not analyzed in detail. For instance, the limitations of current translators are not analyzed.

3 Context of the Experiment

We are involved in the development of ontologies to be used in a Natural Language dialogue system. This dialogue system is to be placed in a bus terminal, as a ticket-vending/information machine. An important requirement of this application is that the language must be Portuguese (although the concepts represented in an ontology are, in general, language independent,[3] the terms used to refer to those concepts are not). The construction of this ontology will involve several subontologies in different domains related to traveling, for instance commercial transactions (buying and selling), geographical information and time.

We began the development of this ontology with the subontology of time. One of the requirements that was a-priori imposed was that the ontologies should be developed in a locally installed tool. Therefore, the tool that was chosen was Protégé-2000.

[3] In Portuguese there is a concept that finds no parallel in other languages: “Saudade”, which is a kind of nostalgia, home sick, is a genuine Portuguese concept.

4 A Reuse Process in Protégé-2000

The building process of the ontology of time has already went through several stages. In Fig. 1 we represent part of this process. The activities performed with Protégé-2000 are shadowed.

Import from Ontolingua To build our ontology, we reused the Simple-Time ontology[4] from the Ontolingua Server. We used the OKBC tab plug-in[5] to import the ontology into Protégé-2000.

Analysis - identification of missing and misplaced knowledge Before performing the analysis, we had already applied a manual reengineering [1] process to the source code. The result of this process was one possible conceptual model for Ontolingua's Simple-Time ontology. At this stage, we compared this model with the translated version of the ontology. We found that the translation process provided the taxonomic hierarchy of concepts (classes+instances). However, it did not translate all Ontolingua functions and it did not translate any relation or axiomatic definition. Moreover, some functions were misplaced.

Revision - rearrangement and extension of knowledge The knowledge we found misplaced in the ontology was relocated. In what concerns missing knowledge, we introduced the relations and functions. Following the guidelines provided in the documentation of Protégé-2000, we left the introduction of axioms for a later stage.

Analysis - technical evaluation of source ontology Having the taxonomy, relations and functions, we evaluated the Simple-Time ontology in the Ontolingua Server according to the criteria proposed in [6]. For that we used both the source code and the conceptual model. The reasons why we used both were: (1) if we only use the conceptual model we are not able to analyze the syntactical errors[6] (language conformity) and (2) if we only use the source code we loose the overall perspective of the ontology which is crucial to perform a thorough analysis.[7]

Revision - Correction, Natural Language Translation and Extension The problems found in the source ontology were corrected in the version kept in Protégé-2000. Then we translated all the names of the knowledge pieces represented in the ontology into Portuguese. Finally, the axioms were added to our ontology using Protégé Axiom Language (PAL). Moreover, some concepts, that are needed to describe the time domain for our particular application, were added.[8] For instance, half an hour.

[4] It defines 14 classes, 209 instances, 17 relations and 14 functions.

[5] `http://protege.stanford.edu/plugins/okbctab/okbc_tab.html`

[6] For instance, the `equals` relation in Ontolingua is defined for both `time-points` and `time-ranges` (polymorphic refinement). However, there is an error, since the definition for `time-ranges` introduces two variables that are not declared.

[7] For instance, without the conceptual model, we could miss the fact that functions `month-of` and `month-name-of` are the same function.

[8] The description of the requirement specification performed in this development process is out of the scope of this paper.

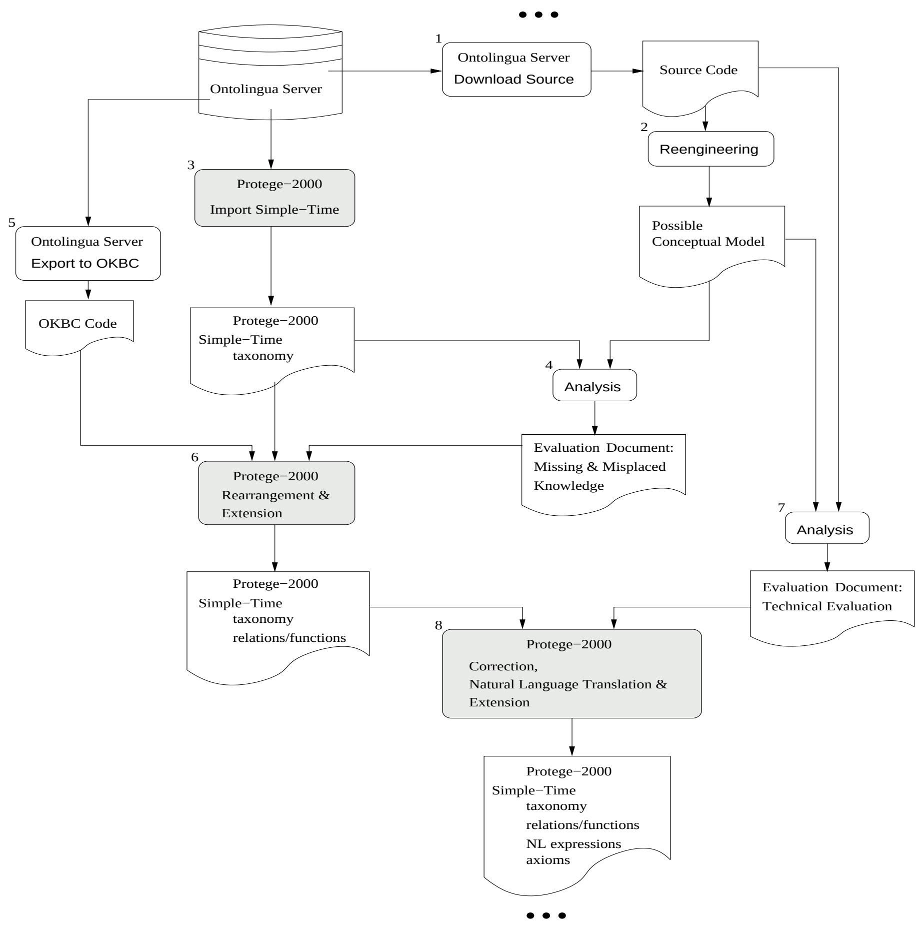


Fig. 1. Building process

In the following sections we analyze the use of Protégé-2000 to perform this process. The focus is placed on the advantages and disadvantages of using Protégé-2000 to build our time ontology. Since the tool was not used in the analyses stages of this process, we will not address them in this paper.

4.1 Problems after Importing from Ontolingua

To import the Simple-Time ontology from the Ontolingua Server, the OKBC tab plug-in was used. We found that part of the knowledge represented in the source ontology and imported using this translator was lost.

Comparing the definitions of the same concepts after using Ontolingua's export translator and after using Protégé-2000 import translator we can see that there are important differences. Take, for instance, the definition of the `Meets`

```
(define-relation Meets (?time-range-1 ?time-range-2)
"a time range ?time-range-1 ends at the same time a time range ?time-range-2 starts."
    :iff-def (Equals (End-Time-Of ?time-range-1)
                     (Start-Time-Of ?time-range-2)))
```

Fig. 2. Definition of relation `Meets` in Ontolingua

relation in Ontolingua, Fig. 2. The Ontolingua OKBC export translator represented this binary relation as a slot, Fig. 3. However, the OKBC import translator of Protégé-2000 lost this relation. Since axioms are outside the OKBC model, the axioms defining the relations were not translated, for instance the axioms defining the relation `Meets`.

Although one can argue that relations (and functions) are also outside the OKBC knowledge model,[9] the same problem seemed to affect the translation of slots. For instance, we tried to import the Agents ontology from the Ontolingua Server. The `agent` class is defined as a frame with a template slot `name`. After importing this ontology into Protégé-2000, all slots were lost.

Some of the functions were imported, but not all. For instance, the function `+` involving `time-points`, `time-ranges` and `durations` was lost. Moreover, the functions that were imported, were not translated as expected. For instance, the function `year-of` is defined in the source ontology with domain `time-point`. However, after the translation, the slot that was created was not attached to the `time-point` class. It was only attached to the classes `calendar-year`, `calendar--date` and `universal-time-spec`. These concepts were characterized in their Ontolingua [7] definitions by having one (`has-one`) `year-of`. Moreover, the minimum cardinality constraints of the slots corresponding to these functions were lost.

Summarizing, the problems we found in the translation of the Simple-Time ontology were:

- All knowledge represented using the `define-relation` primitive was not translated, for instance the relation `Meets`.
- Some functions were imported, but most were lost. The only slots that were created in Protégé-2000 correspond to functions that appear in the definition of classes using the `has-one` primitive. Although the slot was created, it was attached to the incorrect class (when compared to the source code) and the minimum cardinality constraints were lost.

4.2 Rearrangement and Extension of Knowledge

In the beginning of this stage we had just identified what knowledge had been lost and misplaced. At this time, we decided to rearrange misplaced knowledge and only add the missing relations and functions.

[9] Classes, instances, slots and facets.

```
(define-okbc-frame Meets
                   :frame-type :slot
                   :direct-types (Relation Binary-Relation)
                   :own-slots ((Arity 2))
                   :primitive-p common-lisp:nil
                   :sentences
                   (...
                    (=> (Meets ?time-range-1 ?time-range-2)
                     (Equals (End-Time-Of ?time-range-1)
                             (Start-Time-Of ?time-range-2)))
                    ...))
```

Fig. 3. Definition of relation `Meets` in OKBC using Ontolingua's export translator

Part of the rearrangement and extension done at this stage is shown in Fig. 4. For example, with the translated version of the ontology, it was not possible to characterize the day of a given `time-point`. Since the class of `time-points` lost all its attributes, we could not partially characterize specific `time-point` instances. This can be done in the Simple-Time ontology at the Ontolingua Server. The only classes for which we could do this were `calendar-year`, `calendar-date` and `universal-time-spec`.

The tool proved to be a good help, since relocating knowledge simply corresponds to a drag&drop action of the mouse. Like in a regular frame-based system, another way of relocating a slot is removing the slot from the class where it is wrongly associated and attach it to the correct class. In fact, the intuitive use of the tool is one of its main features.

To manually introduce the missing slots, we used the OKBC code produced by Ontolingua's export translator as a guide, Fig. 3. We simply created the slots that were defined in this translation and attached them to the appropriate classes. All relations (17) and lost functions (9) were introduced.

4.3 Correction, Natural Language Translation, and Extension

In the beginning of this stage we had a translated version of the Simple-Time ontology, consisting of the whole taxonomy, the relations and functions. We also had the technical evaluation document. The problems found in the analysis of the source ontology were corrected (Correction), the names in the ontology were translated into Portuguese (Natural Language Translation) and the axioms that were lost during the import translation from Ontolingua were added (Extension).

Correction consisted mainly in changing the ranges and domains of functions and relations. In Protégé-2000, this means relocating slots (change of domain) or changing the value of the `allowed-classes` facet of the slot (change of range).

After correcting the taxonomy and before starting to write axioms, the ontology was translated into Portuguese, because axiom definitions refer to names of frames in the ontology, and changes to these names are not propagated to the textual definitions of the axioms. One major advantage of using this tool to perform this task was that once we change the name of a frame, this change is immediately propagated through the entire taxonomy.

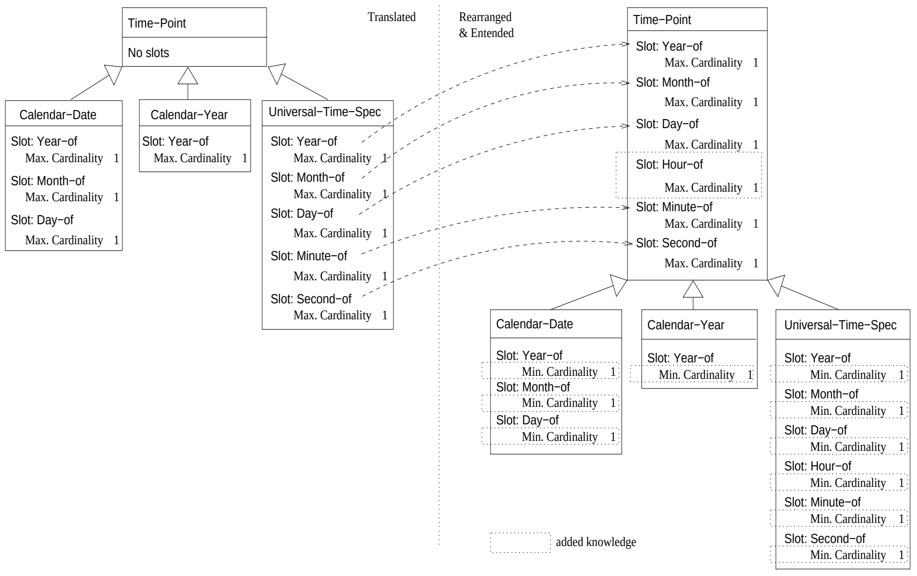


Fig. 4. Rearrangement and Extension of `time-point` and its subclasses

Finally, we added knowledge that was completely lost in the translation process: axioms. To write axioms in Protégé-2000 the PAL constraints tab plug-in[10] was used. PAL constraints are part of the PAL toolset plug-in for Protégé-2000. The idea is to allow the user to write constraints over the possible values of instances that cannot be represented using only classes, slots and pre-defined facets.[11] PAL is a limited predicate logic extension of Protégé-2000. Its syntax is similar to KIF [5]. However, statements like `defrelation` and `deffunction` are not supported. The PAL language is completely integrated with the Protégé-2000 knowledge model. Constraints are instances of the `:pal-constraint` class. When writing a PAL constraint, we can use any slot as a predicate, for instance `(start-time-of (time-range x) (time-point y))`. If the slot has a `maximum-cardinality` of 1, it can also be used as a function. In this case, `(start-time-of (time-range x))` represents `(time-point y)`. The PAL constraint checking mechanism can be called by the user, to show which constraints are violated and by which instances. It can also be called programmatically by an application that uses the Protégé-2000 API.

The PAL constraint checking mechanism also has a trace mechanism that allows us to follow the evaluation of a given constraint. This is very useful when writing constraints, since it helps to understand why they are not working as we

[10] `http://protege.stanford.edu/plugins/paltabs/PAL_tabs.html`

[11] For instance, `value-type`, minimum and maximum `cardinalit`ies, `minimum` and `maximum` values allowed.

```
(defrange ?time-range-1 :FRAME TIME-RANGE)
(defrange ?time-range-2 :FRAME TIME-RANGE MEETS)
(forall ?time-range-1
        (forall ?time-range-2
                (=> (and (MEETS ?time-range-1 ?time-range-2)
                         (own-slot-not-null END-TIME-OF ?time-range-1)
                         (own-slot-not-null START-TIME-OF ?time-range-2))
                    (= (END-TIME-OF ?time-range-1)
                       (START-TIME-OF ?time-range-2)))))
```

Fig. 5. Axiom written in PAL

think they should. However, this tracing mechanism only traces predicates and functions predefined in PAL language, for instance < for numbers, and only one at a time. It would be also useful to provide trace-ability of user-defined slots (treated in PAL as predicates or functions).

Although the PAL constraint checking mechanism is useful, it is not very easy to use at first. One of the reasons is because knowledge represented in KIF axioms that is not supported by PAL has to be transformed before it can be incorporated. PAL axioms consist on a set of variable range definitions and a predicate that must hold over those variables. An example of an axiom written in PAL is shown in Fig. 5.

This first user's reaction to PAL was not a very good one. This could be easily changed with a more detailed documentation. We found some examples on how to write a new constraint in the documentation, but some examples on how to transform an axiom written in KIF (or any other language) into a PAL constraint would also be useful.[12]

5 Translator Analysis

We analyzed the OKBC-tab plug-in code, in order to better understand why knowledge was being lost or misplaced. The plug-in starts by connecting to an OKBC compliant server (in our case, the Ontolingua Server) and then uses the OKBC protocol to get information about the ontology being imported. The main import procedure, `getClassDetails`, starts with an initial set of classes and then goes down the `is-a` hierarchy getting for each class its name, documentation, instances, template slots,[13] and for each template slot the value of the facets `value-type`, `maximum-cardinality` and `minimum-cardinality`. When reaching instances it gets their names, slots and slot values. This procedure leaves out any knowledge that is not explicitly represented in the definition of a class or instance frame. Moreover, any knowledge that is represented as an own slot in a class frame is also lost.

[12] Since our goal was to reuse the definitions implemented in KIF (or Ontolingua).
[13] The knowledge model of Protégé-2000 does not include own slots attached to classes.

In the case of the Simple-Time ontology all relations are represented in OKBC as slot frames and are not explicitly referred in the definition of any class frame. Therefore, all relations were lost.

However, this still did not explain why in the Agents ontology the slot `name` associated to the `agent` class was not translated. We found that the OKBC plug-in does not import frames whose names are keywords and `name` is a keyword in OKBC.

We also discovered the causes of misplaced knowledge. For instance, the `time-point` class is defined in Ontolingua as being the `domain-of` of the function `year-of`. Since the definition of the class `time-point` in OKBC only makes reference to the `year-of` function in the own-slot `domain-of`, the corresponding slot is not created. However, the class `calendar-date` is defined in Ontolingua as having one (`has-one`) `year-of`. In this case, the OKBC definition of `calendar--date` explicitly mentions that `year-of` is a template slot of this class. Therefore, the slot is created in Protégé-2000 and attached to the `calendar-date` class.

Knowledge about a slot is explicitly represented in the definition of a class frame in OKBC if in the Ontolingua definition of that class some specific primitives are used. If we use the `define-class` primitive we can use the `has-one` or `has-some` primitives to constraint cardinalities. If we use the `define-frame` primitive to define a class we can specify one of the following items: (1) the name of the template slot, (2) its `maximum-cardinality`, (3) `minimum-cardinality` or (4) `value-type`. Regarding cardinalities, Protégé-2000 only uses the values of cardinalities to assess whether the slot is multiple-valued. However, there are some problems. The maximum cardinality is always one and the minimum cardinality is not used at all.

To summarize:

- The import translator of Protégé-2000 does not import knowledge that is not explicitly represented in the definition of a specific class or of an instance in the Ontolingua Server's OKBC code.
- Frames (classes, instances and slots) whose name is a keyword of the OKBC protocol are not imported.
- All knowledge represented in own-slots of classes is not imported.
- Not all Ontolingua primitives can be used to explicit knowledge about template slots in the definition of a class.
- All facets, except `value-type`, are lost. Although the translator looks for the facets `maximum-cardinality` and `minimum-cardinality`, Protégé-2000 only uses that information to assess the multiplicity of the slot.

6 Related Work

A translation process involving the Simple-Time ontology in the Ontolingua Server is described in [10]. In this case the export translator of Ontolingua into Loom was used. At that time, the conclusions were that the automatic translators were still at draft level. Although they were useful to provide initial versions, considerable human interaction was needed to improve the automatic versions. The two problems found at that time were:

Mismatch of modeling styles The way knowledge is modeled in Ontolingua is different from the way it is usually modeled in Loom. The constructs provided by each language form a representation ontology that is different for each language. A translator between two languages must start by mapping between the two representation ontologies. In general, building a translator is easier if these ontologies are similar.

Inference engine bias Even if the knowledge is modeled without a specific application in mind, it is usually modeled considering certain inferences. For instance, in Loom, knowledge is usually modeled considering that it is going to be used by its built-in inference engine.

In our case study we found that:

- Both the Ontolingua Server and Protégé-2000 are OKBC-compatible, so there is no mismatch in modeling styles.
- However, the inference engine bias is very strong in our case. For instance, when it comes to axioms, the PAL language has a rather different style from KIF. The PAL toolset is a constraint-checking rather than theorem-proving mechanism. This means that PAL only checks constraints based on the instances in the ontology. So, axioms written in PAL have to make strong closed-world assumptions about the knowledge that is being modeled. Moreover, in order to simplify the axioms, we added a slot `representacao--numerica` (number-representation) to all classes that are subclasses of integers (`hour-number`, `year-number`, etc.). This was done because in PAL the relations `<`, `>` and `=` are already defined for integers.

7 Conclusions and Future Work

In this article we report a reuse experience that involved translation, rearrangement, correction and extension of an ontology using Protégé-2000. We tried to evaluate the support provided by this tool in reuse processes. We have found that the OKBC import translator of Protégé-2000, although useful for providing an initial version, is not ready to be used in a fully automatic translation process. We analyzed the source code of the OKBC-tab plug-in and discovered the source of the problems that we had identified.

Regarding usability, we have found the tool to be intuitive and easy to use. The tool eased our reuse process. In particular, we have found that it was more cost effective (time+effort) to use the tool rather than building the whole time ontology from scratch in Protégé-2000.

In future we plan to build other ontologies by means of reuse, namely the other subontologies needed for the natural language dialogue system, using the support of available tools. We also plan to improve the OKBC import translator of Protégé-2000.

8 Acknowledgements

We thank the anonymous reviewers for their usefull comments. We also thank the support provided by the technical support teams of Protégé-2000 and Ontolingua Server. We thank FCT.

References

1. Blázquez, M., Fernández, M., García-Pinar, J. M., Goméz-Pérez, A.: Building Ontologies at the Knowledge Level Using the Ontology Design Environment. In: *Proceedings of the Knowledge Acquisition Workshop* (KAW98), Banff, Alberta, 1998.
2. Chaudri, V., Farquhar, A., Fikes, R., Karp, P., Rice, J.: *Open Knowledge Base Connectivity 2.0.3*. Knowledge Systems Laboratory, KSL-98-06, Stanford University, 1998.
3. Duineveld, A. J., Stoter, R., Weiden, M. R., Kenepa, B., Benjamins, V. R.: Wondertools? A comparative study of ontological engineering tools. In: *Proceedings of the Knowledge Acquisition Workshop* (KAW99), Banff, Alberta, 1999.
4. Farquhar, A., Fikes, R., Rice, J.: The Ontolingua Server: A Tool for Collaborative Ontology Construction. In: *Proceedings of the Knowledge Acquisition Workshop* (KAW96), Banff, Alberta, 1996.
5. Genesereth, M.: Knowledge Interchange Format. In:J. Allen and R. Fikes and E. Sandewall (eds.): *KR91 Proceedings*, Morgan Kaufmann: 599–600, 1991.
6. Gómez-Pérez, A., Juristo, N., Pazos, J.: Evaluation and Assessment of the Knowledge Sharing Technology. In: N.J.I. Mars (eds.): *Towards Very Large Knowledge Bases*, IOS Press: 289–296, 1995.
7. Gruber, T.: A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, **5**: 199–220, 1993.
8. Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., Musen, M.: Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems* **48**(2): 60–71, 2001.
9. OntoWeb: *Deliverable 1.3: A survey on ontology tools*, 2002.
10. Russ, T., Valente, A., MacGregor, R., Swartout, W.: Practical Experiences in Trading Off Ontology Usability and Reusability. In: *Proceedings of the Knowledge Acquisition Workshop* (KAW99), Banff, Alberta, 1999.

Integrating Ontology Storage and Ontology-based Applications Through Client-side Query and Transformations

Peter Mika

Vrije Universiteit, Amsterdam
pmika@cs.vu.nl

Abstract. This paper investigates the integration of ontology storage and ontology-based applications through the example of the EnerSearch case study conducted within the On-To-Knowledge research project. We look at the problem of integrating the Sesame storage and query facility with its client application and identify both functional and technical needs for a new software package for client-side query and transformations. We introduce the solution developed during the case study that also opens the way for creating Portable Inference Modules that capture transformation knowledge in a modular way. We discuss future extensions to this package based on the belief that the issues at hand will equally effect future Semantic Web applications.

Introduction

The World Wide Web has drastically changed both the form and availability of information in the past years. As the number of web pages and users on the public internet skyrocketed, companies around the world have just as eagerly adopted internet technologies as a basis for their own networked, electronic information stores. The resulting intranets, filled with weakly structured, weakly organized information, have created a knowledge management problem that could not be any more handled by existing document management solutions.

The On-To-Knowledge research project [1] is a joint, EU-funded research effort that aims to improve on the state of the art of web-based knowledge management solutions for SMEs and distributed organizations by leveraging ontologies. Instead of building a complete knowledge portal, the approach of On-To-Knowledge is to provide for the interoperability of the components developed within the project, based on open standards and agreements among the partners, such as a common data model for representing domain knowledge. The aim of the case studies within On-To-Knowledge is to validate this approach by showing that it is indeed possible to integrate these components into customized solutions that fit the specific knowledge management problems of the case study partners.

EnerSearch, a pan-European research organization investigating new IT based business strategies and customer services in deregulated energy markets, carried out the case study that we will use as an example in this paper. The company joined On-To-Knowledge for what ontologies promise with respect to greater user satisfaction through more effective querying of its corporate memory. The status of EnerSearch as a virtual knowledge organization warranted that the value attributed to finding the right information is high enough for an increased interest in state-of-the-art semantic solutions.

The ontology developed in the case study is a combination of a lightweight domain ontology obtained by natural language processing with OntoExtract [2] and a rich ontology reverse engineered from the publication database of EnerSearch[1]. The ontology, which currently contains over 140,000 statements about approximately 20,000 resources, is represented in RDF(S) format and stored in a repository at the central Sesame storage server [3]. It provided semantic data for the searching and browsing interfaces that were generated using the QuizRDF [4] and Spectacle [5] tools, respectively.

The case study provided ample evidence that ontology-based tools should not be evaluated as standalone applications, but should be qualified as part of integrated frameworks or applications. In the following we discuss our experience in application integration to demonstrate the relevance of such an approach.

In the course of the case study we identified a bottleneck in integrating the ontology storage facility and the ontology-based applications: the division of query and transformation work between client and server inhibited creating applications that would have scaled up to industrial standards. Moreover, support was lacking for custom inference using semantics that is available only at the client side.

This paper presents a detailed description of the issues at hand and proposes a solution to address the need for client-side query and transformations. In the following section we take a closer look at the workings of ontology storage facilities through the example of the Sesame storage server. This will lead us to a better understanding of the observations that follow in Section 0 with regard to the present approach to query and transformations. These observations also form requirements in that they point to the necessity of client-side query and transformations.

The solution that was developed during the case study is presented in Section 0. We also show that this solution finds an even wider application in creating Portable Inference Modules (PIMs). These modules not only serve to improve efficiency, but enable the sharing of transformation knowledge by capturing it in a portable way. We outline future work in Section 0 and offer some conclusions in the closing section of the paper.

[1] The database contains metainformation about the documents such as the author, title, publication date etc. It has been used on the current website to render a table of the publications.

The Sesame RDF(S) storage and query facility

In many early Semantic Web initiatives semantic data was embedded within HTML or XML pages or it was simply stored within separate files and served to clients by web servers (cf. SHOE [6] or OntoBroker [7]). However, as ontologies became all the more valuable and ontology-based applications started to appear, the need has arisen for specialized ontology servers to make data manipulation efficient and to provide advanced services such as querying, versioning, access control and security.

To better understand the problems related to ontology storage and query, it's helpful to take an in-depth look at how these facilities operate. For this purpose, we've chosen the Sesame storage and query server for its role in the EnerSearch case study and for its open architecture. Sesame, originally developed and further supported by AIdministrator B.V. in the Netherlands, has become open source since March 2002.

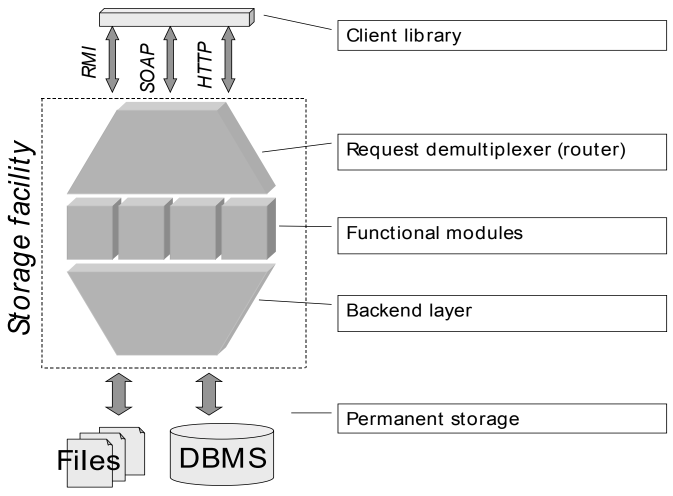


Figure 1. Overall design of web-based storage facilities

Although actual implementations differ, ontology storage facilities share the common design of web-based storage facilities shown in Figure 1. In the following, we will describe how Sesame implements the components of this design.

As all Semantic Web tools, Sesame primarily communicates with clients through HTTP, though in the future some services will be offered through SOAP and RMI as well. Sesame also has a client API, which is a Java programming interface that abstracts remote communication. In other words, programmers can use the API to access the services of Sesame in their ontology-based applications without having to deal with the actual communication between client and server.

The next layer, the request router forwards requests to the functional modules of Sesame. The server currently implements the following operations on repositories:

1) *Data manipulation*

   a) Upload of RDF(S) documents. Currently only RDF-XML format is supported.
   b) Extraction of ontology and data in RDF(S). It's possible to separately extract the ontology (schema) and the instance data.
   c) Removal of statements.
   d) Clearing the entire repository.

2) *Query*

a) Formal queries. Support for two RDF query languages have been implemented, namely RQL [8] and RDQL [9], with RQL being strictly more expressive than RDQL.[2]
b) Browsing of the repository. The user can navigate through the RDF graph model by clicking through the nodes that are of interest.

From the applications point of view, the significance of the query engines is that they provide access to the content of the repository on the level of the data model, i.e. the RDF graph model complemented with the semantics of the reserved vocabulary of RDF(S).[3] This means that queries may not only match triples, but can also refer to paths in the graph model or elements of the schema. For the software engineer this allows to formulate complex queries in a relatively simple language.

A support function to querying is the built-in inference module of Sesame that applies the RDF(S) closure rules [10] to data, thereby calculating the complete RDF model of the repository. At present inferencing is applied at upload time, where the alternative would be to run the inference module at query time. The difference is that the present solution suits query intensive applications better as opposed to applications that rely on frequent manipulation of the data.

The functional modules access the repositories through another abstract layer called SAIL. The purpose of the SAIL API is to hide the implementation of permanent storage. Storage SAILs exist for relational databases (PostgreSQL, mySQL) with several others under way. Development plans call for in-memory and peer-to-peer network storage.

Revisiting the design shown in Figure 1, the reader may notice that ontology-based storage facilities such as Sesame only differ from other web-based storage systems in their functional modules. More specifically, ontology servers are partially aware of the semantics behind the data which enables them to offer services such as query and reasoning. However, as we will see in the next section, server-based query and transformations do not always provide an ideal solution for building scalable ontology-based applications.

Ontology storage, query and transformations in the EnerSearch case study

As mentioned above, in the framework of the EnerSearch case study Sesame was used to store the lightweight domain ontology extracted by OntoExtract and the ontology obtained by converting the publication database of EnerSearch. This repository was then queried for the semantic data that was needed by the two ontology-based applications, the QuizRDF search engine and the Spectacle presentation. Through the course of building these applications, we have made six observations that led to the development of a new package for client side query and transformations. Note that these observations are generic on purpose. In particular, nothing is claimed about the absolute performance or scalability of Sesame, for such statements could only be made within a framework and in comparison to some other product, theoretical limit or baseline.

Observation 1: Small, frequent queries are inefficient.

The first, naïve implementation of the Spectacle application called for querying Sesame on an on-demand basis, i.e. whenever the value of a property or the subclass(es) of a class became relevant. While testing the application with ontologies containing more than a hundred classes, it was found that posing small, frequent queries to Sesame presents a serious bottleneck in generating the presentation. Even though

[2] Note that parallel to the development of ontology representation languages, query languages go through an intensive phase of development. Moreover, there is no widely accepted RDF(S) query language yet that would also have the sanction of a standardization body.

[3] Similarly to how relational query languages operate on tables and XML query languages work on the XML tree, regardless of external representation.

the users would not encounter the problem[4], runtimes on the scale of days were deemed unacceptable even for off-line website generation.

Here, the bottleneck is caused by the communication costs of accessing the server through HTTP.

Observation 2: Some queries that are useful from an application perspective cannot be expressed or efficiently executed.

As it turned out, some of the more advanced queries also suffer from slow evaluation times. An example of such a query is the following RQL select statement that returns the concepts from the ontology along with page(s) where they appear.

```
select c, p from {p} oe:isAbout . rdf:type {c} . rdf:type { concept }

where concept = oe:Concept

using namespace

   oe = http://ontoserver.cognit.no/otk_rdf# ,

   rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#
```

The path expression in the query matches a three edge long path in the graph and returns two points along the path. Despite its relative simplicity, evaluation times for this query ranged over a few dozen minutes even for an ontology with few hundred concepts.

An ever more challenging problem turned out to be querying for all direct subclass relations within an RDF(S) model. While such information is readily available in the level of the SAIL API, there is no direct support for it in the RQL language. More specifically, while there is a query with the intended meaning, evaluating it calls for matching all possible pairs of classes. Again, the net effect is an execution time that is unacceptable for sizeable ontologies.

Finally, there are transformations that require data manipulations that cannot be expressed in a declarative language, but require the full power of a programming language. We will see such an example in the following section when transforming a literal value of comma-separated words into several separate relations.

At the moment steps are being taken to optimize both the query language and the evaluation of queries. Naturally, there will always be inefficient queries in every sufficiently expressive language; the goal here is to optimize the queries that are useful from an application perspective. Unfortunately, there are very few real applications to provide feedback to that work.

It also seems that the technical barrier to optimization is the cost of communication with the underlying permanent storage, although an in-memory storage implementation could remedy the situation at the expense of size scalability.

Observation 3: The smaller the repository, the faster a transformation executes.

While this seems a trivial observation, its consequences are far reaching. Consider the case of applying the RDF(S) closure rules to data that is being uploaded to Sesame.

Figure 2 shows the processing speed for a series of 17 consecutive uploads to an originally empty repository for two implementations of the RDF inference engine. For the 'old' series we used the first, naïve implementation, while values for the 'new' series were obtained using an optimized version of the inference module. The optimizations, for example, streamline communication with the underlying database and employ heuristics such as dependencies between the inference rules.

[4] In the current version, both the navigation and content of the Spectacle website is rendered in advance, and server only adds the design at request time. For a truly dynamic site slow queries could hinder the user experience as well.)

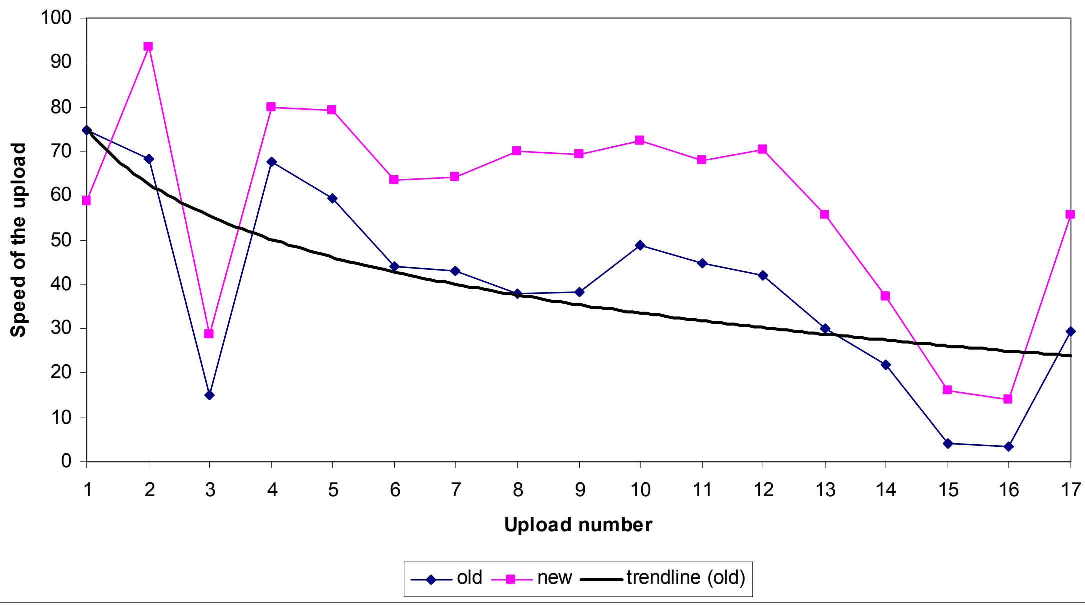


Figure 2. The total number of statements per second over a series of 17 uploads for the old and new implementation of the inference engine, with a logarithmic trend line for the former.

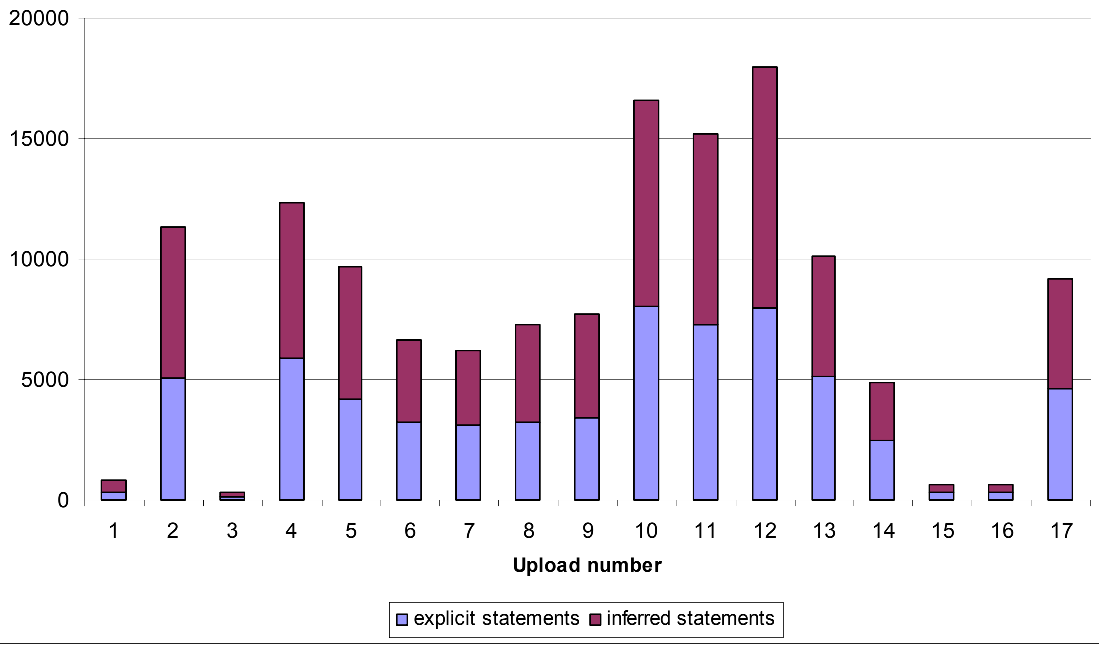


Figure 3. Size of the uploads as the total number of statements.

The logarithmic trend line fitted on the graph reveals that the time needed for the inference is proportional to the number of statements already in the repository. As the figure also demonstrates, improvements over earlier implementation of the inference module considerably alleviated the situation and most of the time during upload is now actually spent on parsing the input and adding the parsed statements to the repository. For relational data stores, it would be even possible to move some of the computation to the database server in the form of built-in procedures at the expense of portability. Nevertheless, the basic observation would remain the same, due to the very nature of inference: every added statement has to be matched against the rules and the statements in the repository to see if it results in new, inferred statements.

Observation 4: Server-side transformation is inefficient for small datasets.

Looking at Figure 2, the reader might have wondered what is the reason behind the variation in the upload speed, i.e. the noise that seems to be superposed over the general declining trend. Figure 3 provides the clue: the size of the respective uploads in Figure 2.

The sharp dips on the earlier figure correspond to comparatively small uploads on the order of a few hundred statements compared to several thousand statements for the other uploads. However, the difference in the time needed for the upload to complete was not nearly as big as the difference in sizes. In other words, the speed of processing smaller uploads is significantly lower then for processing larger ones.

Similarly to Observation 1, the explanation concerns the fixed administration costs of carrying out an upload. The consequence for the applications that rely on frequent manipulation of smaller parts of the dataset is again a performance problem.

Observation 5: Transformations weigh heavily on the storage facility.

Although this cannot be seen from the previous figure, reasoning also consumes considerable processing power and makes the storage facility much less responsive to other requests during the inference process. Inference over more expressive languages such as DAML+OIL [11] will be even more costly and might seriously set back performance with respect to base functionality.

Observation 6: There is a distinct need for client side transformations.

Even if server side query and transformations could be made efficient at will, clients are in all cases aware of much more of the semantics behind the data – semantics that in many cases cannot be described in today's ontology languages, yet do not warrant reasoning with a full scale predicate logic.

A typical example from the case study is the situation when a new relation is composed from other properties. For example, if we would like introduce an occursIn relation between concepts and pages we would need to compose three relations using the following rule:[5]

$$\exists \text{instance}\,(\text{page}, \text{oe}:\text{isAbout}, \text{instance}) \wedge (\text{instance}, \text{rdf}:\text{type}, \text{concept}) \wedge$$
$$\wedge\, (\text{concept}, \text{rdf}:\text{type}, \text{oe}:\text{Concept}) \rightarrow (\text{concept}, \text{oe}:\text{occursIn}, \text{page})$$

Such composition cannot even be described in DAML+OIL. Certainly, languages could always be extended to allow for encoding all kinds of rules and axioms (up to the expressiveness of full logic), but we would most likely still prefer a less expressive language considering the lightweight nature of the ontology.

Moreover, if the transformation includes resources or properties from separate repositories, inferencing cannot be done at any single server. How to execute queries on disparate servers is an oft-neglected research area, even though it is paramount to realizing the Semantic Web.

The above observations compelled us to develop the core of a new API for client-side query and transformation. This package –code named on2k.graph- is presented next.

The on2k.graph package

The on2k.graph package is a generic Java API that operates on a powerful graph paradigm to support programmatic query and transformations of RDF(S) models. The solution itself is based on a freeware Operations Research package from DRA Systems.

[5] The reader might conclude that the above RQL query returns those concept and page pairs that satisfy the precedence of this rule. In fact, many reasoning tasks and constraint checks can be rewritten as simple actions/checks on the result of queries.

The API is geared toward importing, manipulating and exporting mono-property graphs, i.e. graphs where every edge is labeled using the same property. This design decision is motivated by

- Performance issues. Ontologies, especially lightweight ontologies, typically contain only a select number of properties. Mono-property graphs are therefore ideal cross-sections from a performance point of view: importing them is efficient and once the transformation has been carried out then the result can be exported just as efficiently in the form of such graphs.
- Functional issues. Queries and transformations typically involve only a select number of properties and the resources related by them. Some examples from the EnerSearch case are presented later in this section.

Therefore, the solution addresses both the performance and functional concerns that have been listed as observations in the previous section.

The package has generic interfaces to import and export graphs using any source or target that is capable of producing or consuming statements with a given predicate. Actual implementations of these interfaces are given for Sesame using the client library.

Support is also provided for the mapping of ontological classes to Java classes. (At present the programmer provides these classes, although it would be possible to compile them from their descriptions as done in the work of Cranefield [12].) These dynamically instantiated classes are used to hold literal properties and to provide operations on literal properties. For example, in the EnerSearch domain the class Publication provides a getDate() operation that returns the date of the publication.

Simple graph manipulations such as taking the union or intersection of graphs and the support for symmetric properties are provided by the underlying graph environment. Additional expressiveness, however, is kept to the minimum, because extra expressive power could become an overhead for applications using lightweight semantics. Instead a helper class built using the functionality of the package provides basic operations such as determining subclasses, direct subclasses, superclasses and instances of classes. (Types of instances are readily available without this class.) A more extensive library of queries, inferences, constraint checks might be added later and invoked on an on-demand basis.

Using on2k.graph in the EnerSearch case

Originally, the package has been developed within the case study to optimize communication with Sesame when generating the Spectacle presentation, but it has been later employed to facilitate simple transformations as well.

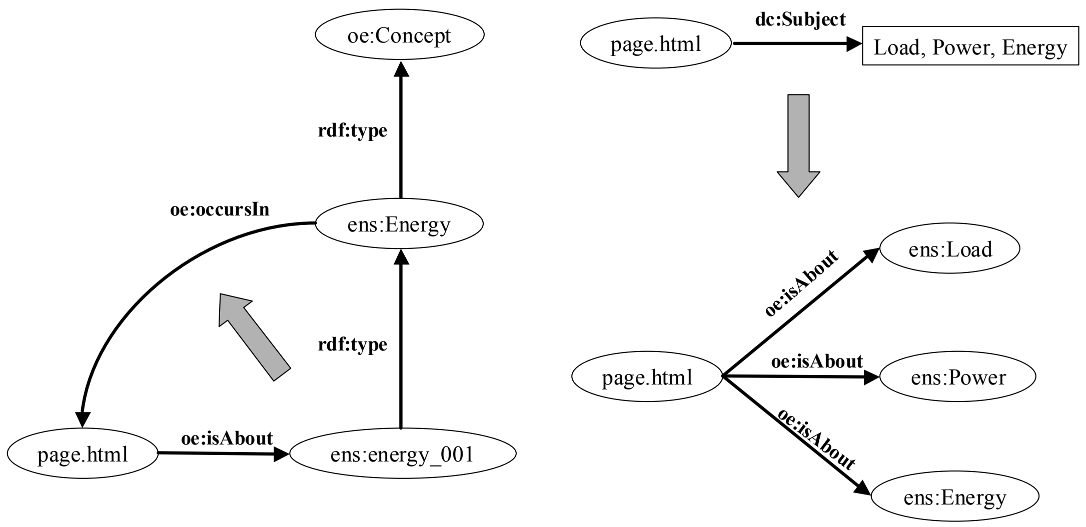


Figure 4. An example of a simple inference and a transformation on the domain ontology.

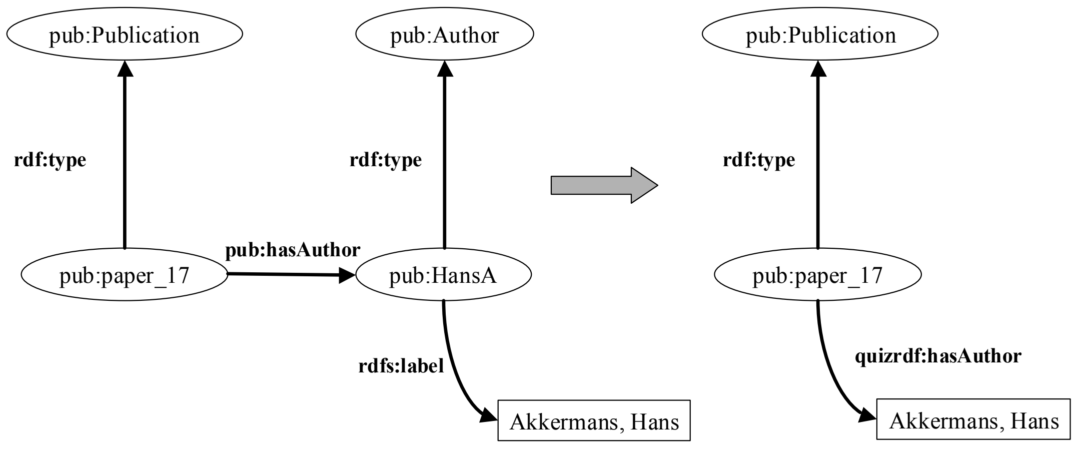


Figure 5. An example of a transformation on the publication ontology.

Some examples of the transformations carried out are shown on Figure 4 and Figure 5. On the domain ontology, we infer the occursIn relation as mentioned before and convert the comma separated list of values in the Dublin Core [13] Subject field into relations between pages and concepts. (See Observation 2 in Section 0.) Furthermore, to transform the publication ontology into a form preferred by QuizRDF, we transform the relation between a publication and an author into a literal attribute of the author resource. Other examples not shown on the figures include inverting relations and filtering out non-key concepts.

All transformations work in three steps, (1) querying the necessary graphs from Sesame, (2) carrying out the transformations and (3) uploading the resulting graph(s) if necessary. For better performance, the output of a transformation may be kept in memory if it's used by another transformation, as is the case with the ones shown in Figure 4. Currently, there is no automated discovery of such a dependency (or trigger), although this information could be inferred based on the input/output relations of our transformations.

The on2k.graph package not only enabled client side transformation (see Observation 6), but also allowed the expressiveness mentioned in Observation 2. It greatly improved performance and made it possible for our application to scale to the level of the EnerSearch ontology (addressing the issues in Observations 1, 3 and 4.) The overburdened central Sesame server was relieved from the effects of transformations (see Observation 5), except for RDF inference. On the client side memory requirements have never exceeded 100 MB, not even with all resources kept in memory as Java classes.

Portable Inference Modules

The queries and transformations implemented using the on2k.graph package do not need to be limited to the scope of a single application. In fact, once there is an agreement on the programmatic representation of the data model of the ontology language, the procedures that operate on this model become what we call Portable Inference Modules. PIMs are a sharable, reusable form of transformation knowledge captured in a procedural form.

PIMs need not to be constrained to represent custom semantics either. Semantics of RDF-S, for example, can be fully described as a set of inference rules that operate on the RDF model [10]. PIMs can be used to capture the built-in inference rules and constraints of ontology languages, thereby making it possible to use only the subset of a language that is necessary for an application or to mix-and-match functionality from various ontology languages.

Moreover, if PIMs are trusted and fine grained enough to be explanatory, a series of such modules can be used to support a chain of reasoning, since they provide evidence that can be checked if necessary by executing them over the data. Looked at from another perspective, PIMs can be taken as a compression mechanism: applications only need to transfer the explicit data and the PIMs, because the full model of the data (i.e. all statements that are valid) can be reconstructed from these components.

PIMs can also be considered for use on the server side as the semantic equivalent of stored procedures.[6]As we know it from the database world, such server-side transformations are indeed justified in many scenarios.

The concept of Portable Inference Modules differs from Semantic Patterns introduced by Staab, Erdmann and Maedche [15] in several respects. Semantic Patterns are defined on a higher level of abstraction: while PIMs capture inference knowledge, patterns concern themselves with design knowledge. Patterns consist of a natural language description and a set of formal constraints on their instantiations. (There are four types of constraint relating to what an instantiation must, must not, should and should not entail, based on the input.) PIMs, on the other hand, are given in a programmatic form (although they might be accompanied by formal descriptions in the future), which means that they work on the level of the representation model of a specific language (RDF). Due to the higher level of expressiveness PIMs are expected to be practical building blocks of Semantic Web applications, while semantic patterns are better suited for communicating, cataloguing, reverse-engineering and problem-solving on the design level.

Future work

The major value driver of the Semantic Web will be its applications ability to reason over data. Unlike software using conventional databases, semantic applications not only reuse previously stored data, but base their workings on facts inferred from data.

Despite its significance, there is very little attention paid to how inference knowledge will be represented on the Semantic Web. Even the most expressive of ontology languages such as DAML+OIL provide only limited ways to express axioms. Although they could be extended up to the expressiveness of first order predicate logic, the resulting language would be a large overkill for applications operating with lightweight ontologies, such as the system developed within the EnerSearch case study.

However, if at some point in the future an agreement is reached on how transformation knowledge should be represented in a declarative way, it may become possible to compile declarative descriptions into procedural form. Alternatively, such descriptions may accompany PIMs as an interpretation of their workings.

In a similar fashion parameterized rules could be translated into parameterized procedures. Note that many of the modeling constructs used in ontology languages can be interpreted as such: for example, the transitivity of a property can be considered a generic rule that is parameterized by a property. Once we have a 'transitivity PIM' available, and the transitivity of a property becomes evident, the PIM can be loaded, instantiated and executed to carry out the constraint check or inference.

In the late future rule types as the ones used in the CommonKADS methodology [14] for clustering similar rules may also be considered as templates for PIMs.

Conclusion

Due to their complexity and dynamic nature, Semantic Web applications of the future are likely to be loosely coupled, distributed or agent-based solutions woven from a variety of components and services. In such a setting efficient query and transformation of semantic data will largely influence the performance and scalability of applications.

As we have seen through the example of the EnerSearch case study conducted within the On-To-Knowledge project, the present division of query and transformation between the ontology storage and query facility and its clients presents a significant bottleneck. The solution is to move part of these tasks to the client, which is also aware of much more of the semantics behind the data. Besides greater efficiency, this also opens the way to the creation of portable transformation modules that can be shared between agents and applications.

Transformation knowledge is key to creating interoperable semantic applications. Therefore, when captured in the right form, it may become a valuable commodity on the Semantic Web.

[6] Note that "server side" is a relative notion: storage engines such as Sesame can be embedded into applications, if required.

Acknowledgements

The author would like to express his gratitude to Frank van Harmelen (VU), Jeen Broekstra (VU, AIdministrator) and Arjohn Kampman (AIdministrator) for their valuable comments to this paper, with special thanks to Arjohn for his work with measuring and charting the performance of Sesame's inference engine.

References:

[1] The European On-To-Knowledge project (IST-1999-10132). See http://www.ontoknowledge.org.

[2] R. Engels. CORPORUM-OntoExtract: Ontology Extraction Tool. On-To-Knowledge Deliverable D6. See http://ontoserver.cognit.no.

[3] J. Broekstra and A. Kampman. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. On-To-Knowledge Deliverable D10. See http://sesame.aidministrator.nl.

[4] U. Krohn and J. Davies. The Search Facility RDFferret. On-To-Knowledge Deliverable D11. See http://www.ontoknowledge.org.

[5] Spectacle: White Paper on Advanced Information Disclosure, 2001. See http://www.aidministrator.nl/publications/SpectacleWhitePaper.pdf

[6] J. Heflin and J. Hendler. Dynamic Ontologies on the Web. In *Proceedings of American Association for Artificial Intelligence Conference 2000*. See http://www.cs.umd.edu/projects/plus/SHOE/

[7] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In R. Meersman et al., editor, *Proceedings of DS-8: Semantic Issues in Multimedia Systems*, Kluwer Academic Publisher, 1999. See http://ontobroker.semanticweb.org/

[8] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis & M. Scholl.
RQL: A Declarative Query Language for RDF. To appear in *Proceedings of the 11th International Conference on the WWW*, Hawaii, 2002. See http://zeus.ics.forth.gr/forth/ics/isl/publications/paperlink/dql-rdf.pdf.

[9] A. Seaborne. RDQL: A Data Oriented Query Language for RDF Models, 2001. See http://www.hpl.hp.com/semweb/rdql.html.

[10] P. Hayes. RDF Model Theory. W3C Working Draft, April 2002. Available online at http://www.w3.org/TR/2002/WD-rdf-mt-20020429/.

[11] F. van Harmelen, P. Patel-Schneider and I. Horrocks. Reference description of the DAML+OIL ontology markup language, March 2001. See http://www.daml.org/2001/03/reference.html.

[12] S. Cranefield. UML and the Semantic Web. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, 2001.

[13] Dublin Core Metadata Element Set, Version 1.1: Reference Description. Available at http://dublincore.org/documents/1999/07/02/dces/

[14] G. Schreiber, H. Akkermans, A. Anjewierden, R. Hoog, N. Shadbolt W. Van de Velde and B. Wielinga. *Knowledge engineering and management: The CommonKADS Methodology*. MIT Press, Massachussets, 1999.

[15] S. Staab, M. Erdmann and A. Maedche. Engineering Ontologies using Semantic Patterns. In *Proceedings of the IJCAI'01 Workshop on E-business and the Intelligent Web*. Seattle, 2001. See http://www.csd.abdn.ac.uk/~apreece/ebiweb/programme.html.

The integration of OntoClean in WebODE

Mariano Fernández-López, Asunción Gómez-Pérez

Facultad de Informática . Universidad Politécnica de Madrid
Campus de Montegancedo, s/n. 28660 Boadilla del Monte. Madrid. Spain
{mfernandez, asun}@fi.upm.es}

Abstract. Enterprises will only be interested in the use of ontologies if such ontologies are evaluated enough. Therefore, the development of ontology evaluation tools is a crucial matter. We have built the ODEClean module in the workbench for building ontologies named WebODE. ODEClean allows cleaning taxonomies following the OntoClean method, and WebODE provides technical support to the Methontology methodology for building ontologies. We approached the development of this module in two steps. Firstly, we have integrated the OntoClean method into the conceptualisation activity of Methontology. Secondly, we have designed and implemented ODEClean using a declarative approach for specifying the knowledge to be used on the evaluation. ODEClean uses: (a) the Top Level of Universals, (b) meta-properties based on philosophical notions, and (c) OntoClean evaluation axioms. The main advantage of this approach is that the system could easily allow the user relax or stress the evaluation of the taxonomy just selecting more or less meta-properties.

1 Introduction

Currently the semantic web [1] attracts researchers from all around the world. Numerous tools and applications of semantic web technologies are already available [2] [3] [4] and the number is growing fast [5]. Ontologies play an important role for the semantic web as a source of formally defined terms for communication. They aim at capturing domain knowledge in a generic way and provide a commonly agreed understanding of a domain, which may be reused, shared, and operationalised across applications and groups. The large visibility of the semantic web, its tools and applications already attracts industrial partners, e.g. in numerous projects funded by the European Commission. As they move from academic institutions into commercial environments they have to fulfil stronger requirements (e.g. concerning correctness, consistency, completeness, conciseness, etc.). Therefore, the evaluation is a key activity in ontology development. Some of the most well-known proposals for ontology evaluation are: Gómez-Pérez's proposal [14] [15] [16], Kalfoglou and colleagues' proposal [22][23], and OntoClean [26]. OntoClean is a method for cleaning tangled taxonomies founded in philosophical notions as: rigidity, identity, unity, etc.

Most of the methodologies and methods ([25], [18], [10], [24], etc) for building ontologies include an evaluation activity. Most of the times, ontology evaluation is done once the ontology is finished and implemented in a given ontology language. Methontology [12] proposes to evaluate the ontology during its whole life cycle: it recommends to carry out most of the evaluation of the content at the conceptualisation activity to prevent the detection of faults in the ontology code. WebODE [8] is the workbench that gives technological support to some activities of Methontology.

However, Methontology does not propose a set of design principles that guide the development of taxonomic knowledge and methods to clean tangled taxonomies. Therefore, given that OntoClean allows cleaning wrong *subclass of* links in taxonomies using notions like *rigidity*, *identity* and *unity*, it is an appropriate complement to be used for building taxonomies at the conceptualisation activity in Methontology. As a consequence, we integrated OntoClean method in Methontology, as we presented in [11].

Once the unification at the methodological level was performed, we were in the appropriate situation to build the software that gives support to OntoClean in WebODE. We call this software ODEClean. The inclusion of the ODEClean module would allow the use of the OntoClean method in an efficient way. Indeed, until now, OntoClean is being used in several industrial and academic settings to evaluate taxonomies [19]. However it is usually applied by hand.

Our solution also fits with the idea presented in [13], since we have not built an isolated tool for OntoClean, but a module integrated in an ontological engineering workbench.

The base of ODEClean is Guarino and their colleagues' top-level level ontology of universals [20], whose instances are concepts (in contrast with the top-level of particulars, whose instances are individuals). We have implemented the top level ontology of universals in WebODE. We enriched this ontology including the **meta-properties** (rigidity, identity, unity) and the evaluation rules proposed by OntoClean method. Then, the ontology was completly translated automatically using WebODE translators into Ciao Prolog. Thus, ODEClean consults the enriched top-level of universals and the axioms in Prolog every time that it has to evaluate a given domain ontology. That is, **the main advantage of the ODEClean module is that the knowledge used to evaluate ontologies is declaratively expressed through an ontology inside our ODEClean module**. **The user could relax or to stress the evaluation just clicking on more or less meta-properties**

Section 2 will present OntoClean, section 3 will present the top-level ontology of universals, section 4 will show ODEClean's ontology (enriched top-level of universals), section 5 will present WebODE, section 6 will show the ODEClean plug-in, its functions and how we have developed it, and, finally, section 7 will be devoted to conclusions and future lines.

2 OntoClean method

OntoClean has been elaborated by the Ontology Group of the LADSEB-CNR in Padova (Italy). It is a method to clean taxonomies according to notions such as: *rigidity*, *identity* and *unity*. Let us see these notions [17]:

- *Rigidity*. This notion is defined based on the idea of essence. A property is essential to an individual if and only if necessarily holds for that individual. Thus, a property is *rigid* (+R) if and only if is necessarily essential to all its instances. A property is *non-rigid* (-R) if and only if it is not essential to some of its instances, and *anti-rigid* (~R) if and only if it is not essential to all its instances. For example, the concept `person` is usually considered rigid, since every person is essentially such, while the concept `student` is not normally considered anti-rigid, since every student can possibly be a non-student a few years later.

- *Identity*. A property *carries an identity criterion* (IC) (+I) if and only if all its instances can be (re)identified by means of a suitable "sameness" relation. A property *supplies an identity criterion* (+O) if and only if such criterion is not inherited by any subsuming property. For example, *person* is usually considered a supplier of an identity criterion (for example the fingerprint), while `student` just inherits the identity criterion of `person`, without supplying any further identity criteria.
- *Dependency*. An individual *x* is constantly dependent on *y* if and only if, at any time, *x* cannot be present unless *y* is fully present, and *y* is not part of *x*. For example, a hole in a wall is constantly dependent on the wall. The hole cannot be present if the wall is not present. A property *P* is constantly dependent if and only if, for all its instances, there exists something they are constantly dependent on. For instance, the concept `hole` is constantly dependent because every instance of `hole` is constantly dependent.
- *Unity*. We can say that an individual is a *whole* if and only if it is made by a set of parts unified by a relation *R*. For example, the enterprise Iberia is a whole because it is composed by a set of people that are linked by the relation `having the same president`. A property *P* is said to *carry unity* (+U) if there is a *common* unifying relation R such that all the instances of *P* are wholes under *R*[1]. For example, the concept `enterprise-with-president` carries unity because every enterprise with president is made up people linked through the relation `having the same president`. A property carries *anti-unity* (~U) if all its instances can possibly be non-wholes. Properties that refer to amounts of matter, like *gold, water,* etc., are good examples of anti-unity.

Note that the definition of these notions refer to properties of properties. For example, *rigid* is a property that can take different values in different properties (*yes* in `person`, *no* in `student`, etc.). Another example is *carries an identity criterion*, since it can also take different values in different properties (*yes* in `person`, *no* in `student`, etc.). These properties of properties are called **meta-properties**, and to indicate their values, special symbols are used. For example, +R means that the meta-property *rigid* has the value *yes*. The meta-properties are useful to detect wrong *subclass of* relations. For example, `person` cannot be subclass of `student` because the former one is rigid and the later one not. In fact, if we had this link, what would it happen if a person was not student any more?

According to LADSEB-CNR's proposal, the specific steps to clean the wrong *subclass of* links in a taxonomy are (based on [26] and interviews with LADSEB-CNR's group):

1) *Put tags to every property assigning meta-properties*. This eases the analysis, because all the meta-properties are simultaneously visible.

[1] In the actual definition, the authors use *essential wholes* instead of *wholes*. We will sometimes sacrifice the accuracy to make clear the ideas of this paper to people still non very familiarised with Formal Ontology.

2) *Focus just on the rigid properties*. A taxonomy without rigid properties is called *backbone taxonomy*. It is the base of the rest of the taxonomy, that is, the essential part.
3) *Evaluate the taxonomy taking into account principles based on the meta-properties*. For instance, a rule suggested in OntoClean is “a property carrying anti-unity has to be disjoint of a property carrying unity”. As a consequence, “a property carrying unity cannot be a subclass of a property carrying anti-unity”. Therefore, `bronze statue` (it carries unity) cannot be a subclass of `bronze` (it carries anti-unity), for example.
4) *Consider non-rigid properties*. When the backbone taxonomy has been examined, the modeller has to evaluate the non-rigid properties. One of the proposed rules is: “a rigid property and an anti-rigid property are ever disjoint”. As a consequence, “a non anti-rigid property cannot be a subclass of an anti-rigid property”. Therefore, `person` (rigid) cannot be a subclass of `student` (anti-rigid).
5) *Complete the taxonomy with other concepts and relations.* There can be several reasons to introduce new concepts. One of them is the transformation of concepts in relations, for example, `student` could be transformed into a relation between `person` and `university`.

OntoClean has been used by IBM, OntologyWorks[2], Document Development Corporation[3]. At the Italian National Research Council Laboratories (LADSEB-CNR and ITBM-CNR), in Padova and Rome, OntoClean is in use in several projects including the development of an upper-level ontology based on a restructuring of WordNet, and the development of a core ontology for financial knowledge interchange [19].

3 Top level Ontoloy of Universals

The LADSEB-CNR’s Ontology Group (in Italy) has built two top-level ontologies, as presented in figure 1: one of universals, and another of particulars. Universals are concepts like `car` or `computer`, etc. and individuals are instances of these concepts, like `my car` or `my computer`, etc. Thus, for example, the particular `my car` is an instance of the universal `car`.

Top Level Ontology of Universals (TPU) is made up by meta-concepts like `type` or `role`, for example (see figure 2) [20]. The instances of such meta-concepts are concepts (universals). Concerning Top Level Ontology of Particulars (and every domain ontology) it is made up by concepts (universals) whose instances are particulars. That is, the tag "universals" or "particulars" associated to the names of the two CNR's ontologies are given by the kind of instances that they can contain.

[2] www.ontologyworks.com
[3] www.docdev.com

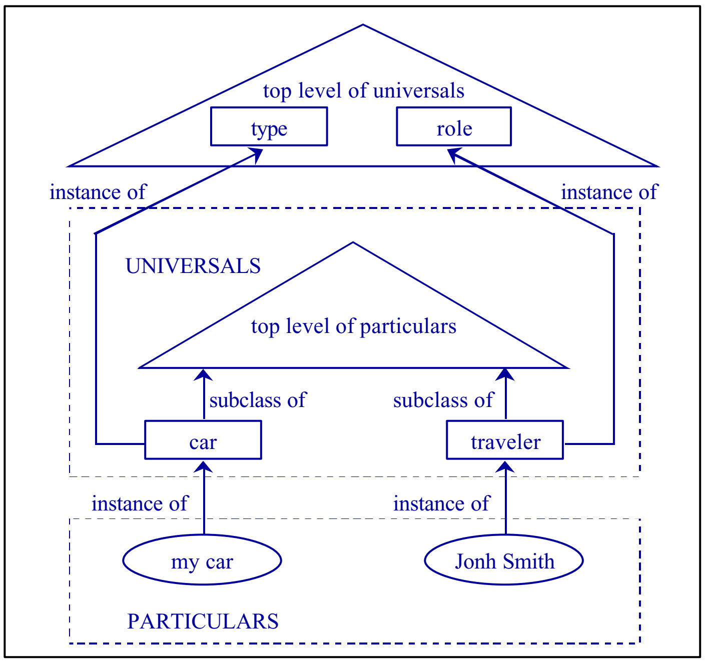


Fig. 1. Relationship between particulars and universals

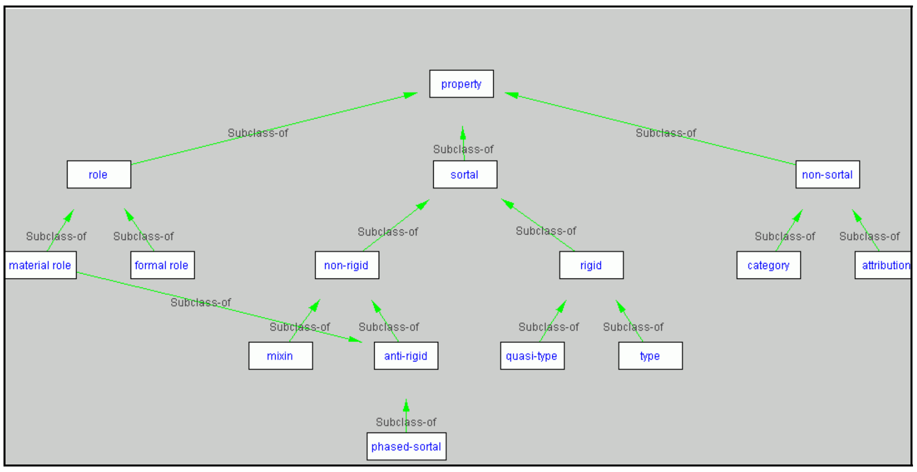


Fig. 2. Class taxonomy of the top-level ontology of universals

4 ODEClean's ontology

LADSEB-CNR's group continues its research in defining well-defined criteria for cleaning taxonomies, therefore, the proposed axioms can be modified and extended.

That is, every tool that implements OntoClean should be flexible. Consequently, we have taken a declarative approach to implement the knowledge used to clean taxonomies in ODEClean. Moreover, the representation of OntoClean rules to clean taxonomies also requires the representation of knowledge about meta-properties (*rigid*, *carries an identity criterion*, etc.). Because of this, ODEClean uses the top-level ontology of universals [19] enriched with LADSEB-CNR's meta-properties [17] and evaluation axioms [26].

To build ODEClean's ontology in WebODE, we mixed the following components:

1) *The top level of universals*. We introduced the taxonomy that appears in figure 2, which was obtained from [20].
2) *Meta-properties*. They were introduced as instance attributes of the root of TPU (`property`) according to WebODE knowledge model. Figure 3 shows the meta-concept `property` and its attributes.
3) *OntoClean axioms*. The OntoClean axioms to evaluate ontologies that appear in [26] were also included in TPU using the WebODE WAB module. Figure 4 shows the axiom that says : "a non anti-rigid property cannot be a subclass of an anti-rigid property".

During its working, ODEClean automatically links every concept inserted in the ontology into the root of its ontology through the relation *instance of*, as we can see in figure 3. Consequently, the TPU ontology meta-properties will be meta-attributes (class attributes) of every concept of the ontology to be cleaned. Hence, the user can assign values to the meta-properties in every concept of the ontology that (s)he is building.

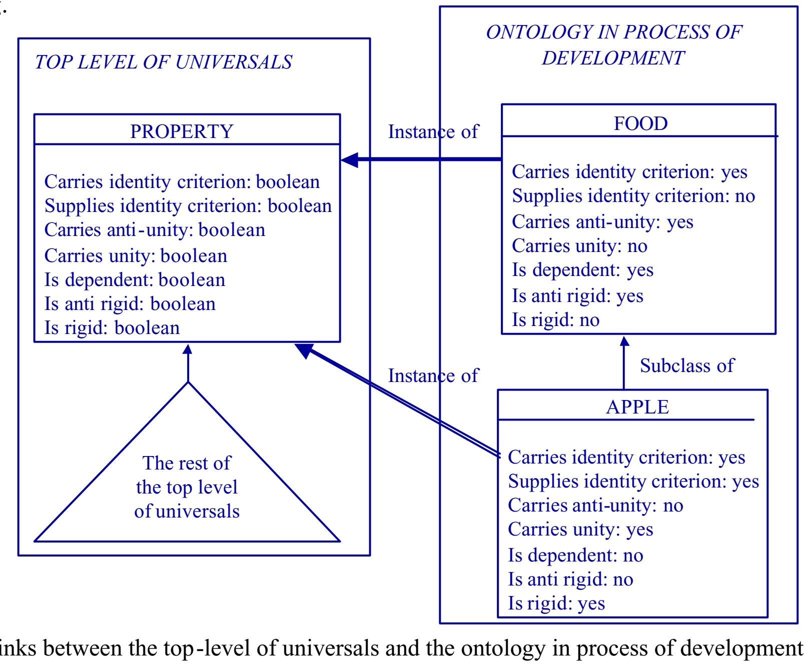
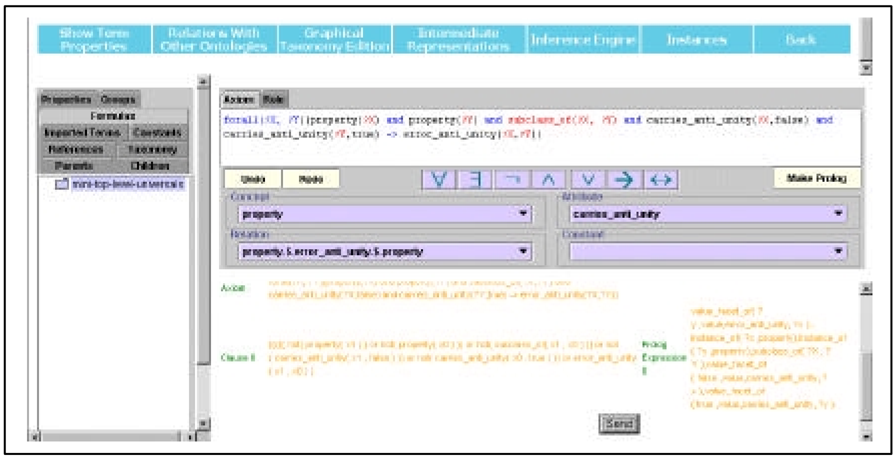


Fig. 3. Links between the top-level of universals and the ontology in process of development


Show Term Properties
Relations With Other Ontologies
Graphical Taxonomy Edition
Intermediate Representations
Inference Engine
Instances
Back
Axiom
Rule
Undo
Redo
Make Prolog
Send

```
forall(?X, ?Y)(property(?X) and property(?Y) and subclass_of(?X, ?Y) and carries_anti_unity(?X,false) and
carries_anti_unity(?Y,true) -> error_anti_unity(?X,?Y))
```

Concept: property
Attribute: carries_anti_unity
Relation: property.$.error_anti_unity.$.property

Fig. 4. OntoClean axiom in WebODE

In the current version of ODEClean the complete TPU hierarchy is not necessary, since OntoClean meta-properties are defined in a single meta-concept. However, the complete TPU hierarchy will be very useful. On the one hand, the values that the meta-properties take in the domain ontologies could be used to automatically classify the domain ontology concepts as instances of the meta-concepts of the TPU ontology (`role`, `type`, etc.). In fact, each TPU meta-concept has values associated to different meta-properties. For example, every role is anti-rigid, dependent, etc. On the other hand, TPU is already a part of OntoClean [26]. When the ontologist has to assign meta-property values to a domain concept, (s)he can take into account if that domain concept (for example, `food`) is a role, a type, etc. Indeed, some meta-properties values in the domain concepts could be deduced from the links between the domain ontology and TPU.

Nowadays, the problem to use TPU as a part of OntoClean is to know which meta-concept is each domain ontology concept instance of. Even more, depending on the point of view adopted by the modeller, the same concept can be, for example, a role or a type. In any case, ODEClean already includes the different meta-property values through all its ontology. Thus, for example, the meta-property *anti-rigid* takes the value *yes* in the meta-concept *role*. In this way, ODEClean is already prepared to help, in the future, in meta-property value inference.

5 WebODE

WebODE is a scalable, integrated workbench for ontological engineering that eases the representation of ontologies, the reasoning with ontologies and the exchange of ontologies with other ontology tools and ontology-based applications [8]. It has been developed by the Ontology Group of the Technical University of Madrid. The WebODE's knowledge model [6] is based on the intermediate representations proposed in Methontology [10]. Hence, it allows modelling concepts and their attributes (both class and instance attributes), taxonomies of concepts, disjoint and exhaustive class partitions, ad-hoc binary relations between concepts, properties of relations, constants, axioms and instances of concepts and relations.

WebODE is built according to a four-tier architecture: client, presentation, business logic, and database tiers. In all these tiers, we have used standard technology. The client tier uses HTML, XML, CSS, JavaScript and Java applets. The presentation tier uses servlets and JSPs. The business logic tier uses Java and RMI-IIOP. Finally, the database tier uses JDBC and Oracle. The main WebODE services are:

- *The WebODE Ontology Editor*. It allows the collaborative construction of ontologies at the knowledge level. It provides a default form-based web user interface to create ontologies according to the knowledge model aforementioned. The WebODE Ontology Editor also includes *OntoDesigner*, a visual tool that aids in the construction of concept taxonomies and ad-hoc relations between concepts.
- *WebODE Axiom Builder (WAB)*. WAB is an axiom and rule editor that is integrated in the WebODE Ontology Editor. It allows creating first order logic axioms and rules using a graphical user interface. It also provides a library of built-in axioms, which can be reused for creating other axioms, rather than building them from scratch.
- *WebODE's inference engine service*. WebODE includes an OKBC-based inference engine. This inference engine reasons with a subset of the OKBC protocol's primitives [7].
- *WebODE interoperability services*. Ontologies built with WebODE can be easily integrated in other ontology servers or used in ontology-based applications. Possible choices for interoperability include WebODE's ontology access API, which can be accessed by other applications using RMI, and is completely compliant with the WebODE's knowledge model. Currently, WebODE is able to export to and import ontologies from: RDF(S), OIL, DAML + OIL, the XMLization of CARIN and FLogic. It also can export to JESS and Prolog.
- *WebPicker* [9] is a set of wrappers that allow importing standards of classification of products and services in the context of electronic commerce into WebODE (UNSPSC, e-cl@ss and RosettaNet). We are currently extending it to wrap other sources of information, such as Cyc.
- *ODECatalogue* is able to generate electronic catalogs from ontologies according to some parameters. The catalogue generation from an ontology assures a correct and rich classification of the different products.
- *ODEMerge* performs a supervised merge of concepts, attributes and relationships from two different ontologies built for the same domain, according to semantic criteria and resources used for natural language processing.
- *ODEClean* plug-in, which will be presented in this paper.

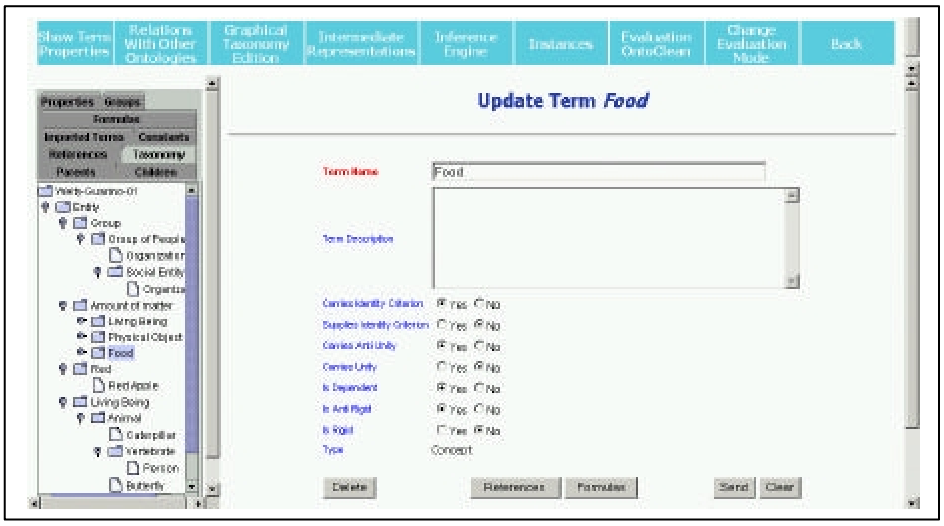
WebODE has been successfully used, with different domains and purposes and by different groups of people, in the following projects: The European IST project MKBEEM (IST 1999-10589), the OntoWeb thematic network (IST-2000-29243), the Spanish CICYT project ContentWeb (TIC-2001-2745), the Spanish CICYT project on Methodology for Knowledge Management (TIC-980741), etc.

6 The ODEClean plug-in of WebODE

To present the plug-in, first of all, we show its functions (section 6.1), and then, we will describe how ODEClean module has been built (section 6.2). In section 6.2 we will not describe the integration process of OntoClean in METHONTOLOGY because it was presented at [11].

6.1 Functions of the ODEClean plug-in of WebODE

The purpose of ODEClean is to allow developers to evaluate taxonomies using OntoClean method. ODEClean is a plug-in of WebODE and WebODE was designed taking into account the METHONTOLOGY methodology. When the ontologists build an ontology in WebODE, it is possible for him to select wheather he wants to build the taxonomy taking into account the OntoClean principles. It is also possible to pick up an ontology from WebODE ontology library and to clean its taxonomy just assigning values of the meta-properties of each concepts. One way to assign meta-properties to the concepts is through the form-based web user interface of WebODE (see figure 5). The other way to assign meta-properties is through the visual tool *OntoDesigner* (see figure 6). This last way allows the developer to tag the concepts of the ontology like if (s)he was designing the taxonomy in a blackboard.



Update Term *Food*

Term Name	Food
Term Description	
Carries Identity Criterion	[✓ Yes] No
Supplies Identity Criterion	Yes [✓ No]
Carries Anti Unity	[✓ Yes] No
Carries Unity	Yes [✓ No]
Is Dependent	[✓ Yes] No
Is Anti Rigid	[✓ Yes] No
Is Rigid	Yes [✓ No]
Type	Concept

Fig. 5. Form-based web for ODEClean

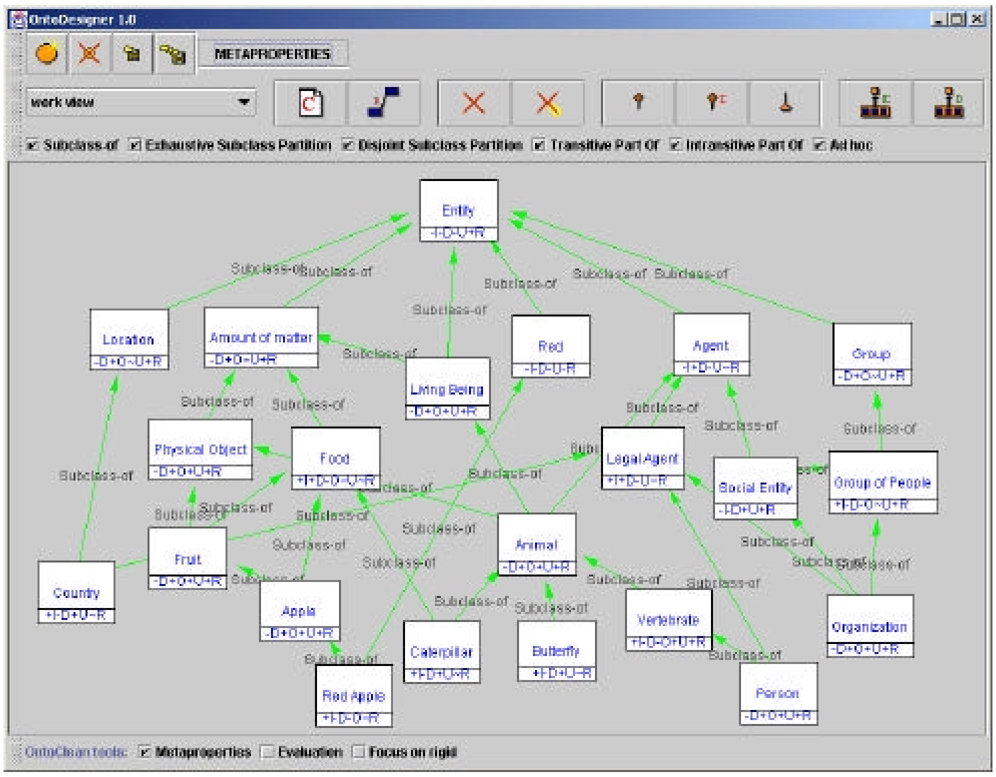


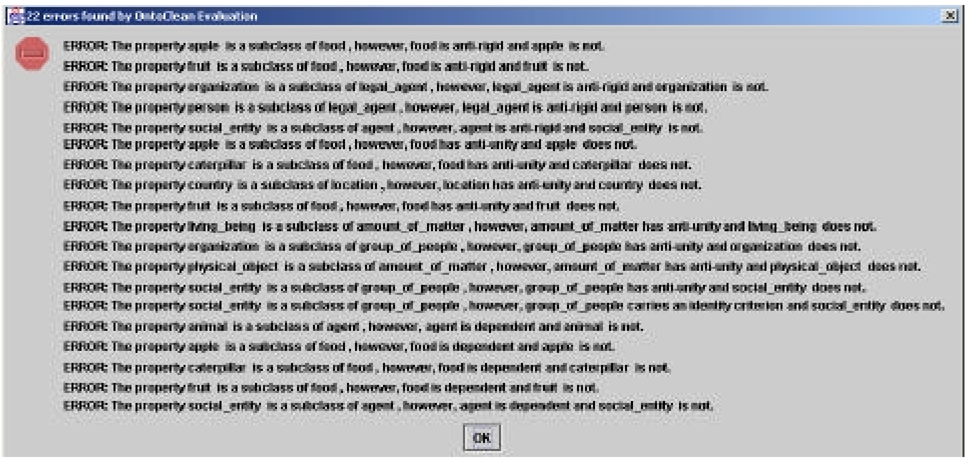
Fig. 6. OntoDesigner for evaluating taxonomies following OntoClean (taxonomy taken from [26], where the authors use it to show how to evaluate ontologies with OntoClean)

The main functions provided by ODEClean are:

1. *Establishing the evaluation mode*. The user can choose whether the system has to show the errors every time that it detects a problem in the domain ontology, or the system only has to show the errors when the user ask for them. This option is available in the button *Change Evaluation Mode* of the form-based web (see top figure 5), whereas it is available in the signal *Evaluation* (figure 6) of OntoDesigner.
2. *Assigning meta-properties to concepts*. The user will be able to set up meta-properties concerning identity, unity, dependency and rigidity. If the form-based web is used, then a change in the value of a meta-property can provoke an automatic change in the value of other meta-property. For example, if you click in *yes* in *supplies an identity criterion*, then the value of *carries an identity criterion* is automatically established as *yes*. On the other hand, the assignment of values to the meta-properties using the *OntoDesigner* is performed tagging each concept with the OntoClean classical symbols introduced in section 2 (~*R*+*I*-*O*, etc.). A user that does not wish to see the meta-properties with OntoDesigner can

hide them clicking in *Metaproperties*.

3. *Focusing on rigid properties*. The user can decide whether to show or not the non-rigid properties. As you can see in section 2, one of the step of OntoClean is to focus on rigid properties.
4. *Evaluation according to the taxonomic constraints*. If the user order to evaluate the ontology, then the found errors are shown. Each error message describes the violation of a OntoClean axiom (see [26]) in a link *subclass of* between two concepts. The first error that appears in figure 7, for example, shows that the concept `food` is anti-rigid whereas `apple` (a subclass of `food`) does not. This is a violation of OntoClean axioms.



```
22 errors found by OntoClean Evaluation

ERROR: The property apple  is a subclass of food , however, food is anti-rigid and apple  is not.
ERROR: The property fruit  is a subclass of food , however, food is anti-rigid and fruit  is not.
ERROR: The property organization  is a subclass of legal_agent , however, legal_agent is anti-rigid and organization  is not.
ERROR: The property person  is a subclass of legal_agent , however, legal_agent is anti-rigid and person  is not.
ERROR: The property social_entity  is a subclass of agent , however, agent is anti-rigid and social_entity  is not.
ERROR: The property apple  is a subclass of food , however, food has anti-unity and apple  does not.
ERROR: The property caterpillar  is a subclass of food , however, food has anti-unity and caterpillar  does not.
ERROR: The property country  is a subclass of location , however, location has anti-unity and country  does not.
ERROR: The property fruit  is a subclass of food , however, food has anti-unity and fruit  does not.
ERROR: The property living_being  is a subclass of amount_of_matter , however, amount_of_matter has anti-unity and living_being  does not.
ERROR: The property organization  is a subclass of group_of_people , however, group_of_people has anti-unity and organization  does not.
ERROR: The property physical_object  is a subclass of amount_of_matter , however, amount_of_matter has anti-unity and physical_object  does not.
ERROR: The property social_entity  is a subclass of group_of_people , however, group_of_people has anti-unity and social_entity  does not.
ERROR: The property social_entity  is a subclass of group_of_people , however, group_of_people carries an identity criterion and social_entity  does not.
ERROR: The property animal  is a subclass of agent , however, agent is dependent and animal  is not.
ERROR: The property apple  is a subclass of food , however, food is dependent and apple  is not.
ERROR: The property caterpillar  is a subclass of food , however, food is dependent and caterpillar  is not.
ERROR: The property fruit  is a subclass of food , however, food is dependent and fruit  is not.
ERROR: The property social_entity  is a subclass of agent , however, agent is dependent and social_entity  is not.
```

Fig. 7. Errors detected by ODEClean

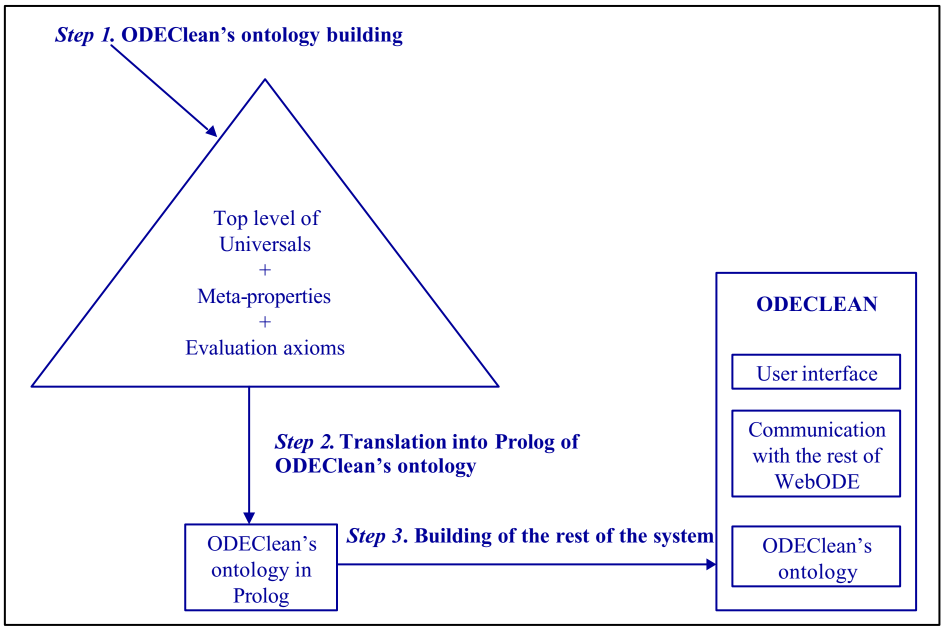
6.2 How we have built ODEClean

To develop ODEClean, we firstly built TPU using the WebODE Ontology Editor. We enriched it with the necessary meta-properties for OntoClean. Then, using WAB, we added the LADSEB-CNR's rules into the top level of universals. Then, we translated this ODEClean's ontology into Prolog using the WAB service of WebODE. Such Prolog ontology is the base of our system.

Thus, the particular steps that we have carried out to develop ODEClean are (see figure 8):

1. *ODEClean's ontology building*. As we have said in section 4, we made an ontology that contains OntoClean knowledge, useful for taxonomy cleaning.
2. *Translating into Prolog of ODEClean's ontology*. The purpose of this step was to generate a code with inference engine available. We used the WebODE translator that generates Prolog. WebODE translator into Prolog uses OKBC primitives. The use of OKBC primitives could ease the interaction with other systems.

3. *Building the rest of the system.* Taking the Prolog ontology, we built the rest of the modules of ODEClean: the user interface and the communication with the rest of WebODE.



Step 1. ODEClean's ontology building
Top level of Universals + Meta-properties + Evaluation axioms
Step 2. Translation into Prolog of ODEClean's ontology
ODEClean's ontology in Prolog
Step 3. Building of the rest of the system
ODECLEAN
User interface
Communication with the rest of WebODE
ODEClean's ontology

Fig. 8. The development of ODEClean

Concerning the internal behaviour of the system, WebODE's inference engine makes use of Ciao Prolog [21]), as a consequence, the inference engine that applies the OntoClean rules uses Ciao Prolog.

7 Conclusions

In this paper we have presented the plug-in of WebODE that implements OntoClean, the method to clean ontologies elaborated in the LADSEB-CNR of Padua (Italy). WebODE is the ontology development platform developed by the Ontology Group of the Technical University of Madrid. This plug-in allows the developer to assign meta-properties to concepts, focus on non-rigid properties, automatically check errors, etc. The user can visualise the ontolology either through a form-based web user interface or graphically with OntoDesigner.

This plug-in is not only the product of software development, but also a work at the ontology development methodological level. That is, first of all, we integrated OntoClean in METHONTOLOGY. Then, we made the ODEClean plug-in integrated in WebODE, the METHONTOLOGY software support.

The plug-in has been built using as base LADSEB-CNR's top-level ontology of universals translated to Prolog. We have used the WebODE WAB plug-in to add it the OntoClean evaluation rules before translating it to Prolog.

The main contributions of our work are:

1. The new module is a consequence of the integration of an evaluation method in a development methodology. That is, we have carried out an integration at the methodological level before performing it at the software level.
2. We have built the first tool integrated in a ontology development platform that supports the method OntoClean.
3. An ontology built by a group that has not participated in the development of WebODE has been introduced in WebODE. Moreover, the ontology enriched with meta-properties and axioms coded in Prolog is thought to be reusable in other platforms or tools different to WebODE.

   Kalfoglou and colleagues' evaluation of applications is also based on the use of ontologies. However, their approach is more focussed on the use of an ontology as the formal a specification of the application that they are going to evaluate.
4. **The knowledge used to evaluate ontologies is declaratively specified**,. which means that:
   - New meta-properties could be added easily, just introducing new attributes in ODEClean's ontology.
   - New axioms could be added or modified using WAB.

According to our experience developing this plug-in, if the future evaluation tools are declaratively developed, they will be flexible.

Acknowledgements

This work is supported by the project "ContentWeb: Plataforma tecnológica para la web semántica: ontologías, lenguaje natural y comercio electrónico[4]" (TIC-2001-2745), and by the project "Esperonto Services" (IST-2001-34373). This work would not have been possible without the help of Emilio Raya.

References

1. T. Berners-Lee, J. Hendler and O. Lassila. A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 2002, cf. http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html.
2. EU IST-1999-10132 project "On-To-Knowledge: Content-driven knowledge management tools through evolving ontologies", cf. http://www.ontoknowledge.org.
3. US DARPA project "DARPA Agent Markup Language (DAML)", cf. http://www.daml.org.
4. EU IST-2000-29243 thematic network "OntoWeb: Ontology-based Information Exchange for Knowledge Management and Electronic Commerce", cf. http://www.ontoweb.org.

[4] ContentWeb: Platform for the Semantic Web: ontologies, natural language and e-commerce

5. A survey on ontology tools. Deliverable D13. IST OntoWeb Thematic Network. May 2002.
6. Arpírez, J.C.; Corcho, O.; Fernández-López, M.; Gómez-Pérez, A. WebODE: a scalable ontological engineering workbench. First International Conference on Knowledge Capture (K-CAP 2001). Victoria, Canada. October, 2001.
7. Chaudhri V. K.; Farquhar A.; Fikes R.; Karp P. D.; Rice J. P. The Generic Frame Protocol 2.0. Technical Report, Stanford University.1997.
8. Corcho, O., Fernández-López, M., Gómez-Pérez, A., Vicente, O. WebODE: an integrated workbench for ontology representation, reasoning and exchange. 13th International Conference on Knowledge Engineering and Knowledge Management EKAW02. 2002.
9. Corcho, O; Gómez-Pérez, A. WebPicker: Knowledge Extraction from Web Resources. 6th Intl. Workshop on Applications of Natural Language for Information Systems (NLDB'01). Madrid. June, 2001.
10. Fernández-López, M.; Gómez-Pérez, A. "Overview and analysis of methodologies for building ontologies". *Knowledge Engineering Representation* (to be published).
11. Fernández-López, M.; Gómez-Pérez, A.; Guarino, N. 2001. "The Methontology & OntoClean merge". *Technical Report, OntoWeb special interest group on Enterprise-standards Ontology Environments*. Amsterdam. 2001.
12. Fernández-López, M.; Gómez-Pérez, A.; Pazos, J.; Pazos, A. Building a Chemical Ontology using methontology and the Ontology Design Environment. IEEE Intelligent Systems and their applications. #4 (1):37-45. 1999.
13. Gómez-Pérez, A. *A proposal of infrastructural needs on the framework of the semantic web for ontology construction and use*. FP6 Programme Consultation Meeting 9. April 27th, 2001.
14. Gómez-Pérez, A. *Evaluation of Ontologies*. International Journal of Intelligent Systems. 16(3). March, 2001.
15. Gómez-Pérez, A. 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. 1994.
16. Gómez-Pérez, A. From Knowledge Based Systems to Knowledge Sharing Technology: Evaluation and Assessment. Technical Report. KSL-94-73. Knowledge Systems Laboratory. Stanford University. December 1994.
17. Gangemi, A., Guarino, N., Masolo, C., and Oltramari, A. 2001. Understanding top-level ontological distinctions. Proc. of IJCAI 2001 workshop on Ontologies and Information Sharing .
18. Grüninger, M.; Fox, M. S. 1995. "Methodology for the design and evaluation of ontologies." *Workshop on Basic Ontological Issues in Knowledge Sharing*. Montreal (Canada).
19. Guarino, N. and Welty, C. 2002. "Evaluating Ontological Decisions with OntoClean". *Communications of the ACM*, 45(2): 61-65.
20. Guarino, N. and 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. Springer Verlag: 97-112.
21. Hermenegildo, M., Bueno, F., Cabeza, D., Carro, M., García, M., López, P., Puebla, G. *The Ciao Logic Programming Environment*. International Conference on Computational Logic (CL2000). July, 2000.
22. Y.Kalfoglou, D.Robertson. "Managing Ontological Constraints", In Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, August 1999.
23. Y.Kalfoglou, D.Robertson,"Use of Formal Ontologies to Support Error Checking in Specifications" In Proceedings of the 11th European Workshop on Knowledge Acquisition, Modelling and Management (EKAW99), Dagsthul, Germany, May 1999.

24. Staab, S.; Schnurr, H.-P.; Studer, R.; Sure; Y. “Knowledge Processes and Ontologies”, *IEEE Intelligent Systems, 16(1)*, January/February 2001.
25. Uschold, M. King, M. 1995. “Towards a Methodology for Building Ontologies”. *Workshop held in conjunction with IJCAI on Basic Ontological Issues in Knowledge Sharing*.
26. Welty, C.; Guarino, N. *Supporting Ontological Analysis of Taxonomic Relationships*. Data and Knowledge Engineering. September 2001.

Ontology Evolution within Ontology Editors

L. Stojanovic, B. Motik

FZI - Research Center for Information Technology at the University of Karlsruhe,
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
{Ljiljana.Stojanovic, Boris.Motik}@fzi.de

Abstract. An ontology over a period of time needs to be modified to reflect changes in the real world, changes in the user's requirements, drawbacks in the initial design, to incorporate additional functionality or to allow for incremental improvement. Although changes are inevitable during the development and deployment of an ontology, most of the current ontology editors unfortunately do not provide enough support for efficient copying with changes. Since changes are the force that drives the evolution process, in this paper we discuss the requirements for the ontology editors in order to support ontology evolution.

1 Introduction

Ontologies aim at capturing domain knowledge in a generic way, and provide a commonly agreed understanding of a domain, which may be reused and shared across applications and groups. Although there are several approaches for a semi-automatic ontology development ([6], [8]), most of the existing ontologies are created manually using ontology editors.

Ontology editors are tools that enable inspecting, browsing, codifying, and modifying ontologies and support in this way the ontology development and maintenance task [11]. Existing editors vary in the complexity of the underlying knowledge model, usability, scalability, etc. Nevertheless, all of them provide enough support for the initial ontology development. However, ontology development is necessarily an iterative and a dynamic process [1]. Very seldom is an ontology perfect the first time it is made, and then continues, without change, to be as useful over time as it was when it was first deployed. The reasons for changes are inherent in the complexity of reality and in the limited ability of humans to cope with this complexity. Thus, ontologies must be able to evolve for a number of reasons, including the following:

- Ontologies often contain "design error" and sometimes do not immediately meet the requirements of its users;
- The environment in which the ontology operates can change unpredictably, thereby invalidating the assumptions that were made when the ontology was built;
- Users' requirements can change after the ontology is initially built, requiring that the existing ontology evolve to meet the new requirements.

The necessity to support change management can be derived from many real-word applications, since they typically operate in changeable environments. A typical example is MEDLINE database containing over 11 million references to articles from 4,600 worldwide journals in life sciences. It is in the irregular operation in November and December as NLM makes the transition to a new year of Medical Subject Headings[1] (MeSH), somewhere called medical ontology. Another example is UNSPSC[2] classification of products currently consisting of the hierarchy of about 16.000 product categories. Every two weeks, a change is performed, which alters between 100 and 600 concepts. This causes serious problems for companies that use it to classify their product data to allow e-commerce [4].

Therefore, the methods and tools for the ontology evolution enabling coping with the changes in a more systematic way have become an essential requirement for an ontology-based application [1]. The ontology evolution [4] is the timely adaptation of the ontology as well as the consistent propagation of these changes, because a modification in one part of the ontology may generate subtle inconsistencies in other parts of the same ontology, in the instances, depending ontologies and applications.

The ontology evolution is becoming more important nowadays. The major reason for this is the increasing number of ontologies in use and the increasing costs associated with adapting them to changing requirements. Developing ontologies and their applications is expensive, but evolving them is even more expensive. However, even though evolution over time is an essential requirement for useful ontologies [11], appropriate tools and strategies for enabling and managing evolution are still missing. This level of ontology management is necessary not only for the initial development and maintenance of ontologies, but is essential during deployment, when scalability, availability, reliability and performance are absolutely critical [1].

Since an ontology is usually developed using an ontology editor, many requirements for the ontology evolution have to be part of the ontology editors. An ontology editor must provide an interface that allows the knowledge engineer to modify the underlying ontology. The interface is based on the set of available ontology changes. Moreover, there are many features which can significantly improve the usability of an ontology editor and enhance its functionality regarding the ontology evolution. In this paper, we discuss the most critical requirements for ontology editors in order to be more robust to a changing environment.

The paper is organised as follows: In the second section, we elaborate a set of requirements for an ontology editor to be able to support ontology evolution. The evaluation of the some ontology editors in terms of these requirements is given in section 3. Section 4 contains concluding remarks.

2 Requirements for the Ontology Evolution

Ontology development is a dynamic process [7] starting with an initial rough ontology, which is later revised, refined and filled in the details [5]. Consequently, an ontology almost certainly should be evolved[3] in order:

[1] MeSH is a controlled vocabulary thesaurus used to index the articles [http://www.nlm.nih.gov/pubs/factsheets/medline.html]
[2] http://eccma.org/unspsc/
[3] IEEE 1219 1993

- to fix “bugs” in the initial design (corrective maintenance);
- to adapt itself to the changes in the environment (adaptative maintenance);
- to improve itself after it has become operational (perfective maintenance);
- to avoid future changes and to alleviate maintenance (preventive maintenance).

Moreover, ontology evolution has to be supported through the entire lifecycle [11]. Since ontology editors are the main tools for ontology development, the support for evolution should be a required facility in an ontology editor. In other words, the functional specification of an ontology editor has to incorporate requirements for the ontology evolution. In this paper, we have identified a set of requirements for ontology editors to allow users to be able to alter an ontology in a more efficient and convenient manner. These requirements can be divided in several groups:

- Functional requirement specifies all evolution changes that must be supported;
- User’s supervision requirement enables the user-driven process of change resolving;
- Transparency requirement deals with providing control of the evolution process through an insight into the scope of an evolution operation before the operation is applied;
- Reversibility requirement states how the effect of evolution changes can be undone;
- Auditing requirement is related to the management of the ontology change history;
- Ontology refinement requirement provides support for continual ontology improvement;
- Usability requirement allows the user to manage changes more easily by finding ontology inconsistencies and providing the explanation to solve them.

In the rest of this section, we elaborate these requirements in more details.

2.1 Functional requirement

The functional requirement specifies which functionality must be provided for the ontology development and evolution. This functionality heavily depends on the underlying ontology model. The more powerful and expressive model requires a richer set of modelling primitives. Thus, before speaking about functional requirements, the notion of an ontology itself has to be clarified. Corresponding to the variety of ontology models in use[4], there is no standard ontology model. However, an attempt to provide the standard for ontology structure[5] is on the way.

Due to differences in ontology models, we concentrate on the “common” features of ontology models, namely concepts, properties, instances, as well as concept inheritance. Each of these ontology entities can be updated by one of the meta-change transformations: add, remove, modify [3]. A full set of changes (Tab. 1) can thus be defined by the cross product of the set of entities of the ontology model, which form meta schema, and the set of meta-changes.

[4] http://www.ontoknowledge.org/oil/, http://www.daml.org/2001/03/reference.html

[5] http://www.w3.org/2001/sw/WebOnt/

Table 1. Changes in the ontology

Meta changes **Meta enitities**	**Add**	**Remove**	**Modify**
Concept	Add concept	Remove concept	Rename concept
Concept hierarchy	Add subConceptOf relationship	Remove subConceptOf relationship	Set subConceptOf relationship
Property	Add property	Remove property	Rename property
Property Domain	Add property domain	Remove property domain	Set property domain
Property Range	Add property range	Remove property range	Set property range
Instance	Add instance	Remove instance	Rename instance
Property Instance	Add property instance	Remove property instance	Set property instance

The existence of the "modify" change causes the set of primitives not to be minimal with respect to completeness. However, this change adds some important semantic variations to the set of changes, since a modify change is not equivalent to a removal followed by an addition [3]. The difference is that the modification of an entity (i.e. renaming a concept) maintains its identity, while removing and adding loses its identity.

The previously mentioned changes are called elementary changes, since they cannot be decomposed into simpler changes. The basic functionality of each ontology editor from the ontology evolution point of view is specified as a set of elementary ontology changes derived from the corresponding ontology model.

Elementary changes in the ontology specify fine-grained changes that can be performed in the course of ontology evolution. However, this granularity of ontology evolution changes is not always appropriate. Often, the intent of the changes may be expressed on a higher level. Composite changes [9] specify coarse-grained changes that can be performed to improve the ontology structure according to some criteria. They are more powerful, since the designer does not need to go through every step of the sequence of basic changes to achieve the desired effect.

Moreover, composite changes often have more meaningful semantics. For example, the semantics of moving the concept from one parent concept to another is clearly different from the semantics of removal and addition of a subConceptOf relation. While "move" as a composite change maintains the identifiers of a subConceptOf relation and preserves all properties and instances, the removal and subsequent addition create a new identifier for a subConceptOf relation and cause the loss of much information (e.g. at the instance level).

All valid changes to manipulate an ontology can be specified by one elementary or composite change or by a sequence of changes. Changes can be applied to an ontology in a valid state, and after all changes are performed, the ontology and dependent artefacts must transition to another valid state. It means that every change is guaranteed to maintain some constraints. We have identified the following set of system integrities that have to be maintained during resolution in order to achieve the soundness:

- Consistency – A consistent ontology is one that satisfies all invariants of the ontology model. Invariants are constraints that must hold in every quiescent state of an ontology. For example, the concept hierarchy is a direct acyclic graph;
- Validity – We distinguish between syntax and semantic validity [9] of an ontology. Syntax invalidity arises when undefined entities are used or model constraints are invalidated. Semantic invalidity arises when the meaning of an ontology entity is modified. On the other hand, a valid instance is one that conforms to the constraint specified in an ontology;
- Well-formedness - A well-formed ontology and instances are those, which syntactically conform to the language specification.

2.2 User's supervision requirement

The ontology evolution is a process of changing an ontology while maintaining its consistency. The goal of the ontology evolution is thus to evolve an ontology from one consistent stage to the next. However, there are many ways to achieve consistency after a change request. For example, when a concept from the middle of the hierarchy is being deleted, all subconcepts may either be deleted or reconnected to other concepts [9]. If subconcepts are preserved, then properties of the deleted concept may be propagated, its instances distributed, etc. Thus, for each change in the ontology, it is possible to generate different sets of additional changes, leading to different final consistent states.

Hence, a mechanism is required for users to manage changes resulting not in an arbitrary consistent state, but in a consistent state fulfilling the user's preferences. In order to enable the user to obtain the ontology most suitable to her needs, an ontology editor should allow the customisation of the ontology evolution process. One mean is to enable the user to set up one of evolution strategies that are used for resolving the changes.

An evolution strategy unambiguously defines the way how elementary and composite changes will be resolved. Typically, a particular evolution strategy is chosen by the user at the start of the ontology evolution process. Thus, an evolution strategy defining a common policy must be chosen to specify how to handle each of the following situations:

- how to handle orphaned concepts - those concepts that don't have parents any more;
- how to handle orphaned properties - those properties that don't have parents any more;
- how to propagate properties to the concept whose parent changes;
- what constitutes a valid domain of a property;
- what constitutes a valid range of a property;
- whether a domain (range) of a property can contain a concept that is at the same; time a subconcept of some other domain (range) concept;
- the allowed shape of the concept hierarchy;
- the allowed shape of the property hierarchy;
- must instances be consistent with the ontology.

For each of these situations, there is a set of possible options, e.g. in case of the first issue, orphaned subconcepts of a concept may be connected to the parent concept(s) of that concept, connected to the root concept of the hierarchy or deleted as well.

2.3 Transparency requirement

A change in one part of an ontology may have far reaching consequences on other parts of the ontology and associated instances. If an ontology is large, it may be difficult to fully comprehend the extent and meaning of each change. To improve understanding of effects of each change, the ontology evolution should provide maximum transparency into details of each change being performed. Transparency should provide a human-computer interaction for evolution by presenting change information in an orderly way, allowing easy spotting of potential problems and alleviating the understanding of the scope of the change.

Before any change is applied to the ontology, a list of all implications must be generated and reported to the user. The ontology engineer should be able to comprehend the list, and approve or cancel the change. If the changes are cancelled, the ontology should remain intact. The presentation of changes has to follow the progressive disclosure principle: related changes have to be grouped together and organised in a tree-like form. The user can initially see only the general description of changes. If she is interested in details, she can expand the tree and view complete information. She may cancel the operation before it is actually performed.

2.4 Reversibility requirement

As mentioned, the transparency requirement was introduced to help the ontology engineers comprehend the effect of a change. If properly done, this can help in reducing the number of accidental ontology changes, and can even guide the ontology refinement process. However, there are numerous circumstances where it may be desired to reverse the effects of changes. The reversibility requirement states that an ontology editor has to allow undoing changes at the user’s request. Consequently, the user can control changes and make appropriate decisions.

It is important to note that reversibility means undoing all effects of a change, which may not be the same as simply requesting an inverse change manually. For example, if a concept is removed from a concept hierarchy, its subconcepts will be modified (e.g. attached to the root). Reversing such change is not equal to recreating the deleted concept – one also needs to revert the concept hierarchy into an original state.

To support the undo-redo in a usable fashion, undoing an action must be accompanied by restoring the state of the UI to what it was before the action was performed. For example, if a concept in the concept-hierarchy tree was selected and then deleted, when the change is undone, the same concept must be selected. If the tree was scrolled in the meanwhile, the original scroll position of the tree must be restored (or at least the node must be made scrolled into view). For the navigation in an application, users often rely on the visual features of the application. When an operation is undone, it is essential to restore the previous visual state of the application as close as possible, allowing the users to quickly recognise a familiar state and proceed with their work. If the visual state of the application is not restored well, although the action is undone, the user may not realise this, and may mistakenly request another undo operation.

2.5 Auditing requirement

As business applications of ontologies proliferate, so do the needs for auditing ontology evolution. Changes to business information are often accompanied with responsibility for their effects on the business. Auditing is therefore a typical component of business systems, and must be reflected in the ontology evolution as well.

The ontology evolution auditing involves the following aspects:

- Keeping a detailed log of all performed changes allowing later reconstruction of the events that led to the current state of the ontology;
- Associating meta-information with each log change, such as textual change description, cost of change, time of change etc.;
- Tracking the identity of the change author.

The auditing requirement is also related to the reversibility requirement, since the auditing log is typically used to provide reversibility. The auditing log can also serve as a source for information mining about change trends.

2.6 Ontology refinement requirements

This requirement states that potential changes improving the ontology may be discovered semi-automatically from the ontology-based data and through the analysis of the user's behaviour. We distinguish (i) structure-driven, (ii) data-driven and (iii) usage-driven change discovery [10].

(i) The structure-driven change discovery identifies the following set of heuristics to improve an ontology based on the analysis of the structure of the ontology:

- If all subconcepts have the same property, the property may be moved to the parent concept;
- A concept with a single subconcept should be merged with its subconcept;
- If there are more than a dozen subconcepts for a concept, then an additional layer in the concept hierarchy may be necessary;
- The concept without properties is a candidate for deletion;
- If a direct parent of a concept can be achieved through a non-direct path, then the direct link should be deleted.

(ii) The data-driven change discovery states that some changes are implicit changes in the domain, reflected in its instances and can be discovered only through their analysis. We have found the following set of heuristics:

- A concept with no instances may probably be deleted;

- If no instance of a concept C uses any of the properties defined for C, but only properties inherited from the parent concept, we can make an assumption that C is not necessary;
- A concept with many instances is a candidate for being split into subconcepts and its instances distributed among newly generated concepts.

(iii) The usage–driven ontology evolution takes into account the usage of the ontology in the knowledge management system [10]. It is based on the analysis of the users' behaviour in two phases of a knowledge management cycle: in providing knowledge by analysing the quality of annotations, and in searching for knowledge by analysing the users' queries and the responses from the knowledge repository. For example, by tracking when the concept was last retrieved by a query, it may be possible to discover that some concepts are out of date and should be deleted or updated.

2.7 Usability requirement

An ontology editor addresses the issue of presenting ontologies and allowing the user to operate on ontologies in a consistent way. It also addresses how different functions are integrated into the system in a way natural to the user. An ontology editor has to have an interface that enables the user to create and maintain ontologies, one that is easily understood and allows the user to work efficiently with all the complexities inherent in an ontology editor.

However, the real usability of an ontology editor cannot be achieved only through the graphical means for the creation/modification of ontology entities which relieve the user of the necessity to perform this task manually. An ontology editor has to guide the user through the ontology development process by providing additional information, such as why the user's activity did not succeed, or what else she has to do in order to finish the current activity.

Moreover, a good ontology editor must provide capabilities for identifying inconsistencies. When such conflicts arise, an editor must assist the user in identifying the source of the problem and resolving it. Furthermore, the usability of an ontology editor can be significantly increased by incorporating validation. Validation concerns the truthfulness of an ontology with respect to its problem domain - does the ontology represent a piece of reality and the users' requirements correctly? One technique for supporting validation is generating explanation.

3. Evaluation

Ontology editors are tools that allow users to visually manipulate ontologies. The number of tools for building ontologies developed in the last years is high[6]. In this section, we evaluate three ontology editors which are most frequently used in the Semantic Web community, in terms of the requirements for the ontology evolution. Table 2 shows the result of comparison.

[6] http://www.ontoweb.org/download/deliverables/D13_v1-0.zip

Table 2. Evolution support within ontology editors. Description: "-" means that there is no support, "<>" states that support is partial and "+" corresponds to the full support.

Editors/ Requirements	**Protege**[7]	**OntoEdit**[8]	**OilEd**[9]
Functionality			
elementary	+	+	+
composite	-	<>	-
Supervision	-	-	-
Transparency	-	<>	-
Reversibility	<>	<>	-
Auditing	<>	-	<>
Refinement	-	-	-
Usability			
user-friendly	+	+	+
verification	<>	<>	<>
validation	-	-	-

The basic functionality of each ontology editor is specified as a set of elementary ontology changes. Thus, all editors allow such modifications. Even though composite changes allow an ontology engineer to update an ontology without having to find the right sequence of elementary modifications, most of the existing ontology editors do not include composite changes. Only OntoEdit provides support for some composite changes (move and copy).

Most of the existing systems for the ontology development provide only one possibility for realising a change, and this is usually the simplest one. For example, the deletion of a concept always causes the deletion of all its subconcepts. It means that users are not able to control the way changes are performed (supervision).

Moreover, users do not obtain explanations why a particular change is necessary (transparency). In OntoEdit, the user only obtains the information about numbers of induced changes, but without providing more details.

Furthermore, there is no possibility to undo effects of changes (reversibility). Protégé and OntoEdit have Edit menu with Undo/Redo options, but they are disabled.

Regarding the auditing requirement, OilEd has the activity log. However, it records connections to the reasoner, not all ontology modifications. Protégé also has the command history option, but it is useless, since it is disabled.

As known to authors, none of the existing systems for ontology development and maintenance offer support for (semi-)automatic ontology improvement, even though it makes the ontology easier to understand and cheaper to modify.

Most of the existing ontology editors have a very similar layout. They are ergonomically correct to minimise human errors. They enable operating "quickly" enough, as this is often considered being one of the most important easy-for-use

[7] http://protege.stanford.edu/

[8] http://www.ontoprise.de/com/co_produ_tool3.htm

[9] http://oiled.man.ac.uk/

issues. Moreover, all editors can detect logical conflicts (verification), but they do not provide enough information to analyse the sources of conflicts. However, none of the existing editors provide the means to answer to the questions such as how, why, what if, etc. (validation).

4. Conclusion

In order to enable the user to obtain the ontology most suitable to his or her needs, we investigate the requirements for an ontology editor in order to customise the ontology evolution process. We identify several means to do that: to enrich the list of possible changes; to enable the user to set up one of the evolution strategies that are used for resolving the changes; to inform her about all effects of a change; to allow undoing changes; to allow inspecting the performed changes; to suggest the user to generate a change and to identify inconsistency and to provide answers to the questions such as how, why, what if, etc.

We believe that an ontology editor that fulfils these requirements will enable maintaining an ontology more easily and according to the user's preferences.

References

1. A. Das, W. Wu, D. McGuinness, *Industrial Strength Ontology Management*', The Emerging Semantic Web, IOS Press, 2002.
2. D. Fensel, *Ontologies: Dynamics Networks of Meaning*, In Proceedings of the the 1st Semantic web working symposium, Stanford, CA, USA, July 30th-August 1st, 2001.
3. W. Huersch, *Maintaining Consistency and Behaviour of Object-Oriented Systems during Evolution*, PhD thesis, College of CS, Northeastern University, Boston, 1995.
4. M. Klein and D. Fensel, *Ontology versioning for the Semantic Web*, Proc. International Semantic Web Working Symposium (SWWS), USA, 2001.
5. N. F. Noy, D. McGuinness, *Ontology Development 101: A Guide to creating your first Ontology*, Stanford KSL Technical Report KSL-01-05, 2000.
6. A. Maedche and S. Staab, *Ontology Learning for the Semantic Web*, IEEE Intelligent Systems, 16(2), March/April 2001. Special Issue on Semantic Web, 2001.
7. S. Staab, H.-P. Schnurr, R. Studer and Y. Sure, *Knowledge Processes and Ontologies*, IEEE Intelligent Systems. 16(1), Special Issue on KM, 2001.
8. L. Stojanovic, N. Stojanovic and R. Volz, *Migrating data-intensive Web Sites into the Semantic Web*, In ACM Symposium on Applied Computing SAC, 2002.
9. L. Stojanovic, A. Maedche, B. Motik, N. Stojanovic, *User-driven Ontology Evolution management*, In Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW, Madrid, Spain, 2002.
10. N. Stojanovic, L. Stojanovic, *An Approach for the Evolution of Ontology-based Knowledge Management Systems*, EKAW'2002 Workshop on Knowledge Management through Corporate Semantic Webs, 2002.
11. Y. Sure, *On-To-Knowledge -- Ontology based Knowledge Management Tools and their Application*, In: German Journal Kuenstliche Intelligenz, Special Issue on Knowledge Management (1/02), 2002.

Assessment of Ontology-based Tools: A Step Towards Systemizing the Scenario Approach

Alain Giboin, Fabien Gandon, Olivier Corby, and Rose Dieng

INRIA Sophia Antipolis, Acacia Project,
2004 route des Lucioles, B.P. 93,
06902 Sophia Antipolis Cedex, France
{giboin|gandon|corby|dieng}@sophia.inria.fr
http://www.inria.fr/recherche/equipes/acacia.en.html

Abstract. Scenarios have been already used for designing and evaluating ontology-based tools. For example, the so-called "motivating scenarios" are a core component of the TOVE ontological engineering method elaborated by Grüninger and Fox (1995; Uschold and Grüninger, 1996). We ourselves used a "scenario approach" for designing and evaluating CoMMA, a corporate memory computer platform based on ontologies and agents; the approach was inspired by the scenario approaches proposed in the HCI and CSCW communities, which we consider more user-oriented than the "motivating scenarios" approach. In this paper, we account for our CoMMA experience and its major lesson: the *necessity to apply the scenario approach more systematically* for assessing the usability and utility of ontology-based tools.

1 Introduction

In "*Some Ideas and Examples to Evaluate Ontologies,*" Asunción Gómez-Pérez [17] made the distinction between evaluation and assessment of knowledge sharing technology (KST), which include ontology-based tools (OBTs): "Evaluation means to judge technically the features of KST, and assessment refers to the usability and utility of KST in companies" – more precisely, as stated elsewhere by Gómez-Pérez [18], assessment refers to "the usability and utility of the ontologies, software environment, and their documentation when they are used within a given organization or by software agents."

The ontology community seems to be more concerned with the evaluation of OBTs, and with providing technological evaluation criteria such as interoperability, "turn around ability," performance, memory allocation, scalability, or integration into frameworks (see, e.g., [1]). In this paper we will rather focus on assessment of ontology-based tools, and on providing usability and utility criteria motivated by scenarios of use. Why? Our own experience of OBT design makes us think that we would not neglect assessment if we want to get "a consistent level of quality and thus acceptance" of OBTs by industry.

A way to give its place to assessment, we claim, is to make a more systematic use of user-centered scenarios, or to apply a scenario approach more systematically. This claim rests on, and is a major lesson of, our experience of the design and assessment of CoMMA, a corporate memory computer platform based on ontologies and agents [6][12][13], and on Corese, a semantic search engine designed in our research team [7][8]. In this paper, we will account for our CoMMA experience, and introduce some considerations about the systemizing of the scenario approach to ontology-based tool design and assessment.

2 Limitation of the "Motivating Scenarios" Approach Familiar to the Ontology Engineering Community

From February 2000 to January 2002, we participated to the IST European CoMMA project aimed at designing the CoMMA platform. The CoMMA project gave us the opportunity to apply a scenario approach to both requirements analysis (design) and assessment (evaluation) of the CoMMA platform – requirements analysis and evaluation being interleaved: "For requirements analysis, the aim is to 'get at' the user needs; for evaluation the aim is to 'tune' the system to make sure that it really does meet those needs" [25].

The requirements analysis of CoMMA was initially oriented by the two following scenarios: *(1) NEI Scenario:* The "integration of new employees" in a company; it concerns the new employees who need to handle a lot of new information about their enterprise in a very short time, to be rapidly efficient; *(b) TM Scenario:* the diffusion of innovative ideas among employees particularly when dealing with "technology monitoring activities;" it concerns the necessity for each enterprise to access in a very effective way to information concerning technology movement through the Internet that could contribute to its development. These scenarios were originally committed to the two industrial partners of the CoMMA project – a German and an Italian telecommunication company – who took the role of the application end-users.

Because the two scenarios were very abstract and vague, we needed to specify them to get requirements that could be converted into operational system specifications. How did we achieve this? We could have applied the "motivating scenario" approach, now classical within the ontology engineering community, and which underlies the TOVE ontological engineering method elaborated by Grüninger and colleagues [10][19][20].

Motivating scenarios are a core component of the TOVE method. The notion of a motivating scenario refers to a "detailed narrative about the enterprise where emphasis is placed on the problems that the enterprise is facing or the tasks it needs to perform to solve the problems" (e.g., improving enterprise planning and scheduling). Ontology engineers use these problems to define an ontology's requirements in the form of *competency questions* that an ontology must be able to answer (e.g., *What sequence of activities must be completed to achieve some goal? At what times must these activities be initiated and ter-*

minated?). The competency of the ontology is tested by proving completeness theorems with respect to the competency questions.

We however found a main limitation to the “motivating scenarios approach:” the informality and user-orientation present in the first steps of the OBT design process were lacking in the evaluation step. We needed a more user-centered approach. Hence we turned towards the scenario approaches proposed by the Human-Computer Interaction (HCI) and Computer-Supported Cooperative Work (CSCW) communities, e.g., the approach of Carroll and his colleagues [4][5].

3 Exploiting Scenario Approaches Familiar to the HCI and CSCW communities

The scenario approaches have been introduced in the HCI and CSCW communities to fill the gap that the “traditional approach” to design created by imposing a technological orientation, abstraction, and other “user-distant” features. The scenario approaches allowed a design team to reintroduce the user’s viewpoint in the design cycle, and to take into account her need of speaking of the system in terms of the work she has to achieve, using concrete and specific terms, and so on. In Carroll and colleagues approach, for example, scenarios of use are defined as descriptions, often narratives, of what people (could) do and experience (e.g., problems) when using computer systems. Scenarios can be developed through direct observation of users performing tasks in their work environment (observed scenarios), or through abstractions from theories of human activities (envisioned scenarios).

By exploiting the scenario approaches of the HCI and CSCW communities, our aim was to balance technology-orientation (prevalent in the ontology engineering community) with user-orientation (recommended by the HCI and CSCW communities), and, more specifically, to balance formality (which is a strong standard within the ontology engineering community) with informality (which is a HCI and CSCW requirement for not losing touch with the user, see e.g. [3]).

Scenarios are a meaningful way of accounting for users’ needs. They embody properties, qualities or criteria that must be “put” in the system so that the system be accepted by its intended users. Scenarios embody criteria that must be found when assessing the system. Scenarios are both requirements and assessment scenarios. This two-sided aspect of scenarios would need to be systemized.

4 Eliciting Scenarios for Requirements Analysis

Applying a Scenario Approach Supposes To Have a Model of Scenario. The scenario formats an techniques proposed by the HCI and CSCW communities are multiple: e.g., scenarios, use cases, examples, stories, narrative descriptions of context, mock-ups, etc. We did not privilege a particular technique or format, but collected from the existing ones the elements that could help us answer methodological questions like: Which types of scenarios did we want to produce? Which contents shall we give to these scenarios? Which procedures are worth to follow to fill the scenario slots? As a result, in collaboration with the industrial partners, we elaborated a scenario grid to be used for requirements analysis by the partners (see Table 1).

Table 1. The CoMMA scenario elicitation grid

CHARACTERISTICS	REPRESENTATIONS	FACETS	
Goal:	*Textual :*	*Actors*	*Resources*
	Graphical :	Profile	Nature
Before:		Role	Services
After:	*Informal :*	Individual goal	Constraints
	Formal :	Task	
	(e.g., UML)	Action	*Flows*
Scope:		Interaction	Inputs
			Outputs
Scenario		*Logic & Chronology*	Paths
Sub-Scenario:		Processes	
		Decomposition	*Environment*
Generic		Sequential/Parallel/	Internal
Specific		Non-deterministic	Organization
Example		Loops & Stop conditions	Acquaintance
	In one scenario description, several types of representations may be used.	Alternatives & Switches	External
Relevance life-time		Compulsory/Optional	
		Functionalities & Rationale	
		Functionalities description	
Exceptions		Motivation, necessity	
Counter examples		Advantages & Disadvantages	

In the grid, “Characteristics” and “Representations” allow to specify the type of scenario to elicit, e.g., a scenario informally describing, in a textual format, a specific existing situation. “Facets” refer to the contents of the scenario: the actors involved in the scenario, having certain roles, using certain resources, performing a certain task in a certain way, and so on. To each of the elements of the scenario grid, we associated definitions and examples, and also questions to help industrial partners elicit relevant knowledge. For example, to the “Actors Interaction” facet, we associated the questions: *Who helps you to perform your job? Which persons do you consult to get information ... / to get that ... done?*

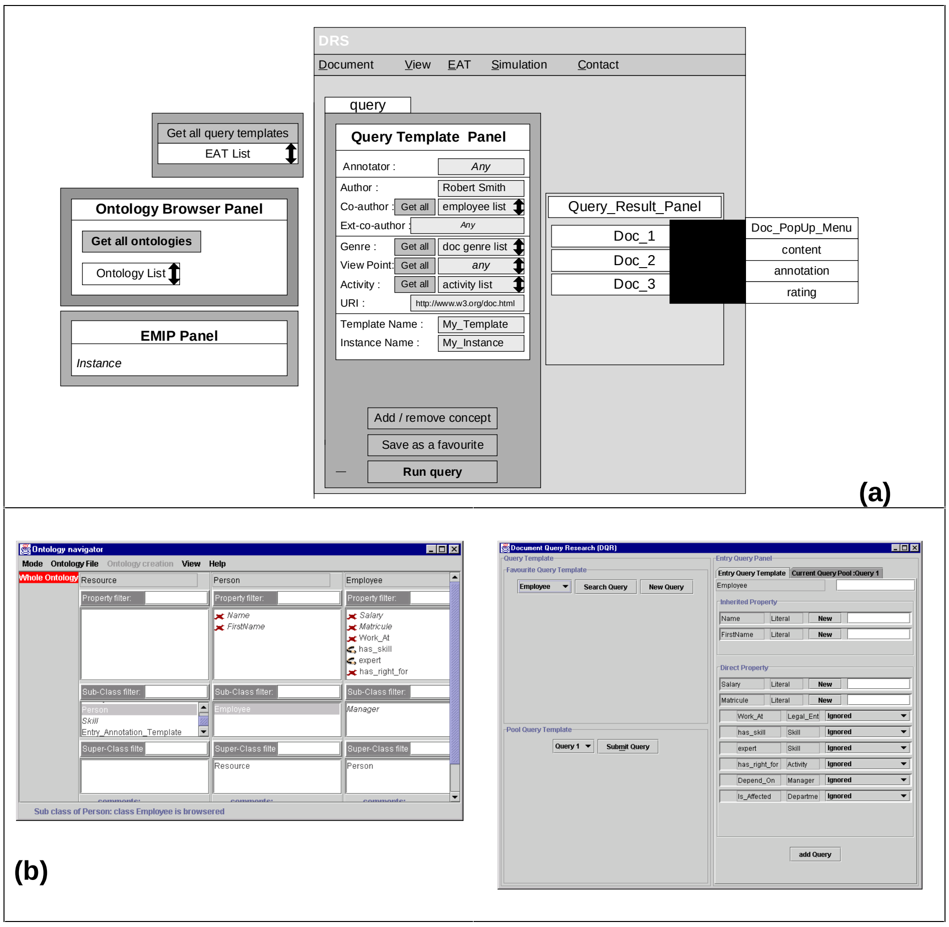


Fig. 1. The first CoMMA interfaces. Example of the Document Retrieval System (DRS) for the Technology Monitoring scenario: *(a)* as designed; *(b)* as implemented (*Left:* the "Ontology Browser Panel;" *Right:* the "Entry Query Template" corresponding to the selection of a concept into the Ontology Browser Panel)

To each of the scenario elements, we also associated techniques and potential corporate sources that can be exploited to answer the questions. The grid was given to the industrial partners who found them helpful for requirements analysis. A number of scenarios significant to end-users were thus elicited.

Applying a Scenario Approach Supposes a Continuous Focus on Scenarios. The CoMMA project was divided into two phases, each one ending with a trial at each industrial partner site. We must admit that, during the first phase, after requirements analysis, we lost sight of the scenarios, and that we consequently lost touch with the end-users. There were two reasons to this: (1) end-user partners were not truly available for the trial preparation and execution (one of them even withdrew from the Consortium, and was replaced by another partner belonging to another industrial sector: construction); (2) the priority of the research partners in the first phase was to perform and test the integration of new technologies, not really to meet end-users' needs. The result of this distance from users was foreseeable: immersed in technology, abstraction, and formality, the Consortium designed interfaces for developers and ontologists, and not interfaces for end-users. Figure 1 gives an idea of these interfaces for technologists. So it is not enough to have an early focus on scenarios of use and on users, it is necessary to have a continuous focus on them.

4 Using Scenarios for Tool Assessment

Using Scenarios to Assess Functionalities with Users. As a consequence of the distance from users, our interfaces were definitely not usable by end-users, and direct usability testing of these interfaces by end-users was impossible. Being however convinced that developing an OBT was a promising solution for supporting corporate memory management, we decided not to give up, and to show the interest of the CoMMA solution to potential end-users by making the CoMMA functionalities tangible through scenarios familiar to users. The goal was to describe the functionalities in terms of the work users will perform with the system.

We illustrated the functionalities of CoMMA through various scenario formats, in particular *storyboards* – a scenario format mixing text and images – of actual information-seeking newcomers' activities within the intranet of one end-users' company. These scenarios allowed to identify specific processes likely to be performed when using the system, e.g.:

> *"Travel Expenses Refund" scenario (excerpt).* A newcomer was seeking instructions in the intranet of his company for the refunding of his travel expenses. During the information-seeking process, the newcomer proposed different keywords to the successive search engines he utilized, or he followed various links related to "Travel expenses." Table 2 provides the sequence, and the transformation, of keywords entered and links followed by the newcomer during his activity. The contents of Table 2 illustrates a user's continuously changing process that we can call "term/concept shifting" (further discussed below).

Table 2. The series of transformations of the keywords used by a new employee searching for instructions for the refunding of his travel expenses

GERMAN KEYWORDS OR LINKS USED	ENGLISH TRANSLATION
Reisekostenabrechnung	**Travel expenses account**
Reisekosten**richtlinie**	Travel expenses **guideline**
R~~eisekostenrichtlinie~~ *(R refers to:* Reisekosten~~richtlinie~~ *)*	Travel expenses ~~**guideline**~~
Reise~~**kosten**~~**antrag**	Travel ~~**expenses**~~ **request**
Reise~~**antrag**~~	Travel ~~**request**~~

(*Convention:* The part of the keyword which did not change from the previous turn to the current turn is printed in gray.)

Going back to the scenario approach, and consequently getting again in touch with end-users, we can show that end-users found the functionalities very useful, and suggest refinements and extensions to these functionalities (e.g., term/concept shifting illustrated in Table 2).

Using Scenarios to Assess Interfaces without Users. Using two scenario-based techniques – namely, Heuristic Evaluation [23] and Cognitive Walkthrough [27] –, we were all the same able to assess the CoMMA interfaces without users, but by putting ourselves in the users' shoes. This indirect assessment permitted us to identify usability problems, to propose recommendations for overcoming them, and to suggest interface specifications based on these recommendations (for details, see [15][16]). The two scenario-based methods have the following advantages:

Contextualizing Assessment Criteria through Scenarios. The Heuristic Evaluation technique consists for the evaluator in looking for violations of common usability principles or heuristics, such as *Flexibility and efficiency of use*: "Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions." The usability inspection is greatly facilitated when evaluators are provided with scenarios: the criteria being affected by the context of use (i.e., user's characteristics, task, environment), scenarios allow to contextualize the criteria, and to make them meaningful. For example, "concept/term shifting" of "Travel Expenses Refund" scenario can be related to a Flexibility issue.

Justifying Scenarios with Activity Models. The Cognitive Walkthrough technique consists for the evaluator in "walking through" the interface, trying to act as a user. The walkthrough process involves examining each individual action step and trying "to tell a believable story" (scenario) about why the prospective user would choose an action. Scenarios in Cognitive Walkthrough are based on, and justified by, a model of exploratory learning of the system, which describes human-computer interaction in terms of four steps:

1. The user sets a goal to be achieved with the system (for example, “I am searching for corporate instructions for the refunding of my travel expenses”).
2. The user searches the interface for currently available actions (menu items, buttons, ontology browsing, etc.).
3. The user selects the action to progress toward the goal (e.g., browse the lists of concepts/terms for the concept/term “Travel expenses guideline”).
4. The user performs the selected action and evaluates the system's feedback for evidence of her progress (e.g., “I see that the term ‘guideline’ doesn't exist to refer to the concept of ‘instructions,’ but a synonymous term exists, that I can use to access to the corporate document I need”).

If we admit that the strength of an assessment method like the Cognitive Walkthrough depends on the relevance of its underlying model, a further step in the systemizing of the scenario approach would be to propose other models of human activity to justify the scenarios, e.g., models of the users’ linguistic activity. For example, the “Travel Expenses Refund” scenario could be explained by the notion of “concept drift” used in the Machine Learning community [22]. It can be also explained by the “vocabulary problem” model [11].

The models we spoke about so far are models of a user's individual activity, which are the most familiar to the HCI community. If we consider OBTs as tools supporting collective activities (e.g., elaborating a common terminology, sharing knowledge, etc.), we will need to refer also to models of collective activity, which are most familiar to the CSCW community. For example, the “Travel Expenses Refund” scenario could be justified by models like “lexical entrainment” [2], “concept and terminology co-ordination” [14], or “ontological drift” [24].

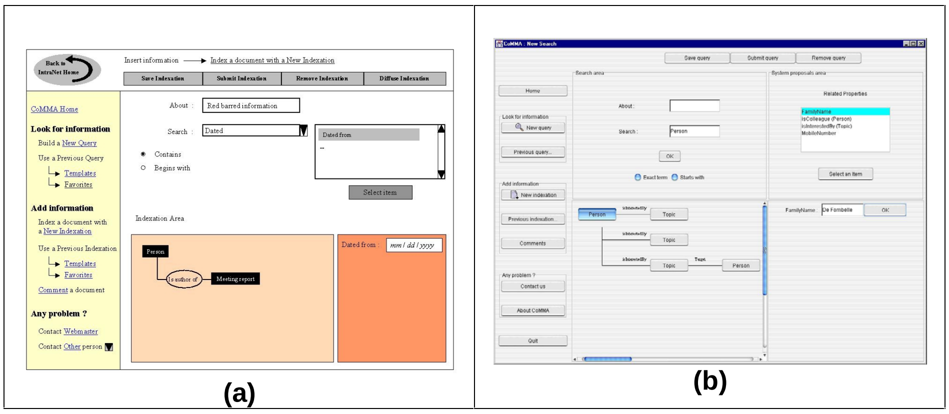


Fig. 2. The second CoMMA interfaces. Example of the Document Retrieval System (DRS) for the New Employee scenario: (a) as designed (with PowerPoint); (b) as implemented

Using Scenarios to Assess Interfaces With Users. For the second phase of the CoMMA project, we indeed made the necessary arrangements for not losing sight of the scenario approach, and not losing touch with the end-users. Among the arrangements we made were the following ones: (a) creating a HCI group, including, among others, end-users, interface developers, and human factors specialists; (b) involving the group in an iterative cooperative design/evaluation process; (c) inciting the group members to use scenario-based representations to discuss about the design and evaluation of the new interfaces. . As a result, we got simplified interfaces, that made sense to the users, and which users found this time usable (see Figure 2; for details of Trial 2, see [9]). However the process was very time-consuming.

5 Conclusion

In their *"Whitepaper: Evaluation of Ontology-based Tools,"* Angele and Sure [1] encourage the ontology engineering community, and more broadly the semantic web community, "to enforce their research efforts by developing further standard criteria [...] and tools that implement these criteria to evaluate ontologies and related technologies." Through the present paper, we tried to contribute to these efforts, showing for example that criteria development cannot be considered in isolation from situations in which the ontology-based tools will be used: to be meaningful and relevant, criteria need to be connected to scenarios of use, and these scenarios to be explained and further analyzed need to be connected to activity models. Put in other words, we claimed in this paper for a balance between usage and technology, and between formality and informality; in fact we advocated for avoiding premature formalization (as pointed out by Buckingham Shum [3]), or reinstalling informality when interacting with end-users.

Through this paper we invite the community to bring some efforts to bear on systemizing the scenario approach to assessment (and design), an approach more developed in the HCI and CSCW communities than in the ontology engineering community. It would be desirable to discuss also how to systemize the scenario approach for technical evaluation; the work by Kazman and his colleagues [21] on scenario-based evaluation of architectures is worth considering in such a discussion.

Acknowledgements. We would like to thank our colleagues of the CoMMA Consortium for their involvement in the assessment of the COMMA platform, especially: Sabine Delaître (CSTB), Bruno Fiès (CSTB), Karim de Fombelle (Atos Integration), Joachim Hackstein (T-Systems Nova), Hervé Karp (Atos Integration), Jérôme Leytier (INRIA), Emmanuelle Loyson (CSTB), Philippe Pérez (Atos), and Marie Soleilhet (Atos Integration). We would also like to thank two anonymous reviewers for their comments on a previous version of this paper.

References

1. Angele, J., Sure, Y.: *Whitepaper: Evaluation of Ontology-based Tools*. Excerpt from the IST-2001-29243 Report, OntoWeb. D1.3. Tools. (2001). Available at: http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/eon2002_whitepaper.pdf
2. Brennan, S. E.: Lexical entrainment in spontaneous dialog. *Proceedings of the 1996 International Symposium on Spoken Dialogue*, Philadelphia, PA: ISSD-96 (1996) 41-44. Available at : http://www.psy.sunysb.edu/sbrennan/papers/brenISSD.pdf
3. Buckingham Shum, S.: Balancing Formality with Informality: User-Centred Requirements for Knowledge Management Technologies. *AAAI Spring Symposium on Artificial Intelligence in Knowledge Management* (1997), Stanford University, Palo Alto, CA. AAAI Press. Available at: http://kmi.open.ac.uk/people/sbs/org-knowledge/aikm97/sbs-paper1.html
4. Carroll, J.M.: *Making Use: Scenario-Based Design of Human-Computer Interactions*. MIT Press, Cambridge, MA (2000)
5. Carroll, J.M., Mack, R.L., Robertson, S.P. & Rosson, M.B.: Binding objects to scenarios of use. *International Journal of Human-Computer Studies 41* (1994) 243-276.
6. CoMMA Consortium: CoMMA: Corporate memory through agents, *Proceedings of E-Work and E-Business'2000* (2000)
7. Corby, O., Dieng, R., Hébert, C.: A Conceptual Graph Model for W3C Resource Description Framework, *Proceedings of ICCS 2000*, Darmstadt, Germany (2000)
8. Corby, O., Faron-Zucker, C.: Corese: A corporate Semantic Web engine. *Proceedings of the International Workshop on "Real World RDF and Semantic Web Applications,"* WWW'2002, Hawai. (2002)
9. Fiès, B., (Ed.) (2002). Assessment Report of CoMMA Trial-step 2, CoMMA project Deliverable.
10. Fox, M.S., Grüninger, M..: Enterprise Modelling, *AI Magazine* (1998) 109-121.
11. Furnas, G.W., Landauer, T.K., Gomez, L.M., and S.T. Dumais.: The Vocabulary Problem in Human-System Communication, *Communications of the ACM 30* (1987) 964-971.
12. Gandon, F.: *Ontology Engineering: A Survey and a Return on Experience.* INRIA Research Report # RR4396, INRIA, France (2002). Available at: http://www.inria.fr/rrrt/rr-4396.html
13. Gandon F., Dieng R., Corby O. et Giboin A.: A multi-agent system to support exploiting an XML-based corporate memory, *Proceedings of PAKM'2000, the Third International Conference on Practical Aspects of Knowledge Management*, Basel, Switzerland (2000)
14. Garrod, S. How groups coordinate their concepts and terminology: implications for medical informatics. *Proceedings of the WG6 IMIA Symposium on Concepts and Terminology*, Jacksonville, Fl. (1997) 279-284
15. Giboin, A, Pérez, Ph. (Eds.): *Assessment Report of [CoMMA] Trial-step 1. Part 1 Technical Evaluation*, CoMMA Project (IST-1999-12217) Deliverable # COMMA/WP6/D10, 69 pages (2001a)
16. Giboin, A, Pérez, Ph. (Eds): *Assessment Report of [CoMMA] Trial-step 1. Part 2 User Evaluation*, CoMMA Project (IST-1999-12217) Deliverable # COMMA/WP6/D10, 156 pages, (2001b)
17. Gómez-Pérez, A.: *Some Ideas and Examples to Evaluate Ontologies*. Technical Report # KSL-94-65, Knowledge Systems Laboratory. Stanford University (1994a). Available at: http://www-ksl.stanford.edu/KSL_Abstracts/KSL-94-65.html

18. Gómez-Pérez, A.: *From Knowledge Based Systems to Knowledge Sharing Technology: Evaluation and Assessment*. Knowledge Systems Laboratory, Technical Report # KSL-94-73 (1994b)
19. Grüninger, M., and Fox, M.S.: The Role of Competency Questions in Enterprise Engineering, *Proceedings of the IFIP WG5.7 Workshop on Benchmarking – Theory and Practice*, Trondheim, Norway (1994)
20. Grüninger, M., and Fox, M.S.: Methodology for the design and evaluation of ontologies, *Proceedings of the IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*, AAAI Press, Menlo Park CA, (1995). Available at: http://www.ie.utoronto.ca/EIL/public/org.ps
21 Kazman, R., Carriere, S. J., Woods, S. G.: Toward a discipline of scenario-based architectural engineering, *Annals of Software Engineering 9* (2000) 5-33.
22. Lane, T. and Brodley, C.E.: Approaches to online learning and concept drift for user identification in computer security. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (1998) 259-263.
23. Nielsen, J.: Heuristic evaluation. In: Nielsen, J., and Mack, R.L. (Eds.), *Usability Inspection Methods*, John Wiley & Sons, New York, NY (1994)
24. Robinson, M. and L. Bannon: Questioning representations. *Proceedings of ECSCW'91, the Second European Conference on Computer-Supported Cooperative Work*, Amsterdam, The Netherlands (1991). Available at: http://www.ul.ie/~idc/library/papersreports/LiamBannon/15/QuestFin.html
25. Thomas, P.J. (Ed.): *CSCW Requirements and Evaluation*, Springer Verlag, London (1996)
26. Uschold, M. and Grüninger M.: Ontologies: Principles, methods and applications. *Knowledge Engineering Review 11* (1996). Also available as AIAI-TR-191 from AIAI, The University of Edinburgh.
27. Wharton, C., Rieman, J., Lewis, C., and Polson, P.: The Cognitive Walkthrough method: A practitioner's guide. In J. Nielsen and R.L. Mack (Eds.), *Usability Inspection Methods*, New York: John Wiley & Sons, (1994) 105-141.

OntoManager: A Workbench Environment to facilitate Ontology Management and Interoperability

Adil Hameed, Derek Sleeman, Alun Preece

Department of Computing Science
University of Aberdeen
Aberdeen AB24 3UE
Scotland, U.K.
{ahameed, dsleeman, apreece}@csd.abdn.ac.uk
http://www.csd.abdn.ac.uk/research/

Abstract. The prime motivation for our research is to enable sharing and reuse of domain knowledge through the engineering and management of ontologies. We contend that there is a need to reconcile ontologies by harmonising mismatches and discrepancies that are present among them. This is a necessary task before any stakeholders can begin to share and/or reuse the underlying knowledge (re)sources. Our key objective is to detect and resolve these mismatches in a consistent and verifiable manner. We have evaluated the state-of-the-art in ontology management tools and selected the best-in-class techniques and methods. We propose implementing a workbench that will integrate these tools and enable interoperability between them in order to facilitate the management of ontologies.

1 Introduction

Researchers have identified various kinds of ontological discrepancies [1, 2, 3, 4] and several types of inconsistencies that are inherent in knowledge and data sources [5, 6]. Also, impediments that are likely to occur during the elicitation of knowledge from multiple experts have been recognised [7]. Further, suggestions have been made about classifications and categorisations of such mismatches [2, 3, 6]. Recently, there has been considerable interest in developing tools and techniques to assist in a variety of ontology management operations, e.g., mapping, merging, alignment, integration, etc. [8, 9, 10, 11, 12, 13]. For any of these processes to be carried out successfully, it is inevitable that mismatches be detected and resolved. None of the available tools tackle all the types of discrepancies we have identified [3]. Moreover, the various tools operate on ontologies expressed in different knowledge formalisms. Essentially, since none of the current approaches are designed to address every type of mismatch, there is a compelling case for providing interoperability between the tools.

2 Background and Motivation

Our focus has been on the engineering & management of ontologies built from knowledge elicited directly from human experts. We make a distinction between *experts'* ontologies (based on inherent conceptualisations) as opposed to *artefact* ontologies.

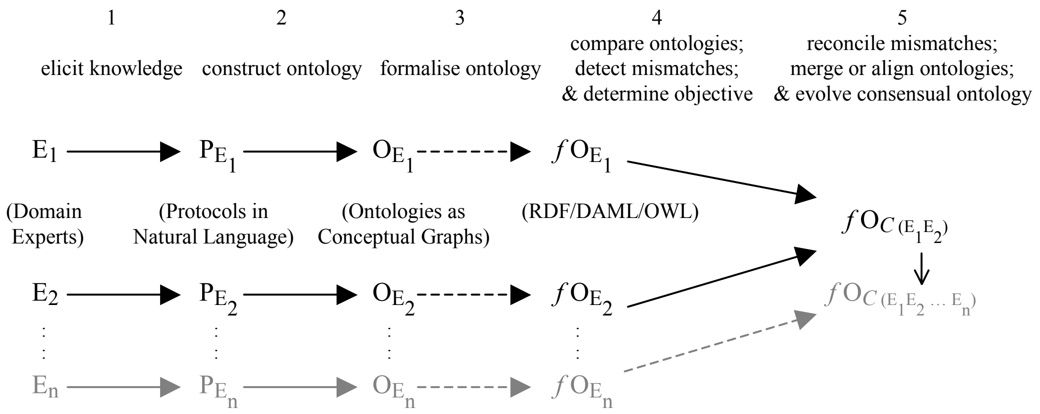

1
elicit knowledge
2
construct ontology
3
formalise ontology
4
compare ontologies; detect mismatches; & determine objective
5
reconcile mismatches; merge or align ontologies; & evolve consensual ontology
E1
PE1
OE1
ƒOE1
(Domain Experts)
(Protocols in Natural Language)
(Ontologies as Conceptual Graphs)
(RDF/DAML/OWL)
ƒOC(E1E2)
E2
PE2
OE2
ƒOE2
ƒOC(E1E2 ... En)
En
PEn
OEn
ƒOEn

Fig. 1. Stages illustrating our evolutionary approach towards the engineering and management of *Experts' Ontologies*

Knowledge was elicited from domain experts ($E_1 \dots E_5$) (Fig. 1) in the form of natural language protocols (P_{E_1}, P_{E_2}, ...) which were then analysed by a systematic approach we developed to construct individual expert's ontologies (O_{E_1}, O_{E_2}, …) [14]. These ontologies were represented in a semi-formal notation as conceptual graphs [15]. In order to aid machine-interpretation and reasoning, it is necessary to formalise the ontologies and 'transform' them into a more expressive representation. We are investigating the efficacy of standard knowledge representation forms such as RDF, DAML+OIL, and the evolving OWL [16].

Since we have detected a wide array of mismatches among our experts' ontologies, extending from simple syntactic discrepancies to a range of rich semantic inconsistencies [3], we realised that it is not plausible to evolve an all-encompassing solution. Instead, we propose an approach based on interoperability between various best-in-class tools. This approach will take into account the type of mismatch and suggest an appropriate and feasible resolution process. In addition to the acknowledged approaches in knowledge representation, ontology engineering, and description logics, we are also investigating novel techniques from areas such as design patterns, fuzzy & softcomputing, among other promising techniques.

3 Ontology Management: The *OntoManager* Workbench

We have sought to assess empirically the effectiveness of the state-of-the-art in ontology management tools. Key features of prominent tools such as PROMPT, Chimaera, FCA-Merge, ODEMerge, ONION, OntoView, etc. are being appraised [17]; first with sample ontologies provided by the respective designers, and then with experts' ontologies from a common domain (PC specification) that were constructed from independently elicited knowledge [14].

After experimenting with these tools, we have obtained a clear understanding of both their strengths and their limitations. An analysis of the limitations has helped us focus on developing techniques that should address issues/problems that these tools have not yet tackled. An insight into each of their strengths has also enabled us to identify particular algorithms and techniques that are currently best-in-class.

We are developing an interactive tool to semi-automate the detection and resolution of various ontological mismatches. We now plan to extend this tool into a workbench environment: *OntoManager,* where different 'procedures/methods' can be added to aid in the resolution of specific kinds of mismatches. It is also anticipated that the tool itself could work with existing ontology development systems such as Protégé, OntoBroker/OntoEdit, WebODE, KAON, ConcepTool, etc. It is envisaged that when these tools are encompassed within OntoManager, it would be able to employ or at least recommend the most suitable tool/technique that could help resolve a specific type of ontological mismatch or discrepancy. The system is being implemented in Java. We aim to demonstrate the integration of at least two of the above tools, and show how interoperability can be achieved between the built-in techniques they offer and the heuristic methods that we have developed.

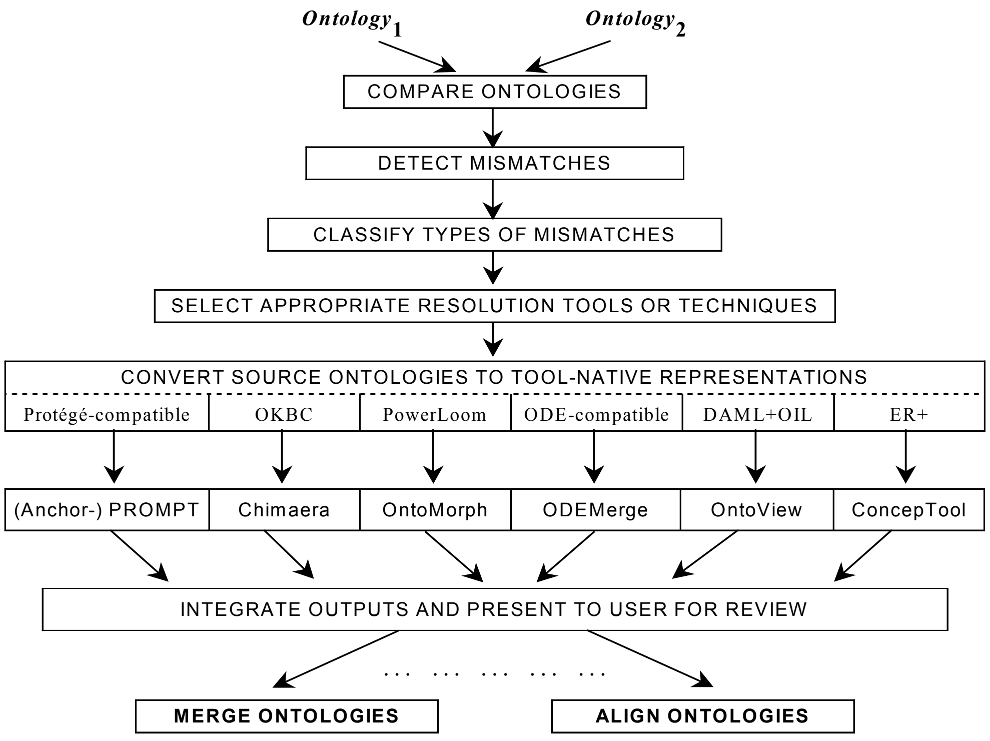


Fig. 2. A schematic diagram of the OntoManager: a workbench environment to facilitate the management of ontologies

Ontology management and interoperability can provide key solutions to the many challenges posed by the progressive transformation of the current WWW into the Semantic Web [18]. Also, the success of the much-advocated Web/Grid Services, which are predicted to proliferate, will certainly depend on successful reconciliation among underlying ontologies.

It is therefore conceivable that when this workbench is deployed in a distributed environment like the Internet, it will provide an innovative and a valuable ontology and knowledge management service for the Semantic Web/Grid.

References

1. Hameed, A., Sleeman, D., & Preece, A.: Reconciling Experts' Ontologies for the Semantic Web. Presented at the First Int. Semantic Web Conf. (ISWC-2002), Sardinia, Italy (2002).
2. Visser, P.R.S., Jones, D.M., Bench-Capon, T.J.M., & Shave, M.J.R.: An Analysis of Ontology Mismatches; Heterogeneity vs. Interoperability. AAAI'97 Spring Symposium on Ontological Engineering, Stanford (1997)
3. Hameed, A., Sleeman, D., & Preece, A.: Detecting Mismatches among Experts' Ontologies Acquired through Knowledge Elicitation. In R&D in Intelligent Systems XVIII, Proc. ES2001: 21st SGES Int. Conf. on Knowledge Based Systems and Applied Artificial Intelligence, Cambridge, U.K., Springer-Verlag, London (2001) 9–24. Also, in Knowledge-Based Systems, 15(5-6) (2002) 265–273
4. Klein, M.: Combining and relating ontologies: an analysis of problems and solutions. Workshop on Ontologies and Information Sharing, IJCAI'01, Seattle, USA (2001)
5. Ceri, S. & Widom, J.: Managing Semantic Heterogeneity with Production Rules and Persistent Queues. Proc. 19th Int. Conf. on Very Large Data Bases, Dublin (1993) 108-119
6. Wiederhold, G.: Interoperation, Mediation & Ontologies. 5th Generation Computer Systems'94 Workshop on Heterogeneous Cooperative Knowledge-Bases, Tokyo (1994) 33-48
7. Shaw, M.L.G., & Gaines, B.R.: Comparing Conceptual Structures: Consensus, Conflict, Correspondence and Contrast, in Knowledge Acquisition, 1(4) (1989) 341-363
8. Hovy, E.: Combining & standardizing large-scale, practical ontologies for machine translation & other uses. 1st Int. Conf. on Language Resources & Evaluation, Granada (1998)
9. Pinto, H.S.: Towards Ontology Reuse, in the Proceedings of AAAI99's Workshop on Ontology Management, WS-99-13, AAAI Press (1999) 67-73
10. McGuinness, D.L., Fikes, R., Rice, J., & Wilder, S.: An environment for merging and testing large ontologies, in Cohn, A., Giunchiglia, F., Selman, B. (eds.), KR2000: Principles of Knowledge Representation and Reasoning, San Francisco (2000) 483-493
11. Noy, N.F. & Musen, M.A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In Proc. AAAI'00, Austin (2000)
12. Mitra, P., Kersten, M., & Wiederhold, G.: Graph-Oriented Model for Articulation of Ontology Interdependencies, in the Proceedings of the 7th Int. Conf. on Extending Database Technology, Springer-Verlag (2000)
13. Stumme, G. & Maedche, A.: Ontology Merging for Federated Ontologies on the Semantic Web. Workshop on Ontologies and Information Sharing, IJCAI'01, Seattle, USA (2001).
14. Hameed, A., & Sleeman, D.: Knowledge Elicitation to construct Ontologies in the domain of PC Specification. AUCS/Technical Report TR0001. Department of Computing Science, University of Aberdeen (2000)
15. Sowa, J.F., ed.: Conceptual Graphs, draft proposed American National Standard, NCITS.T2/98-003 (1998)
16. W3C (The World Wide Web Consortium): Requirements for a Web Ontology Language. W3C Working Draft (2002)
17. Hameed, A.: A Comparative Evaluation of Ontology Management Tools – A state-of-the-art (work-in-progress). Internal report, Dept. Computing Science, Univ. Aberdeen (2002)
18. Berners-Lee, T., Hendler, J., & Lassila, O.: The Semantic Web. Scientific American, 284 (5) (2001) 28-37

NL description on the travelling domain

Let's consider that we are in charge of developing an application for our travel agent in New York, and that we have decided to make use of an ontology to represent explicitly the knowledge that will be used by it. We will focus to travelling and Lodging, but leisure time, cultural events, tours, etc., will be considered in further stages of our ontology.

We know that when a client makes a trip, he chooses: transport and accommodation.

Hence, we start by determining the means of transport that are currently available for a travel agency. We will have in our ontology the following ones: planes, trains, cars, ferries, motorbikes and ships. There are no other kinds of transport. From all of them, the travel agency is specially interested in flights, as it is the means of transport mostly used by its customers. In fact, customers are usually interested in the kind of planes that they will fly on: Is it a Boeing, or is it an Airbus? Furthermore, they are even interested in the specific model of the plane in which they will fly (a Boeing 717 or a Boeing 777). We know that each model of transport belongs only to one kind of transportation (e.g., it's either a plane, or a bus, or a car, etc.

For each flight, the agency knows: the arrival date, the departure date, the arrival city, the departure city, the arrival airport, the departure airport, the prices on first class, business class and economy class, the departure time and arrival time. Time and date will be considered as absolute date.

As for the destinations of customers' travels, they are diverse. Some customers ask for trips to the Statue of Liberty in New York; other ask for trips to Washington, San Francisco, Seattle. There are customers interested in visiting Europe: the most common destinations are London, Paris (either the city or Disneyland Paris) and Madrid. Others are interested in more places, such as Cairo (Egypt). We know that the client can use the following transport to move inside the city: underground, city buses, taxis, and rental cars.

Concerning hotels, the agency recommends in all the cities: hotels, and Bed and Breakfasts. Hotels rank from 1 star hotels to 5 star hotels and each hotel belongs to one of these five categories. For all of them, the agency knows their facilities: address, telephone number, URL, capacity, number of rooms, available rooms, descriptions, dogs allowed, distance to the beach, distance to skiing, etc. The agency also knows the facilities of the rooms: number of beds, rates, TV available, Internet connection, etc.

Once we have defined what are the main elements in our domain, we can go further and try to represent some common sense constraints and deductions that can be performed with them. For instance, we know that it is not possible to go from America to Europe by train, car, bike nor motorbike. Having this information in our system will avoid it to search for possible itineraries using these means of transport when a customer wants to travel to Europe. Another example of this kind of constraint may be related to the distance between the origin and destination of our trip and the available means of transport. If distance between two cities is between 400 and 800 miles, and there is no airport close to one of them, the customer will prefer going by car or by train. The customer also prefer to go by car or train if he hates travel by plane. Distances can be either in km or miles.

Finally, we want to represent knowledge about a concrete trip. John is travelling from Madrid to NY on April 5th, 2002 to see the Statue of Liberty and continuing on to Washington on April 11th. He plans to return to Madrid on April 15th. He has selected two hotels belonging to the Holiday Inn chain in New York and Washington.

http://www.boeing.com/commercial/717/717technical.html provides Boeing717 technical description.

Travelling Domain Experiment: Preliminary Results for OilEd

Sean Bechhofer
Information Management Group
University of Manchester
Kilburn Building
Oxford Road
Manchester M13 9PL

Introduction

OilEd [Bechhofer01b] is a tool that allows the construction of DAML+OIL ontologies. It is targeted primarily at ontology construction (rather than knowledge base population), and thus concentrates on the language constructors related to classes and the relationships between them. A key aspect of **OilEd** is its use of the FaCT reasoner which allows the user to produce classification hierarchies and check classes for inconsistency.

As **OilEd** is not intended for knowledge base construction, this experiment concentrated primarily on trying to represent information about the classes that arise in the example scenario, and the relationships and constraints that appear between them. In addition, as data type support is limited, little was done to model attributes with concrete ranges. As a result, the model produced only really covers a small subset of the application domain – additional functionality would certainly be needed if this were to be used within an application. However, the results demonstrate how we can use DAML+OIL, in particular axioms and expressive power such as negation, for the representation of some quite sophisticated constraints.

OilEd is not an application for query or ontology delivery, so it is difficult to assess in a satisfactory way the model produced in terms of query or use in an application. However, as discussed below, we can see how definitions in the resulting model might be used in applications.

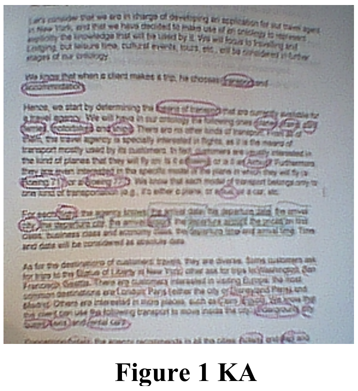
Building the Model

Building the Model consisted of 3 basic stages:

Knowledge Acquisition: identifying the basic classes and properties;
Definition: identifying the relationships between the basic classes;
Constraints: identifying the constraints that limit the ways in which descriptions can be formed.

The process was iterative. Once the basic definitions were in place, these were refined and further elaborated. The reasoner was used regularly throughout the process to assist in organisation of the model and check for consistency.

Vehicles and Accommodation



Let's consider that we are in charge of developing an application for our travel agent in New York, and that we have decided to make use of an ontology to represent explicitly the knowledge that will be used by it. We will focus in travelling and lodging, but leisure time, cultural events, tours, etc., will be considered in further stages of our ontology.

We know that when a client makes a trip, he chooses [illegible] and accommodation.

Hence, we start by determining the means of transport that are currently available for a travel agency. We will have in our ontology the following ones: [illegible], ferries, motorbikes and ships. There are no other kinds of transport. From all of them, the travel agency is specially interested in flights, as it is the means of transport mostly used by its customers. In fact, customers are usually interested in the kind of planes that they will fly on: is it a [illegible] or is it an Airbus? Furthermore they are even interested in the specific model of the plane in which they will fly (a Boeing 7[illegible] or a Boeing 7[illegible]). We know that each model of transport belongs only to one kind of transportation (e.g., it's either a plane, or a bus, or a car, etc.

For each flight, the agency knows the arrival date, the departure date, the arrival city, the departure city, the arrival airport, the departure airport, the prices in first class, business class and economy class, the departure time and arrival time. Time and date will be considered as absolute data.

As for the destinations of customers' travels, they are diverse. Some customers ask for trips to the Statue of Liberty at New York; other ask for trips to Washington, San Francisco, Seattle. There are customers interested in visiting Europe, the most common destinations are London, Paris (either the city or Disneyland Paris) and Madrid. Others are interested in more places, such as Cairo, [illegible]. We know that the client can use the following transport to move inside the city: underground, city buses, taxis and rental cars.

[illegible] the agency recommends in all the cities hotels and [illegible]

Figure 1 KA

The first step was basic knowledge acquisition. The NL description was examined in order to determine what the major classes occurring in the model were likely to be. This basically involved identifying things like the various kinds of Vehicle (Bus, Car etc) and the kinds of Accommodation (Hotel, B&B). Figure 1 shows a marked up section of the scenario. The concepts identified were added to the model as basic primitive classes.

In addition to the concepts, a number of basic relationships and attributes were identified. In the scenario, many of these are what DAML+OIL calls datatype properties, for example arrival or departure dates and times, number of rooms, price etc. The support for data types in **OilEd** is minimal. Relationships can be introduced and asserted to be data type properties and a range can be given, but no complex data types such as ranges are supported.

Once the basic classes had been added, abstract classes were introduced into the model, for example in order to represent the different kinds of vehicle that can be used. These were introduced as primitives, e.g. Air Vehicle, Land Vehicle and Water Vehicle. One possible approach would be to model these as defined classes – for example a Land Vehicle is a vehicle that travels across land. A decision was taken that this was not necessary (at this point). If it became useful to explicitly represent and model these properties, these definitions could subsequently be refactored "in place". An axiom was added stating that the Vehicle class was covered (disjointly) by the three vehicle types (see the discussion below about round tripping). This precludes us from including, for example, amphibious vehicles and asserts that there are no other kinds of Vehicles than the three introduced. Again, for this particular scenario, this was deemed acceptable.

Places

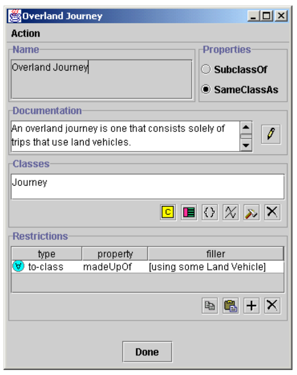
The notion of a Geographical Location encompasses Countries, Cities and Attractions. A rather general notion of containment is provided – Geographical Locations can be in other Geographical Locations. This relationship is transitive, thus an Attraction which is in a City which is in a Country is also in that Country.

Hotels and B&Bs are kinds of Places to Stay. These have a number of datatype properties (relating to address, telephone number etc).As discussed above, this particular model makes no attempt to deal with these relationship other than by simply introducing them.

Particular places (USA, Manchester, EuroDisney) have been introduced as individuals in the model. Although **OilEd** is not intended for large scale knowledge base construction, individuals can be introduced for use in "one-of" expressions. These can then be used in definitions such as a Trip from Europe to America (as discussed below).

Trips and Journeys

Once the basic artefacts of the model were in place (Vehicles and Places), concepts representing travel were introduced. A Trip is the basic atomic unit. Every Trip must have a departure and arrival point. Trips can use at most one vehicle. Trips can be combined into Journeys. Ideally a constraint should be in place that the trips that make up a journey all fit together in a sensible fashion, but this is not currently represented.



Name: Overland Journey

Properties: ○ SubclassOf ● SameClassAs

Documentation: An overland journey is one that consists solely of trips that use land vehicles.

Classes: Journey

Restrictions

type	property	filler
to-class	madeUpOf	[using some Land Vehicle]

Figure 2 Overland Journey

Subclasses of trips can now be defined: for example a Flight is a Trip that uses an Air Vehicle. A Taxi Ride is a trip using a Taxi. Axioms are used to constrain the start and end points of Trips. For example, any Flight must begin and end at an Airport. Similarly, a Sailing (a Trip using a Water Vehicle) must begin and end at a Sea Port.

Further compositional concepts can now be defined. An Overland Journey is one which consists solely of trips using Land Vehicles. A Long Journey is one which involves at least three trips. The expressiveness of DAML+OIL coupled together with the editing facilities of **OilEd** allow us to make these definitions without having to introduce intermediate descriptions. For example, the definitions of Overland Journey as shown in Figure 2 uses a universal restriction whose filler is an existential restriction.

We can now begin to describe additional constraints on the way in which trips can be put together. For example, it is impossible to travel from North America to Europe by train, car, bike or motorbike. This is represented by an axiom which states that any Trip from a place in North America to a place in Europe cannot use a Land Vehicle.

Discussion

Although this was a small scale experiment, and concentrated on a subset of the described domain, there were some interesting aspects. The model took around a day to produce, from examining the NL description to final production of the DAML+OIL model. The classified ontology was also exported as HTML and vanilla RDFS (including the inferred superclasses found by the classifier), and a plot of the hierarchy was produced using output from **OilEd** and dotty[1].

---

[1] http://www.research.att.com/sw/tools/graphviz/

Some Examples & Queries

As discussed above, **OilEd** is not intended for knowledge base construction, and does not directly support queries. However, we can use **OilEd** to define concept expressions which represent queries which might be posed by an application. These can then be classified against the ontology to check their consistency and their relationship with other descriptions – this use of the classifier was the key aspect explored in the experiment.

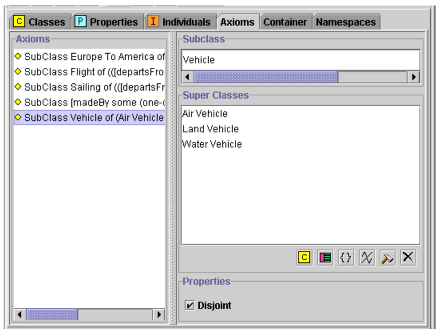
For example, a Non Flyer is defined as somebody who will only take Journeys that are made up of Trips that do not use Air Vehicles. A Non Airbus Traveller is someone who will not take a trip that involves something made by Airbus. An interesting side effect of this definition is the discovery that a Non Flyer is also a Non Airbus Traveller (as an axiom states that Airbus only make aircraft).

Such definitions will be useful for representing the preferences of travellers in an application. In the example model, XX Non Flyer is provided as a specialisation of Non Flyer that takes a trip that includes a Flight. This is spotted as an inconsistency by the classifier.

In a similar vein, the concept XX Europe to America is defined as a Trip from Europe to America using a Bus. This is again spotted as an inconsistency.

Another simple example of an inconsistency is the concept XX Journey, which is defined as being an Overland Journey that includes a Flight.

Round Tripping



Axioms
- SubClass Europe To America of
- SubClass Flight of (([departsFro
- SubClass Sailing of (([departsFr
- SubClass [madeBy some (one-
- SubClass Vehicle of (Air Vehicle

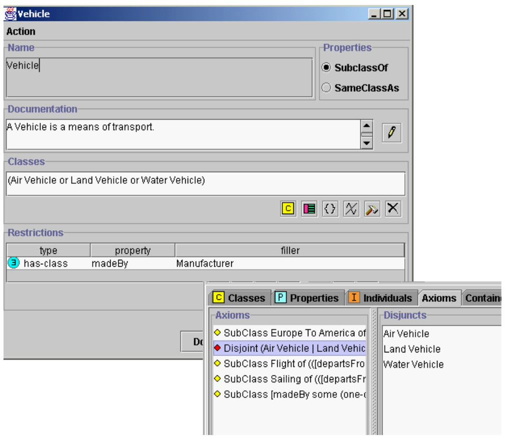
Subclass: Vehicle

Super Classes:
Air Vehicle
Land Vehicle
Water Vehicle

Properties: [✓ Disjoint]

Figure 3 Original Axiom

As discussed in [Bechhofer01a], there can be problems when “round-tripping” from **OilEd** in to DAML+OIL and back again. This was illustrated here by the axioms relating to Vehicles. In the original model, a covering axiom was added stating that Land, Water and Air Vehicles formed a disjoint covering of Vehicle. This is represented in DAML+OIL using a subClass axiom with a disjointUnionOf Class description. However, there is no information in the DAML+OIL that tells us how this was actually presented in the original ontology. When the model is saved and then re-read by the tool, the information is presented as an explicit superclass (of Vehicle) and a disjointness axiom. Of course, the semantics of this ontology are identical to those of the original, but the information is being presented in a different way. This is, admittedly, partly a tools issue – the tool could perhaps do more to try and preserve the information about the so-called *semiotic* information [Euzanet00] in the ontology, through extensions to the RDF used (although this would also require mechanisms for grouping and referring to collections of statements in the RDF model).



Name: Vehicle

Properties: [● SubclassOf] ○ SameClassAs

Documentation: A Vehicle is a means of transport.

Classes: (Air Vehicle or Land Vehicle or Water Vehicle)

Restrictions

type	property	filler
has-class	madeBy	Manufacturer

Axioms
- SubClass Europe To America of
- Disjoint (Air Vehicle \| Land Vehic
- SubClass Flight of (([departsFro
- SubClass Sailing of (([departsFr
- SubClass [madeBy some (one-

Disjuncts:
Air Vehicle
Land Vehicle
Water Vehicle

Figure 4 After Round Tripping

References

[Bechhofer01a] Sean Bechhofer, Carole Goble, Ian Horrocks. *DAML+OIL is not enough.* SWWS-1, Semantic Web working symposium, Stanford (CA), July 29th-August 1st, 2001

[Bechhofer01b] Sean Bechhofer, Ian Horrocks, Carole Goble, Robert Stevens. *OilEd: a Reason-able Ontology Editor for the Semantic Web*. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.

[Euzanet00] Jerome Euzenat. Towards formal knowledge intelligibility at the semiotic level. In *ECAI 2000 Workshop Applied Semiotics: Control Problems, Berlin (DE)*, pages 59–61, 2000.

Travelling Domain Experiment: Engineering with OntoEdit

York Sure
Institute AIFB
University of Karlsruhe
76128 Karlsruhe, Germany
sure@aifb.uni-karlsruhe.de

1 Introduction

This paper presents the results of using **OntoEdit** in the context of the experiment on evaluation of ontology related technologies that was initiated by the Special Interest Group (SIG) on Enterprise-Standard Ontology Tools of the EU IST-2000-29243 thematic network OntoWeb (cf. http://www.ontoweb.org/).

OntoEdit [1,2] is a collaborative ontology engineering environment that has been developed keeping five main objectives in mind: (i) Ease of use. (ii) Methodology-guided [3] development of ontologies. (iii) Ontology development with help of inferencing. (iv) Development of ontology axioms. (v) Extensibility through plugin structure [4].

Modelling ontologies using OntoEdit involves modelling at a conceptual level, viz. as independently of a concrete representation language as possible, and using GUI's representing views on conceptual structures (concepts, concept hierarchy, relations, instances, axioms) rather than codifying conceptual structures in ASCII. The conceptual model of an ontology is stored internally using a powerful ontology model, which can be mapped onto different, concrete representation languages (e.g. RDF(S) or DAML+OIL). As mentioned above, the core functionalities of OntoEdit are easily expandable through a flexible plug-in framework.

In order to provide a clearly defined semantics to the knowledge model of OntoEdit, the knowledge structures of OntoEdit correspond to a well-understood logical framework, viz. F-Logic [5] ("F" stands for "Frames"). F-Logic allows for concise definitions with object oriented-like primitives (classes, attributes, OO-style relations, instances) that fit very nicely with the OntoEdit GUI. Furthermore, it also has PL-1 like primitives (predicates, function symbols). Furthermore, F-Logic allows for axioms that further constrain the interpretation of the model. Axioms may either be used to describe constraints or they may define rules, e.g. in order to define a relation *R* by the composition of two other relations *S* and *Q*. F-Logic rules have the expressive power of Horn-Logic with negation and may be transformed into Horn-Logic rules. Unlike Description Logics (DL), F-Logic does not provide means for subsumption, but (also unlike DL) it provides for efficient reasoning with instances and for the capability to express arbitrary powerful rules, e.g. ones that quantify over the set of classes. Cf. [2] for a more elaborated discussion.

Our inference engine Ontobroker [6] comes with several features that makes it adequate as a backbone for an ontology editor. In particular, it provides: (i) A namespace mechanism: Thus, several ontologies (or ontology parts) may be syntactically split into modules and processed by different inference engines. (ii) Switch-off: It is possible to switch of (possibly singleton) sets of definitions. Thus, one may test interactions and easily distinguish between modules. (iii) DB Connectors: Thus, one may easily map database tables into predicates via JDBC. (iv) User-definable built-Ins: Besides of standard built-ins like "multiply", the user may define his own ones for special purposes. (v) An extensive API: Thus, one may remotely connect to the inference engine and one may also import and export several standards (e.g., RDF(S)).

2 Engineering the Model

For engineering the traveling domain model with OntoEdit we performed the two steps "Kickoff" and "Refinement" of our methodology.

2.1 Kickoff

In the first step, a semi-formal description of the ontology is created by sketching the most relevant elements of the domain. The early stages of ontology development are often driven by brainstorming like knowledge acquisition sessions. In other projects (cf., e.g., [7]) we made good experiences with creating mindmaps as a first draft of relevant elements for a domain. Especially domain experts who were not familiar with modeling preferred using a mindmapping tool instead of directly modeling with an ontology editor. Figure 1 shows the mindmap created from the natural language description of the domain. We rely on a commercial tool for the creation of electronically mindmaps, the MindManager 2002 Business Edition (cf. http://www.mindjet.com/).

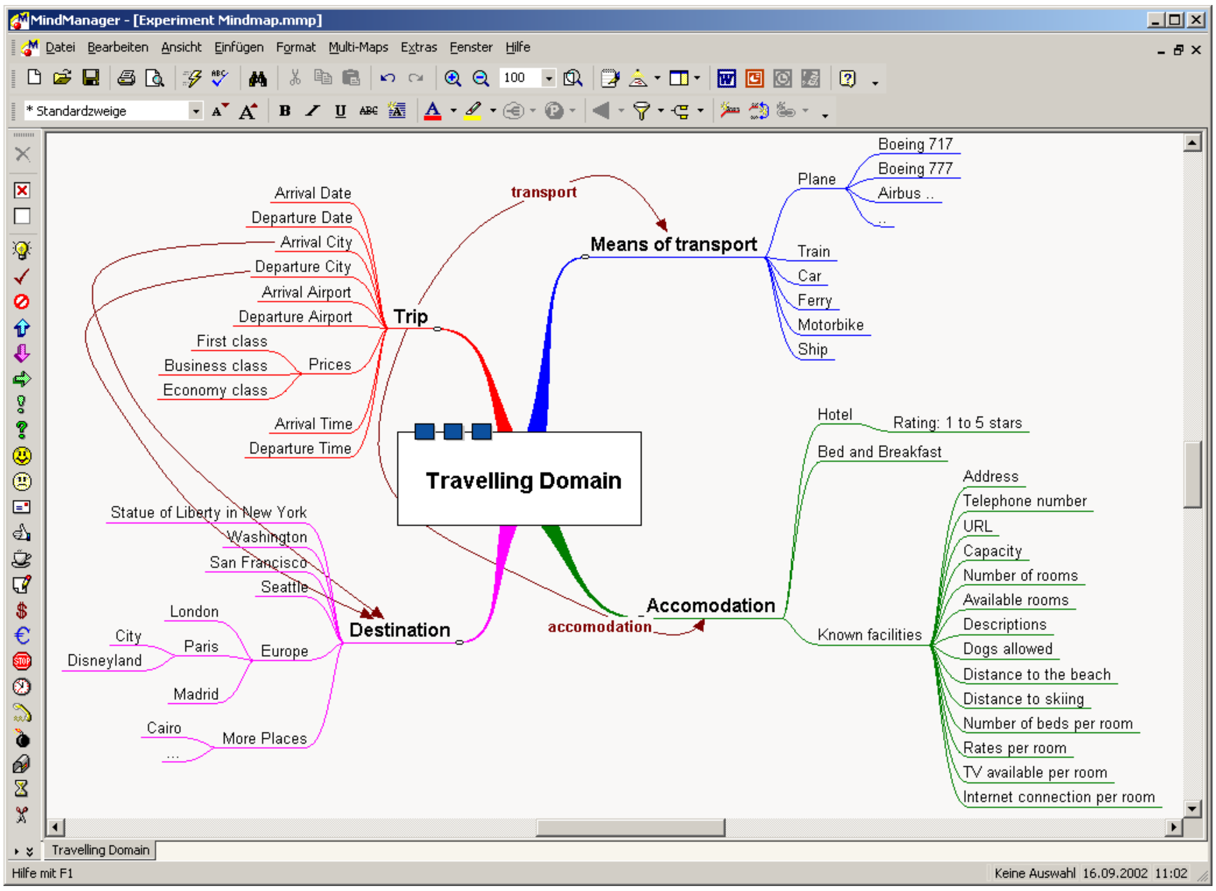
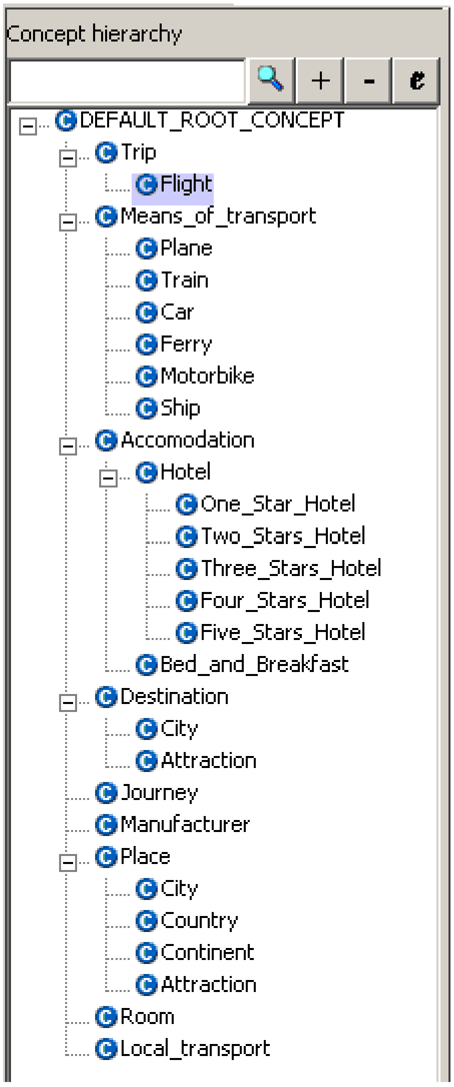


Fig. 1: MindMap of the traveling domain

When collaborating with domain experts the time needed for knowledge acquisition is essential, especially in industrial environments. The advantage of using mindmaps is (i) the quick generation of a graphical representation of relevant domain elements (ii) by using an intuitive and rather well-known tool. The creation of this mindmap took less than 20 minutes.


Concept hierarchy

- DEFAULT_ROOT_CONCEPT
  - Trip
    - Flight
  - Means_of_transport
    - Plane
    - Train
    - Car
    - Ferry
    - Motorbike
    - Ship
  - Accomodation
    - Hotel
      - One_Star_Hotel
      - Two_Stars_Hotel
      - Three_Stars_Hotel
      - Four_Stars_Hotel
      - Five_Stars_Hotel
    - Bed_and_Breakfast
  - Destination
    - City
    - Attraction
  - Journey
  - Manufacturer
  - Place
    - City
    - Country
    - Continent
    - Attraction
  - Room
  - Local_transport

Fig. 2: Concepts

However, when it comes to terms of a formal model of the domain, this representation is no longer suitable. This representation does not clearly distinguish between the notions of concepts, relations etc.. The only semantics for connections (branches or directed edges) in a mindmap is that these elements are "associatively linked". Closer related elements are typically marked with same colors. Typically a mindmap represents the key concepts and their relationships and to formalize it into an ontology, the ontology engineer has now to decide which elements are concepts, how is their hierarchical "is-a" structure and which elements are other named relationships. In some cases one might find prototypical instances, but constraints like the ones given at the end of the domain description are typically not found in mindmaps.

Currently the formalization has still to be "sorted out" manually by ontology engineers. A XML based exchange between the MindManager and OntoEdit guarantees interoperability on a syntactical level. For future versions we plan also to support the decision making during the formalization.

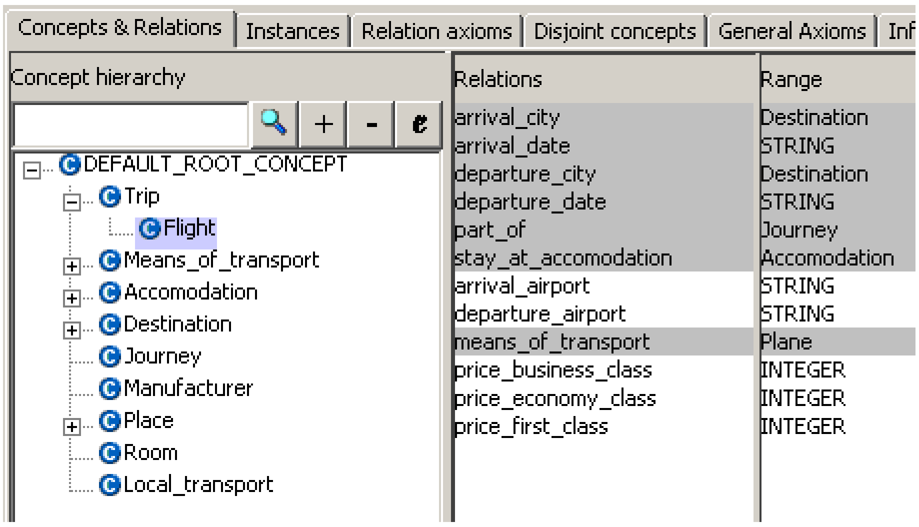
2.2 Refinement

2.2.1 Concepts and relationships

To formalize the mindmap we followed the steps (i) creation of an "is-a" hierarchy of concepts, (ii) adding attributes of concepts and relationships between concepts other then "is-a", (iii) including prototypical instances and (iv) adding axioms that represent constraints and common sense deductions. At several points we needed to introduce further concepts, that were not obvious at first hand from the domain description but necessary to build a complete model. E.g. we introduced a concept `Journey` to combine several trips, some others are mentioned in the text below. This reflects the fact that the mindmap is typically covering only the most relevant elements of a domain, but is not intended to represent more complex relationships in a formally consistent way.

Figure 2 shows the resulting concept hierarchy. When defining a `Trip` we made the following assumption to narrow down multiple possible interpretations in the description: Not only for flights, but for every `Trip` the arrival/departure date, arrival/departure city is known. A `Flight` is a specialization of `Trip` for which additionally the arrival/ departure airport and the prices for first/business/economy class are known. The grey shaded relationships shown in Figure 3 illustrate the inherited relationships for the selected concept `Flight`, i.e. the domain of them is `Trip`. The relationships without shading have as a domain `Flight` itself.

We made some simple assumptions for defining ranges of the relationships: e.g. the dates are coded as `STRING` which directly points to the XML Schema definition for strings (http://www.w3.org/2001/XMLSchema#String) – same holds e.g. for prices/`INTEGER`.

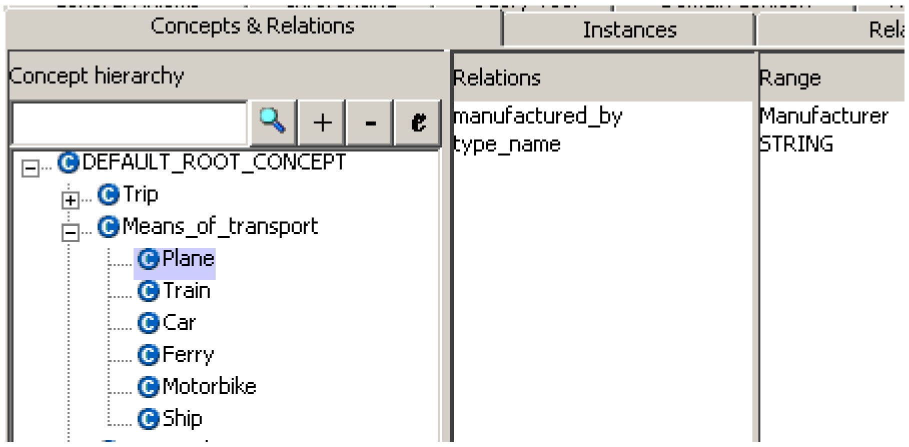


- DEFAULT_ROOT_CONCEPT
  - Trip
    - Flight
  - Means_of_transport
  - Accomodation
  - Destination
  - Journey
  - Manufacturer
  - Place
  - Room
  - Local_transport

Relations	Range
arrival_city	Destination
arrival_date	STRING
departure_city	Destination
departure_date	STRING
part_of	Journey
stay_at_accomodation	Accomodation
arrival_airport	STRING
departure_airport	STRING
means_of_transport	Plane
price_business_class	INTEGER
price_economy_class	INTEGER
price_first_class	INTEGER

Fig. 3: Relationships of „Flight“

The relationship `means_of_transport` is firstly defined for `Flight` with the range `Means_of_transport`. We then refined it for `Flight` by specializing the range to `Plane` that is a subconcept of `Means_of_transport` (cf. Figure 4).

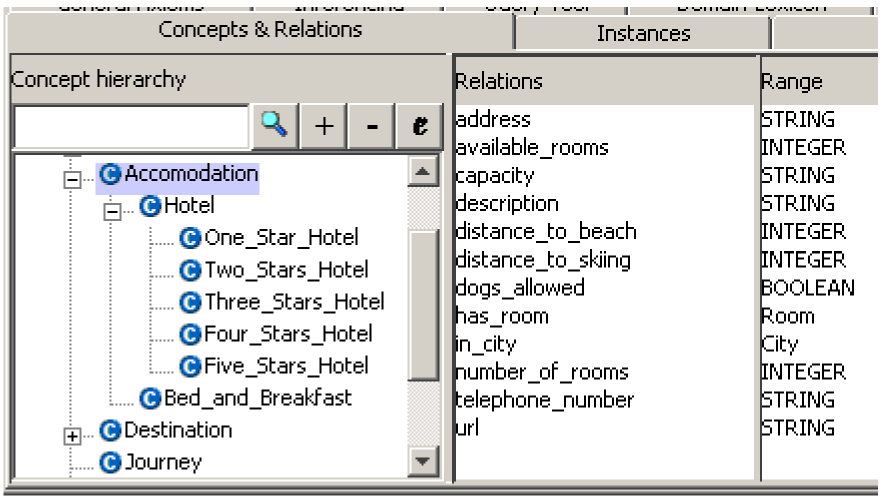


- DEFAULT_ROOT_CONCEPT
  - Trip
  - Means_of_transport
    - Plane
    - Train
    - Car
    - Ferry
    - Motorbike
    - Ship

Relations	Range
manufactured_by	Manufacturer
type_name	STRING

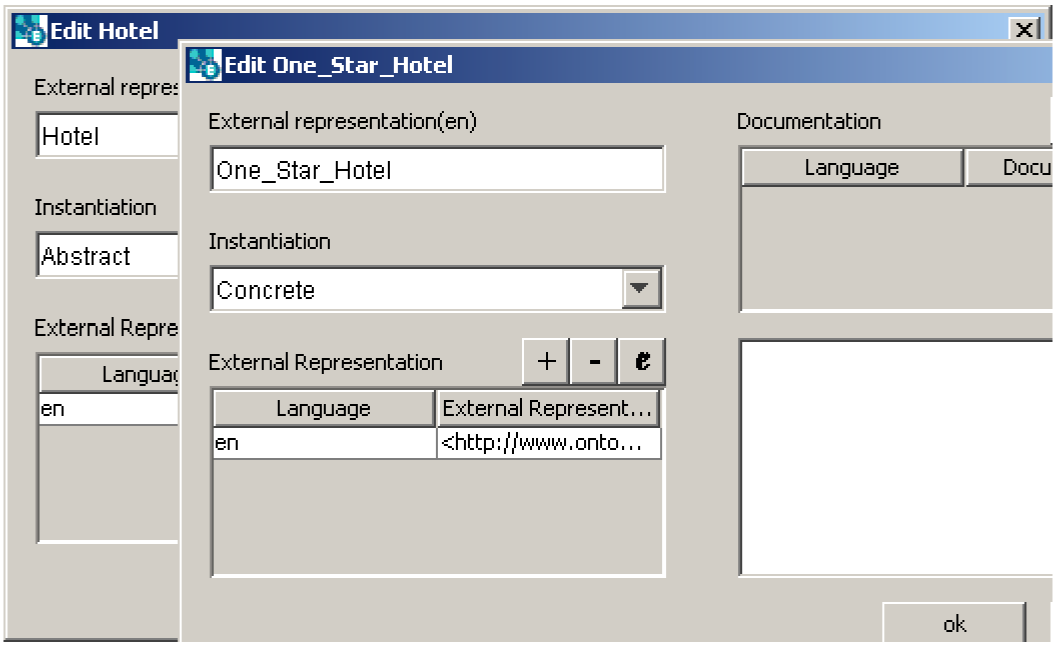
Fig. 4: Relationships for „Plane“

`Accomodation` (cf. Figure 5) has subconcepts `Hotel` and `Bed and Breakfast`. Beside relationships for the facilities mentioned in the description (address, available rooms etc. ..) one can see relationships to `Room` and `City`. To model the star ranking schema for hotels we added further specializations of `Hotel`, e.g. `One-Star-Hotel` (see later in the subsection about instances how we model a particular hotel as an instance). To model that each hotel belongs to one star category, we defined `Hotel` as an “abstract” concept, i.e. there are no instances of this concept allowed, and all subconcepts like `One Star Hotel` as “concrete” concepts (cf. Figure 6). Same holds e.g. for `Means of transport` and its subconcepts.



Relations	Range
address	STRING
available_rooms	INTEGER
capacity	STRING
description	STRING
distance_to_beach	INTEGER
distance_to_skiing	INTEGER
dogs_allowed	BOOLEAN
has_room	Room
in_city	City
number_of_rooms	INTEGER
telephone_number	STRING
url	STRING

Fig. 5.: Relationships of „Accomodation“



External representation(en): One_Star_Hotel

Instantiation: Concrete

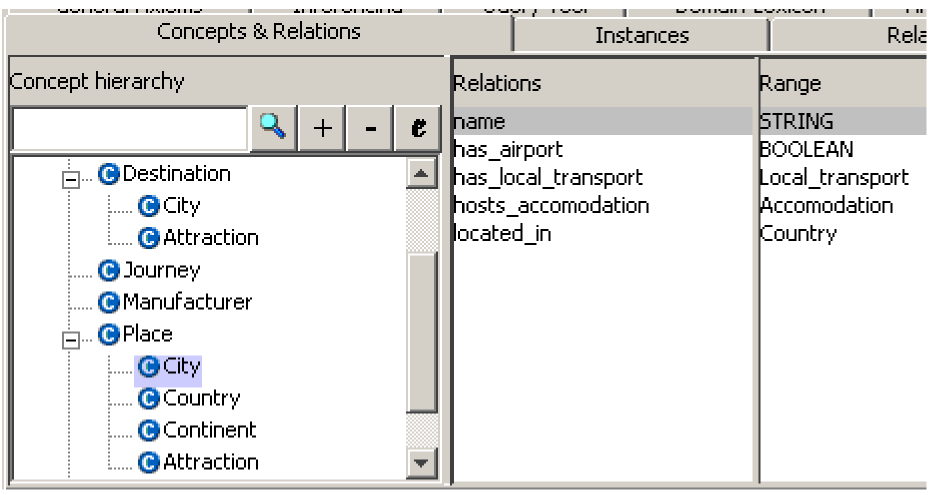
External Representation

Language	External Represent...
en	<http://www.onto...

Fig. 6: “Abstract” vs. “concrete” concepts

We introduced `Place` as a superconcept of `City` (cf. Figure 7). For further axioms on top we also included `Attraction`, `Country` and `Continent`. The concepts are related via the `located in` relationship, e.g. an `Attraction` is `located in` a `City`, a `City` is `located in` a `Country` and a `Country` is located in a `Continent`. As shown later this relationship is transitive. `City` and `Attraction` are also subconcepts of `Destination`, i.e. they are multiply inherited. Alternatively one could consider to add `Destination` as a subconcept of `Place`, too. In the current scenario that would have no effect. Modeling it this way seemed more intuitive to us.

Last but not least, a `Journey` has potentially many parts, i.e. it can be related via `has part` to many instances of `Trip` that belong to this `Journey`. For completeness we included also the inverse relationship part of for `Trip` with the range `Journey` and defined these two relationships as invers (see later subsection on axioms).


Concepts & Relations | Instances | Rela

Concept hierarchy

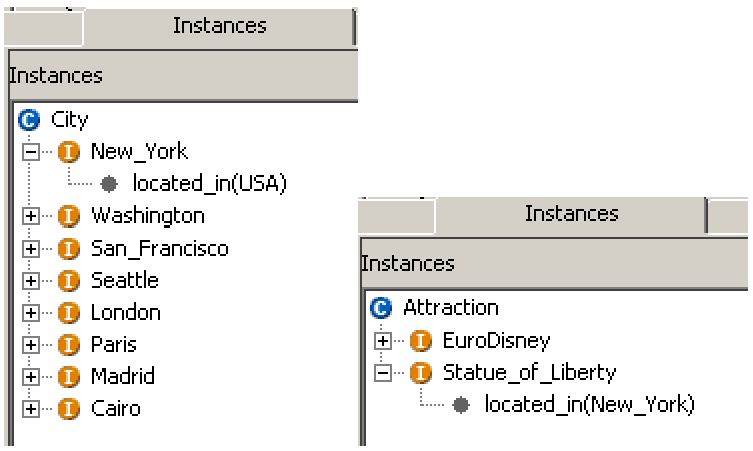
- Destination
  - City
  - Attraction
- Journey
- Manufacturer
- Place
  - City
  - Country
  - Continent
  - Attraction

Relations	Range
name	STRING
has_airport	BOOLEAN
has_local_transport	Local_transport
hosts_accomodation	Accomodation
located_in	Country

Fig. 7 : Relationships of "City"

2.2.2 Instances

We modeled several instances, e.g. shown in Figures 8, 9 and 10. Part of them are given in the first part of the description (e.g. the cities `New York`, `Washington` etc. and the attractions `Statue of Liberty` and `EuroDisney`) , others are given in the last section with an example journey for John (cf. Figure 10). John makes two flights (from `Madrid to NY` and from `Washington to Madrid`) and one trip with a motorcycle (from `NY to Washington`).

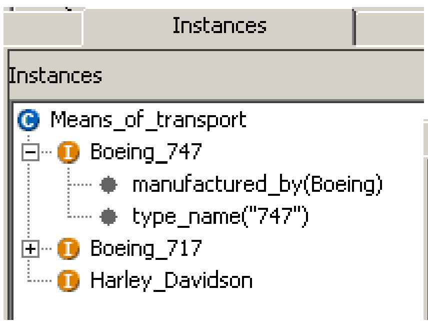

Instances

- City
  - New_York
    - located_in(USA)
  - Washington
  - San_Francisco
  - Seattle
  - London
  - Paris
  - Madrid
  - Cairo

Instances

- Attraction
  - EuroDisney
  - Statue_of_Liberty
    - located_in(New_York)

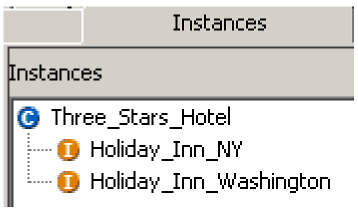
Fig. 8 : Instances of "City" and "Attraction"



Instances

Instances

Means_of_transport
- Boeing_747
  - manufactured_by(Boeing)
  - type_name("747")
- Boeing_717
- Harley_Davidson

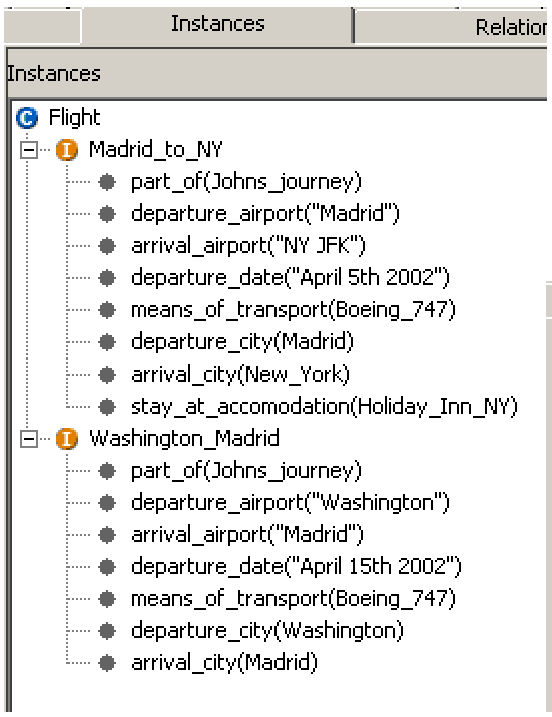


Instances

Instances

Three_Stars_Hotel
- Holiday_Inn_NY
- Holiday_Inn_Washington

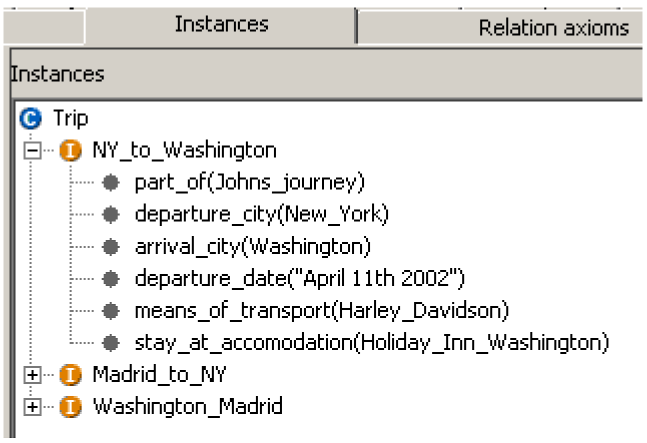
Fig. 9: Instances of "Means of transport" and "Three Stars Hotel"



Instances | Relatior

Instances

Flight
- Madrid_to_NY
  - part_of(Johns_journey)
  - departure_airport("Madrid")
  - arrival_airport("NY JFK")
  - departure_date("April 5th 2002")
  - means_of_transport(Boeing_747)
  - departure_city(Madrid)
  - arrival_city(New_York)
  - stay_at_accomodation(Holiday_Inn_NY)
- Washington_Madrid
  - part_of(Johns_journey)
  - departure_airport("Washington")
  - arrival_airport("Madrid")
  - departure_date("April 15th 2002")
  - means_of_transport(Boeing_747)
  - departure_city(Washington)
  - arrival_city(Madrid)



Instances | Relation axioms

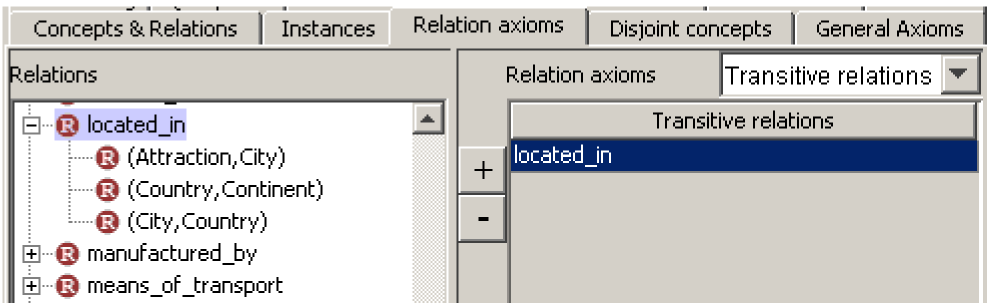
Instances

Trip
- NY_to_Washington
  - part_of(Johns_journey)
  - departure_city(New_York)
  - arrival_city(Washington)
  - departure_date("April 11th 2002")
  - means_of_transport(Harley_Davidson)
  - stay_at_accomodation(Holiday_Inn_Washington)
- Madrid_to_NY
- Washington_Madrid

Fig. 10: Instances of "Flight" and "Trip"

2.2.3 Axioms

On top we defined several axioms: `located in` is transitive (cf. Figure 11), e.g. `has part` is inverse to `part of` (cf. Figure 12), the subconcepts of `Hotel` are pairwise disjoint (cf. Figure 13).



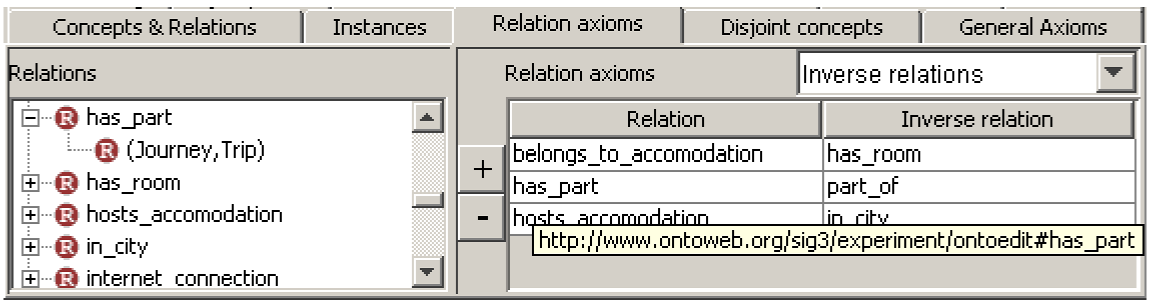
Concepts & Relations | Instances | Relation axioms | Disjoint concepts | General Axioms

Relations
- located_in
  - (Attraction,City)
  - (Country,Continent)
  - (City,Country)
- manufactured_by
- means_of_transport

Relation axioms: Transitive relations

Transitive relations
located_in

Fig. 11: Transitive relationship

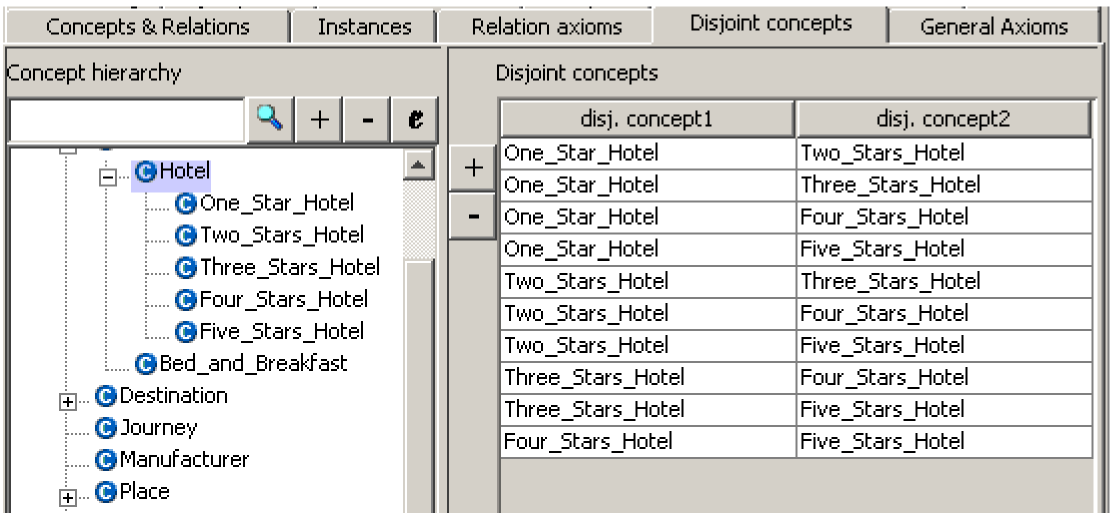


- has_part
  - (Journey,Trip)
- has_room
- hosts_accomodation
- in_city
- internet_connection

Relation	Inverse relation
belongs_to_accomodation	has_room
has_part	part_of
hosts_accomodation	in_city

http://www.ontoweb.org/sig3/experiment/ontoedit#has_part

Fig. 12: Inverse relationships



- Hotel
  - One_Star_Hotel
  - Two_Stars_Hotel
  - Three_Stars_Hotel
  - Four_Stars_Hotel
  - Five_Stars_Hotel
  - Bed_and_Breakfast
- Destination
- Journey
- Manufacturer
- Place

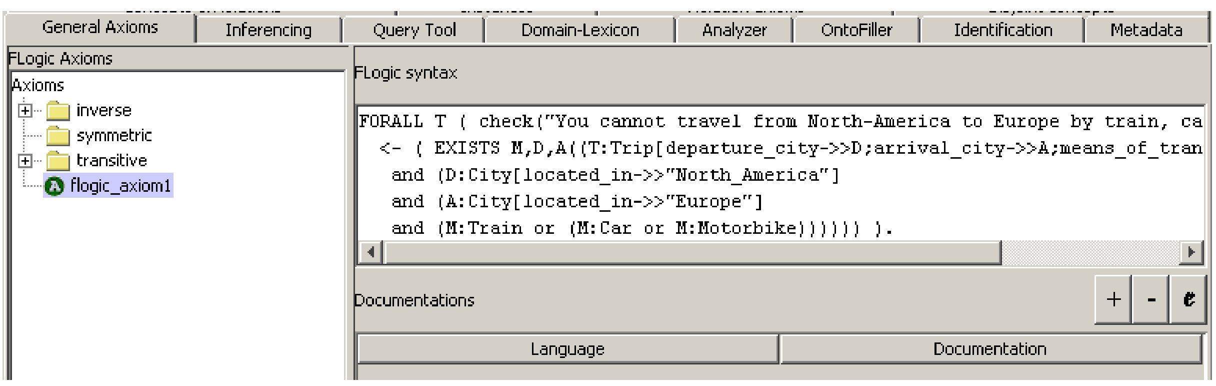
disj. concept1	disj. concept2
One_Star_Hotel	Two_Stars_Hotel
One_Star_Hotel	Three_Stars_Hotel
One_Star_Hotel	Four_Stars_Hotel
One_Star_Hotel	Five_Stars_Hotel
Two_Stars_Hotel	Three_Stars_Hotel
Two_Stars_Hotel	Four_Stars_Hotel
Two_Stars_Hotel	Five_Stars_Hotel
Three_Stars_Hotel	Four_Stars_Hotel
Three_Stars_Hotel	Five_Stars_Hotel
Four_Stars_Hotel	Five_Stars_Hotel

Fig. 13: Disjoint concepts

A more complex axiom is given in the description by “it is not possible to go from America to Europe by train, car, bike or motorbike” (without restricting the generality we excluded bike because it was not given in the previous section for means of transport). We defined a general axiom in F-Logic that can be used to check whether this constraint holds for all given instances (see also Figure 14):

```
FORALL T check("You cannot travel from North-America to Europe
by train, car or motorbike!",T)
     <- EXISTS M,D,A
     T:Trip[departure_city->>D;
            arrival_city->>A;
            means_of_transport->>M]
     AND D:City[located_in->>"North_America"]
     AND A:City[located_in->>"Europe"]
     AND (M:Train OR M:Car OR M:Motorbike).
```

Other given constraints can be formalized similar to this. To perform a check we simply query for all values of the 2-ary predicate `check`.

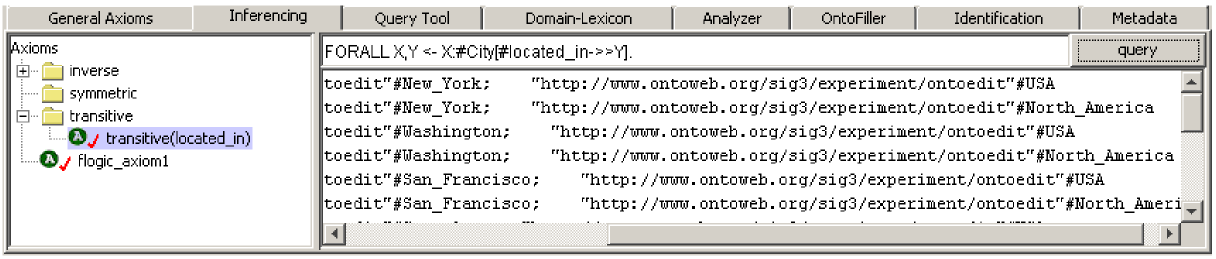


```
FORALL T ( check("You cannot travel from North-America to Europe by train, ca
  <- ( EXISTS M,D,A((T:Trip[departure_city->>D;arrival_city->>A;means_of_tran
   and (D:City[located_in->>"North_America"]
   and (A:City[located_in->>"Europe"]
   and (M:Train or (M:Car or M:Motorbike)))))) ).
```

Fig. 14: General axiom

2.2.4 Inferencing

OntoEdit (Inferencing Edition) can be connected to the inferenced engine Ontobroker. We are thereby able to perform queries for concepts, relationships, instances etc.. E.g. we can ask for all cities and where they are `located in`. If we enable the axiom for transitivity of the relationship `located in` (like shown in Figure 15) we receive as an answer to that query that e.g. `New York` is located in `USA` (an instance of `Country`) as well as the fact the `New York` is located in `North America` (an instance of `Continent`).



```
FORALL X,Y <- X:#City[#located_in->>Y].
```

```
toedit"#New_York;    "http://www.ontoweb.org/sig3/experiment/ontoedit"#USA
toedit"#New_York;    "http://www.ontoweb.org/sig3/experiment/ontoedit"#North_America
toedit"#Washington;    "http://www.ontoweb.org/sig3/experiment/ontoedit"#USA
toedit"#Washington;    "http://www.ontoweb.org/sig3/experiment/ontoedit"#North_America
toedit"#San_Francisco;    "http://www.ontoweb.org/sig3/experiment/ontoedit"#USA
toedit"#San_Francisco;    "http://www.ontoweb.org/sig3/experiment/ontoedit"#North_Ameri
```

Fig. 15: Inferencing in OntoEdit

3 Conclusion

We illustrated the modeling process for engineering the traveling domain with OntoEdit. We were able to formalize the given natural language description. Our aim was to model the domain as close as possible to the given domain description, i.e. we tried to add only those concepts and relationships that were mentioned. On top we defined axioms that reflect constraints given in the description.

4 References

[1] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer and D. Wenke. **OntoEdit: Collaborative Ontology Engineering for the Semantic Web.** In: *Proceedings of the first International Semantic Web Conference 2002* (ISWC 2002), June 9-12 2002, Sardinia, Italia, Springer, LNCS 2342, pages 221-235.

[2] Y. Sure, S. Staab, J. Angele. **OntoEdit: Guiding Ontology Development by Methodology and Inferencing.** In: *Proceedings of the International Conference on Ontologies, Databases and Applications of SEmantics* (ODBASE 2002), October 28 - November 1, 2002, University of California, Irvine, USA, Springer, LNCS.

[3] S. Staab, H.-P. Schnurr, R. Studer, and Y. Sure: **Knowledge Processes and Ontologies.** In: *IEEE Intelligent Systems* 16(1), January/Febrary 2001, Special Issue on Knowledge Management.

[4] Siegfried Handschuh. **Ontoplugins –a flexible component framework.** Technical report, University of Karlsruhe, May 2001.

[5] M. Kifer, G. Lausen, and J. Wu. **Logical foundations of object-oriented and frame-based languages.** *Journal of the ACM*, 42:741–843, 1995.

[6] S. Decker, M. Erdmann, D. Fensel, and R. Studer. **Ontobroker: Ontology based access to distributed and semi-structured information.** In: R. Meersman et al., editor, *Database Semantics: Semantic Issues in Multimedia System*s. Kluwer Academic, 1999.

[7] Th. Lau and Y. Sure. **Introducing Ontology-based Skills Management at a large Insurance Company.** In: *Proceedings of the Modellierung 2002, Modellierung in der Praxis - Modellierung für die Praxis*, Tutzing, Deutschland, 25.-27. März 2002.

Travelling Domain Experiment: Results for Loom

Aldo Gangemi
Research Scientist
Ontology and Conceptual Modeling Group
ISTC-CNR (Istituto di Scienze e Tecnologie della Cognizione)
Institute of Cognitive Sciences and Technologies, C.N.R.
Viale Marx 15, 00137
Roma Italy
gangemi@acm.org

For this experiment, we have tried to apply OntoClean and ONIONS methodologies, by analysing the intended meaning of the requirements provided in the natural language description of the domain with a basic linguistic and cognitive bias. This results into merging the domain ontology with the foundational concepts and relations provided by an early version of DOLCE (cf. WonderWeb EU Project Deliverable D17, http://www.wonderweb.semanticweb.org).

Not all axioms of DOLCE have been explicitly imported in the domain ontology, both for expressivity lack, for efficiency reasons (the requirements seem to address an application rather than a reference domain ontology), and because the experiment is a quick and preliminary one.

We have used the Loom KRS that implements a very expressive DL with important extra-logical features, but has a drawback consisting in having an incomplete reasoner. Another problem with Loom has been the unavailability of a Loom to RDF translator. We provide here (besides the Loom code) a KIF translation as well. A possible path to RDF could be translating Loom into FaCT DL and then feeding OILed tool with the FaCT code and obtaining a DAML+OIL version. This and other solutions can be tried in a short time.

We tried to avoid overloading the ontology with unnecessary axioms, then we add axioms and taxonomical fillings only when a clarification has to made concerning the soundness of the domain ontology compared to the feature of DOLCE foundational ontology.

A more extensive description and evaluation of the experiment will be provided at the WKS.

The domain ontology currently consists of:

86 (domain) concepts
38 (domain) relations
33 (domain) individuals
7 (domain) rules

Axiomatizing relations has been the most idiosyncratic part of the work: in order to anticipate (and prevent) the low expressivity of the final requested RDF translation, which does not support role chaining and n-ary relations, we have provided analytical definitions of complex domain relations by using the :compose Loom construct. This allows the low-end version of the ontology to maintain the conceptual map developed by means of rigorous methodologies; in other words, an application-oriented version of the ontology can be used without loosing the history and motivation of its development that are computable (not at runtime) through the reference version of it.

For bridging some gaps between DOLCE and domain ontology, we have used a refined version of WordNet hyperonymy hierarchies. Here we include the four taxonomies obtained by navigating the four basic branchings of DOLCE down to the domain level.

In the domain ontology the meta-property assignment is still lacking.

Object branching hierarchy:

```
OBJECT
: ABSTRACT-OBJECT
: : MENTAL-OBJECT
: : SOCIAL-OBJECT
: : : COUNTRY
: : : GROUP
: : : INFORMATION
: : : : EXPRESSION
: : : : : MEASUREMENT-UNIT
: : : : Plan
: : : : STATEMENT
: : : : TEXT
: : : PERSON
: : : : NATURAL-PERSON
: : : : : CLIENT
: : : : SOCIALLY-CONSTRUCTED-PERSON
: : : : : LEGALLY-CONSTRUCTED-PERSON
: : : : : : ORGANIZATION
: : : : : : : ENTEPRISE
: : : : : : : ENTERPRISE
: : : : : : : : FLIGHT-COMPANY
: : : : : : : : HOTEL-CHAIN
: : : : : : : : TRAVEL-AGENCY
: : : SERVICE-PROVIDER
: PHYSICAL-OBJECT
: : BODY
: : ORDINARY-OBJECT
: : : ARTIFACT
: : : : CONSTRUCTION
: : : : DEVICE
: : : : FACILITY
: : : : : AIRPORT
: : : : INSTRUMENT
: : : : : EQUIPMENT
: : : : : : ELECTRONIC-EQUIPMENT
: : : : : : : RECEIVER
: : : : : : : : TV
: : : : : FURNITURE
: : : : : : BED
: : : : : INTERNET-CONNECTION
: : : : : MEAN-OF-TRANSPORT
: : : : : : BIKE
: : : : : : CAR
: : : : : : : RENTAL-CAR
: : : : : : : TAXI
: : : : : : CITY-TRANSPORT
: : : : : : : CITY-BUS
: : : : : : : RENTAL-CAR
: : : : : : : TAXI
: : : : : : : UNDERGROUND
: : : : : : FERRY
: : : : : : MOTORBIKE
: : : : : : PLANE
: : : : : : : AIRBUS-PLANE
: : : : : : : BOEING-PLANE
: : : : : : : : BOEING-717
: : : : : : : : BOEING-777
: : : : : : SHIP
: : : : : : TRAIN
: : : : STRUCTURE
: : : : : HOUSING
: : : : : : LIVING-QUARTERS
: : : : : : : ACCOMMODATION
: : : : : : : : B&B
: : : : : : : : HOTEL
: : : : : : : : : FIVE-STAR-HOTEL
: : : : : : : : : FOUR-STAR-HOTEL
: : : : : : : : : ONE-STAR-HOTEL
: : : : : : : : : THREE-STAR-HOTEL
: : : : : : : : : TWO-STAR-HOTEL
: : : : : ROOM
: : : : : : HOTEL-ROOM
: : : : VEHICLE
```

Occurrence branching hierarchy:

```
OCCURRENCE
: ACCOMPLISHMENT
: : Action
: : : ACTIVITY
: : : : ARRIVAL
: : : : DEPARTURE
: : : : LODGING
: : : : REQUEST
: : : : TRAVEL
: : : : : CITY-TRAVEL
: : : : : EUROPE-AMERICA-TRAVEL
: : : : : FLIGHT
: : : : : : ONE-WAY-FLIGHT
: : : : : : OUTWARD-FLIGHT
: : : : : : RETURN-FLIGHT
: : : : : : ROUND-TRIP-FLIGHT
: : : : : OVERNIGHT-TRAVEL
: : : : WORK
: : : : : SERVICE
: : : : : : FLIGHT
: : : : : : : ONE-WAY-FLIGHT
: : : : : : : OUTWARD-FLIGHT
: : : : : : : RETURN-FLIGHT
: : : : : : : ROUND-TRIP-FLIGHT
: : : : : : FLIGHT-SERVICE
: : : : : : HOSTING
: : : TRANSACTION
: ACHIEVEMENT
: FLUX
: NON-RELATIONAL-OCCURRENCE
: PHENOMENON
: PROCESS
: RELATIONAL-OCCURRENCE
: STATE
: : COGNITIVE-STATE
: : : EMOTION
: : : : HATE
```

Quality branching hierarchy:

```
QUALITY
: ABSTRACT-QUALITY
: : ACCOMMODATION-QUALITY
: : ABSTRACT-LOCATION
: : : URL
: PHYSICAL-QUALITY
: : COLOR
: : DISTANCE
: : SHAPE
: : SPATIAL-LOCATION
: : : ADDRESS
: : : GEOGRAPHIC-AREA
: : : : BEACH
: : : : CITY
: : : : CONTINENT
: : : : NATION
: : : : RESORT
: : : : SKIING-AREA
: : VOLUME
: TEMPORAL-QUALITY
: : DATE
: : INTERVAL
: : TEMPORAL-LOCATION
```

Quality Space branching hierarchy:

QUALITY-SPACE
: ABSTRACT-QUALITY-SPACE
: : BOOLEAN
: DIMENSION
: PHYSICAL-QUALITY-SPACE
: : COLOR-SPACE
: : SHAPE-SPACE
: : SPATIAL-REGION
: : : GEOGRAPHIC-SPACE
: : VOLUME-SPACE
: QUALE
: : ABSTRACT-VALUE
: : : PRICE-VALUE
: : : : BUSINESS-CLASS-PRICE-VALUE
: : : : ECONOMY-CLASS-PRICE-VALUE
: : : : FIRST-CLASS-PRICE-VALUE
: : SPATIAL-VALUE
: : : METRIC-VALUE
: : TEMPORAL-VALUE
: : : DATE-VALUE
: : : TIME-VALUE
: TEMPORAL-QUALITY-SPACE
: : TEMPORAL-REGION

OntoWeb Travelling Domain Experiment

Results for *Open*KnoME

Dr Jeremy Rogers
Medical Informatics Group
University of Manchester
United Kingdom
jeremy@opengalen.org

Introduction

OpenKnoME [1] is a client software environment for :

- primary authoring of GRAIL ontologies in a collaborative, distributed multi-author setting
- linking GRAIL ontologies to external data sources
- rapid prototyping of subdomain and task specific ontological schemas (intermediate representations) that are systematic simplifications of a more complex, shared, common ontology [2]

*Open*KnoME, written in Visualworks Smalltalk, uses GRAIL formalism reasoners, which are instantiated as separate server applications. The direct descendent of the original GRAIL development and prototyping environment, *Open*KnoME is the product of more than a decade's research into large scale collaborative ontology building, maintenance and delivery, centred on what is now the *Open*GALEN Common Reference Model of medicine (CRM). *Open*KnoME is available under open source license for non-commercial use only from www.topthing.com.

GRAIL is a relatively old formalism , related to more modern and mainstream description logics. It includes many common DL constructors including existential role restriction, role hierarchies and role transitivity. The language is declarative, and compilation is statement order dependent. The compiled model, however, contains relationships between concepts that were not explicitly stated in the sources but were inferred during compilation.

GRAIL does not support true negation or disjunction, has no notion of instances, no mechanism to declare siblings as either truly disjoint or as covering the domain, and only a very limited implementation of cardinality. GRAIL does not provide native support for specific data types (such as dates, or numerical ranges).

Additionally, however, GRAIL *does* include a role inheritance constructor - also known as refinement or specialisation – that is not supported by any current description logic.

GRAIL models, like any compositional ontology, are inherently dynamic: they are typically constructed as a collection of separate mini-ontologies that can be combined, subject to permissions and constraints also in the model, to build detailed concepts that form the target ontology. However, the terms in the target ontology can not be exhaustively defined explicitly, but instead are an implied conceptual space. It is neither sensible nor possible to pre-enumerate all possible members of the implied ontological space because the number of possible constructs in that space rapidly rises to many billions. For this reason a static dump of a 'fully populated' travel model is not provided as part of this experiment.

Building the Model

Generic Model

The OpenGALEN generic model (an upper ontology) [3] was used as the starting point. This ontology provides an off-the-shelf re-usable, relatively domain independent, upper ontology comprising structures, process and substance together with a rich library of semantic links and associated coherent transit and GRAIL role inheritance rules. This upper ontology was developed as a component of the *Open*GALEN CRM ontology. For this evaluation the upper ontology was pruned slightly to remove a small surviving residual of more clinically oriented content.

A new source file project manager was created within the OpenKnoME, and a copy of the OpenGALEN generic model sources was imported.

New Category Space

Primary knowledge acquisition involved reading the textual scenario description to extract candidates for new elementary classes and semantic link types.

Modes of transport

An ontology of vehicles was constructed, primary classification being by whether the vehicle travels on land, water or through the air. I noted that the scenario description stated that no other forms of transport existed other than planes, trains, cars, ferries, motorbikes and ships. However, the same scenario description later mentions buses and underground transportation. A more detailed and general ontology of vehicles was therefore sketched. Additional and atypical forms of vehicle were considered, such as amphibious vehicles, seaplanes, flying cars, gliders, hot air balloons, water taxis, helicopters, trams, bicycles and rickshaws. These implied further classification of vehicles by mode of propulsion.

The next step was to attempt to construct a common parent of all modes of transport. For this, other non-vehicular forms of transportation were also considered, such as horses and camels. From this it became clear that the concept ‘mode of transport’ would necessarily be a fairly abstract notion, subsuming some built structures (vehicles) as well as some organisms.

The concept of the process of transporting was modelled. It may be further described by the physical mode of transport used, by subprocesses of arriving and departing (each of which may be further described by a time and a location), and by a goal (for example, visiting a specific attraction or a city). The more abstract idea of a travel itinerary, comprising any number of transports and hirings of rooms was next modelled. No attempt was made to model any temporal ordering of the various journeys and hotel bookings, as temporal reasoning is not directly supported within GRAIL.

A list of companies was declared, and the processes of hiring or manufacturing created. Airliners were permitted to be characterised by who made them, and independently to take any model type designation. Constraints were entered to say that any airliner with an specific model type designation must have been manufactured by the appropriate manufacturer, and that any airliner made by a specific manufacturer can only have a model type appropriate to that manufacturer.

Trade and Commerce

A small model of trade and commerce was built, in which relationship are declared between [Hiring] as an act, [Price] and [Currency].

Accomodation

Hotels were modelled to have rooms as discrete components. Rooms have a number of further properties (whether they contain a TV, minibar etc), and can be hired from a specified set of companies. Hotels themselves have a set of characteristics that would normally be populated by free text or numerical data. These are included in this model by way of illustration only. A more appropriate mechanism to store instance information (physical address, URL etc) would be in an external database. The role of ontologies in indexing actual products has previously been explored using this toolset in the UK Drug Product Ontology project.

Geography

A list of cities, continents, countries and locations (museums, airport, beaches) was declared as primitives. The inclusion of apparent instances such as ‘Paris’ and ‘Tacoma Airport’ within a framework that does not support instances is explained by the fact that, within this model, the identifier ‘Paris’ represents that class of all possible Paris’s, of which subclasses might include ‘Paris on Bastille Day’, ‘Paris in the Spring’.

The partitive relationships between these geographic loci were modelled using standard Winston-Odell partonomic relationships, which are provided in the upper ontology.

Pragmatics of travel

The scenario asked that we attempt to model some real world constraints, such as that it is not possible to cross the Atlantic by car, or that while it might be possible to travel from Vladivostok to Johanesburg by car, nobody would want to. A very small subset of all the constraints that might be required to constrain the model to allow only physically plausible modes of travel between specified locations were expressed for illustrative purposes only. The constraints are in the form of ‘a journey between locations of a particular type can only be made using specified modes of transport’, and one result is that an attempt to form the concept of a journey from New York to Cairo by bus will fail.

Examples:

The class of trips, of which John’s would be an instance, is expressed as a single expression:

```
TravelItinerary which <
        hasStructuralComponent (Flying which <
                hasSpecificSubprocess (Arriving which <
                        hasSpecificLocation JFK
                        occursDuring April05>)
                hasSpecificSubprocess (Departing which <
                        hasSpecificLocation Madrid
                        occursDuring April05>)
                hasGoal (Visiting which actsOn StatueOfLiberty)>)
        hasStructuralComponent (Flying which <
                hasSpecificSubprocess (Arriving which <
                    hasSpecificLocation DullesAirport
                    occursDuring April11>)
            hasSpecificSubprocess (Departing which <
                    hasSpecificLocation Madrid
                    occursDuring April11>)>)
    hasStructuralComponent (Flying which <
            hasSpecificSubprocess (Arriving which <
                    hasSpecificLocation Madrid
                    occursDuring April15>)
            hasSpecificSubprocess (Departing which <
                    hasSpecificLocation DullesAirport
                    occursDuring April15>)>)
    hasStructuralComponent (Hiring which <
            hasSpecificPersonPerforming HolidayInn
            actsSpecificallyOn (Room which isStructuralComponentOf
                    (Hotel which hasSpecificLocation JFK))>)
    hasStructuralComponent (Hiring which <
            hasSpecificPersonPerforming HolidayInn
            actsSpecificallyOn (Room which isStructuralComponentOf
                    (Hotel which hasSpecificLocation DullesAirport))>)>
```

..and this is classified automatically under TransatlanticTrip and 'TripToVisitNewYork', even though the USA locations are only identified as airports or monuments.

The complex graph description (above) can be entered as a single expression, within a normal ASCII text editor. It is not a requirement that it be assembled from smaller subgraphs that are declared, evaluated and named separately *a priori* before the larger composition can be expressed.

User Interface

The OpenKnoME includes prototype tools to dynamically construct structured data entry interfaces based on the ontology. A screen shot demonstrates a form generated on the topic of a trip, or transport event, partially completed and showing the sub-form produced in response by the user to describe the departure in more detail.

References

1. Rogers J.E., Roberts A., Solomon W.D., van der Haring E, Wroe C.J., Zanstra P.E., Rector, A.L. (2001) GALEN Ten Years On: Tasks and Supporting tools Proceedings of MEDINFO2001, V. Patel et al. (Eds) IOS Press;: 256-260
2. Solomon W.D., Wroe C.J., Rector, A.L., Rogers J.E., Fistein J.L., Johnson P. (1999) A Reference Terminology for Drugs Annual Fall Symposium of American Medical Informatics Association, Washington DC. Hanley & Belfus Inc. Philadelphia PA;:152-155
3. www.opengalen.org/open/crm

The OntoWeb Evaluation Experiment for Ontology Editors: Using Protégé-2000 to Represent the Travel Domain

Natalya F. Noy
Stanford Medical Informatics, Stanford University
noy@smi.stanford.edu

1 Design decisions

Our goal was to represent the domain as close to its natural-language description as possible. We tried to introduce only the concepts and relations that were necessary and sufficient to represent the facts in the description. We did not add any other common-sense facts about the domain.

1.1 Assumption that we have made in interpreting the domain description

Several facts in the description allowed more than one interpretation. Here is a list of assumptions that we have made:

- Prices for flights (business and economy class) are constant for a particular flight and do not change from day to day. In other words, it always costs the same to fly from Madrid to New York on the flight number UA345
- Other trips (not just flights) also have arrival and departure city, arrival and departure time, port (airport, station) of arrival/departure, etc.
- The description said: "We know that each model of transport belongs only to one kind of transportation (e.g., it's either a plane, or a bus, or a car, etc.)." We interpreted this sentence to mean that each maker of transportation manufactures only one type of transportation. For example, if Boeing makes planes, it cannot make trains.

We believe that other statements in the description were unambiguous and there was only one possible interpretation.

1.2 Classes and slots

Figure 1 shows elements of the class structure and some relations among classes.

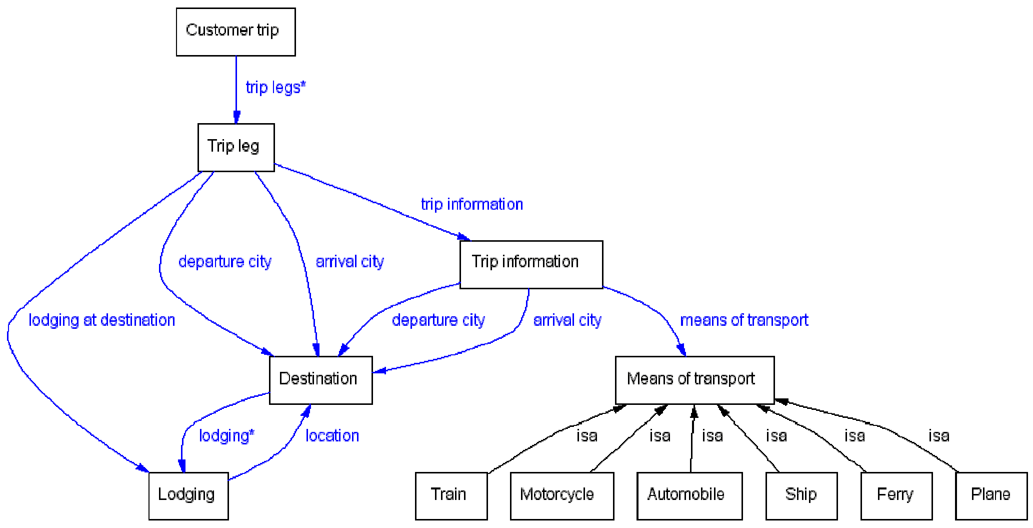


Figure 1. Elements of the class structure and relations. Boxes represent classes and arrows represent relations.

We start defining a customer's trip as an instance of the class `Customer trip`. Each instance of this class contains the customer's `name` and points to one or more legs of the trip. Each `trip leg` is an instance of the class `Trip leg` describing `departure` and `arrival time` and `departure and arrival cities`. It points to more specific `trip information`: the specific flight the customer is taking on that trip, or specific train, or the car he is renting. There is a constraint indicating that the arrival and departure cities for the trip leg must be the same as the arrival and departure cities in the corresponding `Trip information` instance.

An instance of `Trip information` represents information about particular flights, train rides, etc. That is, an instance of this class could be flight UA455 that leaves Paris at 9am and arrives to NY at 1pm every day. The `Flight` subclass of `Trip information` will include prices for economy and business class, and a flight number. We assume that this information does not change from day to day.

Arrival and departure cities on the trips are instances of the `Destination` class. In addition to the city `name`, its `country` and `continent` (we need the latter for one of the constraints), instance of the `Destination` class describes `local transport` in the city, `points of interest`, and a list of available `lodging` options. The options for the local transport are the default values for the `local transport` slot at the destination. The list of available lodging options contains instances of the class `Lodging`. In addition, each destination has a Boolean slot indicating whether it `has an airport`.

The `Lodging` class has two subclasses—`Hotel` and `Bed&Breakfast`. Each instance of `Lodging` points back to the `Destination` (the slot `location` is inverse of the slot `lodging` at the `Destination` class). Each `Hotel` instance has a required slot indicating its `star rating`. Each `Lodging` instance points to an instance of the `Room facilities` class describing individual rooms.

A class `Means of transport` represents different transport options for customer's travels. Specific means of transport are subclasses of this class. Each instance has a `make` and `model`. Hence, we can represent makes and models of particular planes, automobiles, etc. The `Means of transport` class is *abstract* to indicate that every instance of this class must be an instance of one of its subclasses. We attached a PAL axiom to this class expressing the constraint on makes and models: each maker produces only one type of transportation (see the Assumption above). Specific makes and models of planes, cars, etc. are instances of this class.
Instances of `Trip information` point to instances of the `Means of transport` class indicating which model of a plane, ship, train, is used for a particular flight, voyage, train ride, respectively.
There is a class `Distance` table which contains pairs of distances between destinations.

1.3 Constraints

The three constraints describing when customers would prefer to travel by train or car are PAL constraints.
The first constraint is "we know that it is not possible to go from America to Europe by train, car, bike nor motorbike." To express this fact, we attach the following PAL axiom to the `Trip information` class:

```
(forall ?trip
   (=> (and (name (continent ('arrival city' ?trip)) "Europe")
        (name (continent ('departure city' ?trip)) "North America"))
        (instance-of ('means of transport' ?trip) Plane      )))
```

A similar axiom expresses the constraint for the opposite direction (from Europe to North America).
The second constrain is "If distance between two cities is between 400 and 800 miles, and there is no airport close to one of them, the customer will prefer going by car or by train." We attach the following PAL axiom to the `Trip Leg` class:

```
(forall ?tripleg
   (=> (exists ?distance
          (and (to ?distance ('arrival city' ?tripleg))
          (from ?distance ('departure city' ?tripleg))
          (> ('distance in miles' ?distance) 400)
          (< ('distance in miles' ?distance) 800)
          ('has airport' ('arrival city' ?tripleg) FALSE)
          ('has airport' ('departure city' ?tripleg) FALSE)))
          (or (instance-of ('means of transport'
                                        ('trip information' ?tripleg))
                              Automobile)
      (instance-of ('means of transport' ('trip information' ?tripleg))
                  Train))))
```

To express the last constraint "The customer also prefer to go by car or train if he hates travel by plane.", we attach an axiom to the `Customer trip` class:

```
(forall ?customer
   (=> ('hates planes' ?customer true)
     (forall ?tripleg
      (=> ('trip legs' ?customer ?tripleg)
          (or (instance-of ('means of transport'
                                    ('trip information' ?tripleg))
                             Automobile)
```

```
        (instance-of ('means of transport'
                          ('trip information' ?tripleg))
                   Train))))))
```

1.4 Instances

To represent a specific trip, we create an instance of `Customer trip` (Figure 2). It has pointers to three trip legs. Each leg points to a Trip information instance describing specific flights for the trips to and from Madrid. We do not specify means of transportation for the New York-Washington leg.

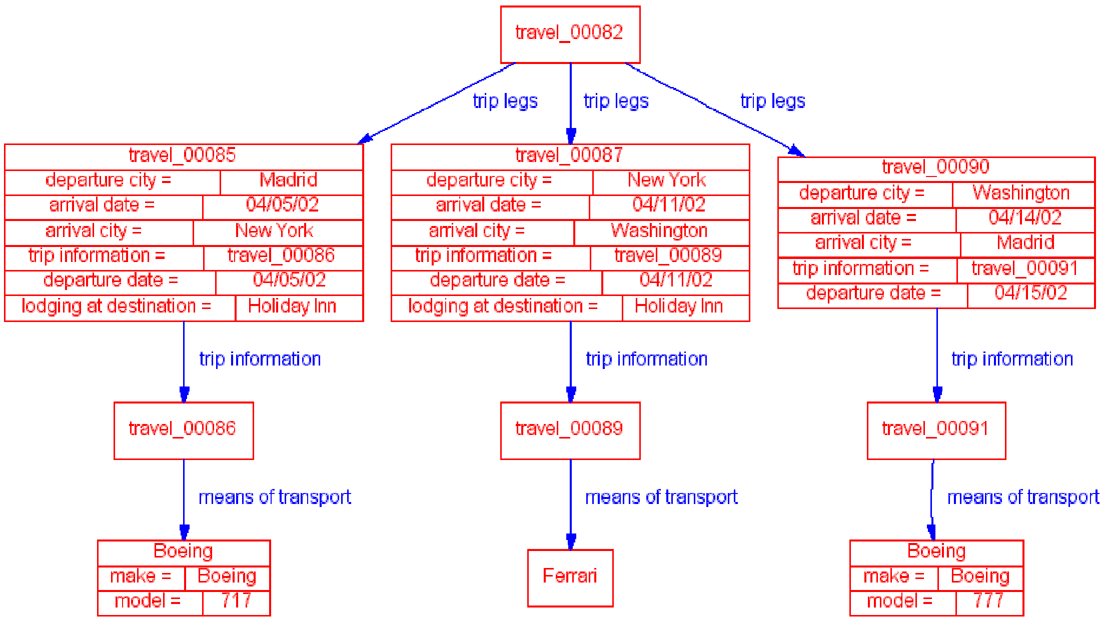


Figure 2. The customer traveling from Madrid to the US

2 *Discussion*

We were able to represent most of the facts from the description. We found that we had to revise the class structure significantly twice: First, when we got to the instance definition, we learned that a trip can have several legs and therefore had to add an intermediate `Trip leg` concept. Then, in order to express some of the additional constraints, we needed to add a number of new attributes to many of the classes and introduce the `Continent` class.

The class structure ended up being somewhat complicated. We believe that this complexity resulted from some of the requirements in the description: that customers can have several legs in one trip, using different means of transportation for each of them, that we define ticket prices for each flight, etc. However, these complexities exist in the real life and a real-life ontology would probably have been even more complicated.

We used many of the available knowledge-modeling primitives:

- *inverse slots* to link lodging and location

- *default values* to indicate default list of options for local transport at the destination. Designers can change this list for a specific destination since not all the towns have metro, for example; and some may have trams.
- *slots as first class objects* to attach the same slots to different classes. The slots `arrival city` and `departure city` are attached both to the `Trip leg` and `Trip information` classes. The slot `name` is attached to several classes as well.
- *abstract classes* to indicate that the subclasses of the `Means of transport` class enumerate all the possible means of transport

We used the Ontoviz plug-in to visualize relationships between classes and instances graphically (and to generate figures in this report). Being able to see the resulting structure in a graph, helped a lot in analyzing the emerging ontology.
We also used the Protégé Axiom Language to express domain constraints that could not be expressed in the frame formalism directly. In addition to the three constraints in the domain description, we specified a PAL axiom linking arrival and departure cities in the instances of `Trip information` and `Trip leg`. We also used a PAL axiom to express the fact that lodging at destination must be located in the same city as the destination.
There were several facts in the domain description that we did not represent. First, we did not represent the following fact: "From all of them, the travel agency is specially interested in flights, as it is the means of transport mostly used by its customers" In our representation, when we fill in the value for the means of transport slot (in the Trip information class), we put in *instances* of specific planes, trains, etc. thus, we cannot seta preferred *class* of transport.
"The most common destinations are ….". If there was only one most common destination, we could have put it as default value for destination. However, selecting some of the destinations as more common and some others as less common (given that we then set the corresponding slot value to only one of those destinations) was not possible.
We used the Protégé RDFS backend to generate RDF. Since the backend is designed to store all the information that is necessary to restore a complete project, we did not need any other format.

3 Conclusions

We were able to represent most of the information in the domain description. To do that, we used many of the knowledge-modeling features available in Protégé, such as inverse slots, default values, slots as first-class objects, abstract classes. Features that we lacked included more flexible default (or some other mechanism) to support preferences and prototypical instances.

Evaluation Experiment for Ontology Editors: SemTalk

The idea of the evaluation experiment for ontology editors is to model a given text with several semantic web related modelling tools. The experiment is part of OntoWeb-SIG3 EON2002 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management EKAW 2002.

The sample text to be modelled is a natural language description of a travelling domain and the task is to model a given flight problem. This text is not a real world document from a travel agency but a textual definition of a problem space. The information given in that document can be regarded as instructions for knowledge engineers how to model this specific domain. This has a major impact on the resulting model:

- Concepts like ‘Car’, ‘Plane’ etc. usually have not to be explained to a user
- They need to be defined to do machine based reasoning in this domain
- There are a couple of ontologies out there which already describe this domain

The product idea of SemTalk is to visualize complex scenarios, often described in documents, with symbols understood by non-technical users. SemTalk is not a tool intended to be used by an high-end knowledge engineers using all features of DAML or OIL. In the demo model we have tried to demonstrate the value which SemTalk adds to a complex solution. A solution framework for reasoning should include compatible high-end editors such as OntoEdit or Protégé beneath SemTalk to express more complex logic. The focus of SemTalk is to enable domain experts to express knowledge in a way that their customers can understand it as easy and fast as possible. SemTalk is competing in the discipline of usability and not in the discipline of most sophisticated ontology modelling.

On Finding Ontologies on the Web for Referencing

In a SemTalk typical scenario we would emphasise on the statements made in one document and relate them to an external ontology. One of the most important aspects of semantic web is to make sure that people are talking about the same topic and avoid to have different representations of it. The way to do it on the semantic web is to store agreed ontologies on a common accessible place like http://www.daml.org/ontologies and create references to objects included in the ontologies via URN / URL. SemTalk is using the namespace of the objects for making references into RDFS / DAML. Using the namespace as the locator of an object enables us to replicate and expand objects later on.

The first step in order to create the demo model was a search on the internet for existing ontologies.
Since there is still no specific ontology search engine out there this has to be done using Google and some background knowledge. Via http://www.daml.org/ontologies searching for travel you will find:

http://ontobroker.semanticweb.org/ontos/compontos/tourism_I1.daml	A couple of ontologies for travel posted by University of Karlsruhe, which are in German and can not be used for the experiment

www.daml.org/2001/06/itinerary/itinerary-ont	The interesting aspect about this one is, that the authors have been modelling "B777" and "First Class" as instances. One other reason not to use this ontology is that the current SemTalk did not understand daml:one-of and the missing 'Restriction' tag properly.
http://opencyc.sourceforge.net/daml/cyc-transportation.daml and http://opencyc.sourceforge.net/daml/cyc.daml	The problem with cyc-transportation ontology was, that the namespace for the objects did not match the location of the file.
http://xmlns.com/wordnet/1.6/	WordNet may be used as an RDFS Webservice in order to lookup common words and return their definition and taxonomy as RDFS.

The result of the experiment was, that we learned a lot about the syntactic variants of how DAML / RDFS has been used in existing ontologies. The SemTalk DAML import definitely has been improved. But we finally ended up using WordNet and the WordNet namespace http://xmlns.com/wordnet/1.6 for the classes and definitions, because it had textual definitions for those very general concepts like "Vehicle", "Car" and "Passenger".

Design issues for the SemTalk Model

SemTalk offers an explorer / browser to navigate the inheritance structure of the ontology. But the way SemTalk presents information to the end user is graphically.

The structure of the resulting SemTalk model in this experiment basically follows the structure of the text. We have tried to capture the contents paragraph by paragraph. For each paragraph a diagram (or "scenario") has been built. The thumb rule for the contents of a diagram is to make not more than 7 "statements" in one drawing. The diagrams actually contain now less than 20 objects each.
Ontologies are basically are boring thing. This does not really matter as long as they are used by machines, but it is an important issue if we are using them to transfer knowledge between humans.
One way to draw attention of people to models is to use pictures and symbols. SemTalk is based on Visio with the intension to make use of the existing Visio shapes. Visio shapes can be selected from a vector graphics based library shipped with Visio, from Office Cliparts or just by using arbitrary images. For this example we found it to be the fasted and most convenient way to use images taken from Google's image search. Using a couple of images in the graphical drawing of the ontology does not really add new information but it makes it more fun to read. Using an existing image is done by copying the jpg to the hard disk. Then drop it in the document stencil and rename it to the class name you need.

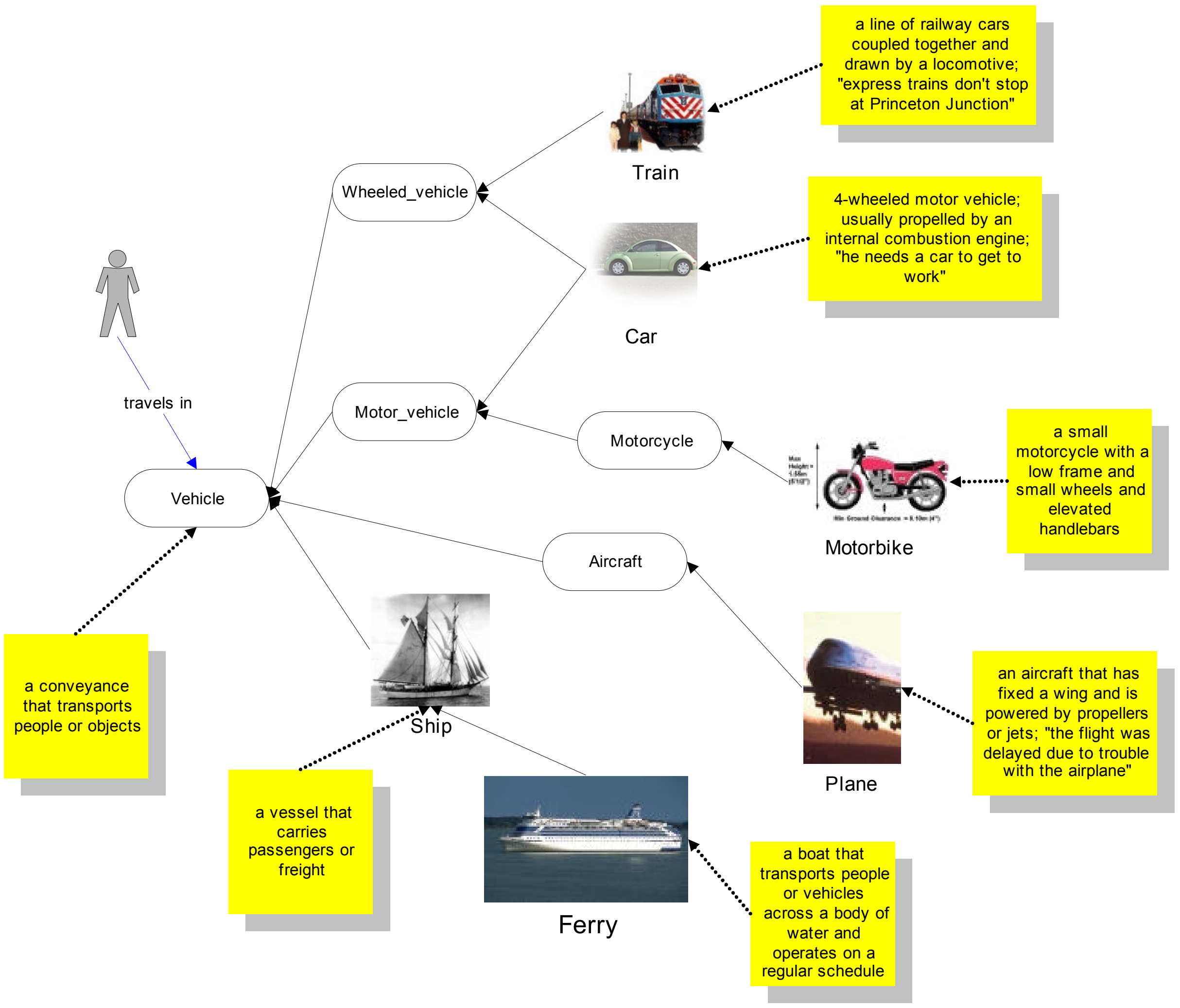

a line of railway cars coupled together and drawn by a locomotive; "express trains don't stop at Princeton Junction"
Train
Wheeled_vehicle
4-wheeled motor vehicle; usually propelled by an internal combustion engine; "he needs a car to get to work"
Car
travels in
Motor_vehicle
Motorcycle
a small motorcycle with a low frame and small wheels and elevated handlebars
Motorbike
Vehicle
Aircraft
a conveyance that transports people or objects
Ship
an aircraft that has fixed a wing and is powered by propellers or jets; "the flight was delayed due to trouble with the airplane"
Plane
a vessel that carries passengers or freight
Ferry
a boat that transports people or vehicles across a body of water and operates on a regular schedule

Fig.1: The Vehicle Ontology

We have attached the definition found in WordNet using a "Post-It"-style comment object. This often helps to understand the ontology even if the contents of that definition is actually ignored by any interpreter.

By assigning a Visio Symbol to a class in the ontology a kind of domain specific modelling tool for instances of the classes is created, where user can build the RDF instance model for is concrete statements using drag & drop from the ontology.

The diagrams in detail are showing:

Vehicle	A taxonomy of vehicle classes mentioned in the text
Agency	Displaying the fact stated in the text, that the agency is interested in subclasses of planes. This diagram demonstrated how to use object properties in order to express associations between objects. Since this a different statement than the vehicle taxonomy it should be visualized in a new picture instead of making the diagram to complex.
Flight	This diagram corresponds to the paragraph the text introducing attributes. The appropriate style to do this in SemTalk is to use UML-style shapes to visualize attributes. The focus of this diagram

	is to talk about the complex relations between Trip, Flight, Transportation and Topic. It also gives examples who SemTalk's inference supports property overloading (arrival, departure and used vehicle) .
Accommodation	This diagram does not add new constructs. It is basically there because it implements a lot of text about the subclasses of hotels and gives us a chance to add a picture of the tower of Chia.
Recommended Vehicle	One of the import aspects of the given text seemed to be the modelling of the relation between vehicles, transportation types and locations. This diagram shows how to do that in SemTalk again with property overloading. You may find the information that a train journey starts and end at a railway station and not at a seaport. For a train journey a train is used as a vehicle.
Destinations	This is our first instance diagram. It models the concrete destinations and continents as instances. Since we have not assigned symbols for city and continent we have used default shapes here. What we experienced as a missing feature in SemTalk was the possibility to assign individual pictures to single instances. This is currently only supported for classes.
Rules	SemTalk's native ontology modelling does not support a rule language or rule engine. Solutions like Integral, a graphical rule editor for SAP's Internet Pricing Configurator have been built on top of SemTalk.
TheTrip	This instance diagram shows simply the instances needed for John's trip.

The resulting model can be published as HTML. In the HTML document we have added source text with some hyperlinks to classes in the ontology.

We also can export RDFS or DAML from this SemTalk model. Classes only, instances only or both combined in one file.

The DAML files can be

- included as markup in the original documents or
- stored as markup besides the original documents or
- published on a server as a reference ontologie or
- used as an ontology spell checker within Office XP or
- ….

Modelling the travelling domain from a NLP description with TERMINAE

Nathalie Aussenac-Gilles (*), Brigitte Biébow (**) et Sylvie Szulman (**)

(*) IRIT, Université Toulouse 3, 118, route de Narbonne,
31062 TOULOUSE Cedex 4,
http://www.irit.fr, Nathalie.Aussenac-Gilles@irit.fr
(**) LIPN, Université Paris 13, Av. J.B Clément, 93430 VILLETANEUSE, http://www.lipn.univ-paris13.fr,
{Brigitte.Biebow, Sylvie.Szulman}@lipn.univ-paris13.fr

1. General TERMINAE method

First of all, TERMINAE proposes together a method and the tool supporting the method to build ontology from texts. The method relies on a linguistic analysis of the texts with the help of several natural language processing tools. We generally use two tools, one for term and relation identification, called SYNTEX [Bourigault,02], and another one for relation or role identification, called Caméléon [Ségéula, 99]. Both of these tools rely on the same linguistic hypothesis : the meaning of words and phrases is specific to a domain and can be inferred by observing the regularities of their use (in documents for instance).

The text here is too short (one page) to use these linguistic tools because they exploit repetition in the use of words or phrases. Nevertheless, we have used a term extractor (Syntex) that provides the list of all possible words and phrases available in the text, some relations between them (syntactical and grammatical dependencies), a direct access to all their occurrences as well as statistics such as their frequencies. Exploring the results of this tool is a complementary means to find out relevant concepts and knowledge.

A part of these results can be directly imported as an input in Terminae. These data are the input of the modeling process together with reading the original text. So identifying knowledge relies on two different main tasks that are carried out alternatively :

1) browsing the Syntex results to identify “important” knowledge or to decide how to represent some information according to the use of the words in the text ;
2) linear reading of the text to systematically extract as much knowledge as possible ;

Each piece of knowledge considered to be worth being integrated in the model is then represented. Terminae knowledge representation language relies on the following primitives : terminological file (for terms), generic concept (for classes), primitive concept (for instances) and role (for relations). The tool guides the various steps followed to define one of these item in the ontology.

The next stage in knowledge representation is normalization. The aim is to get to a well structured ontology, where each concept definition is justified through its relations with other concepts and comments. We suggest here to apply differentiation criteria that lead to make explicit the common and different properties of a concept with its father concept and brother concepts thanks to its roles.

The final stage is formalization in Terminae formal language, which is a kind of description logic. A classification function available in Terminae makes it possible to check the correctness of generic concept definitions. Concepts should be defined only once and have differentiating roles.

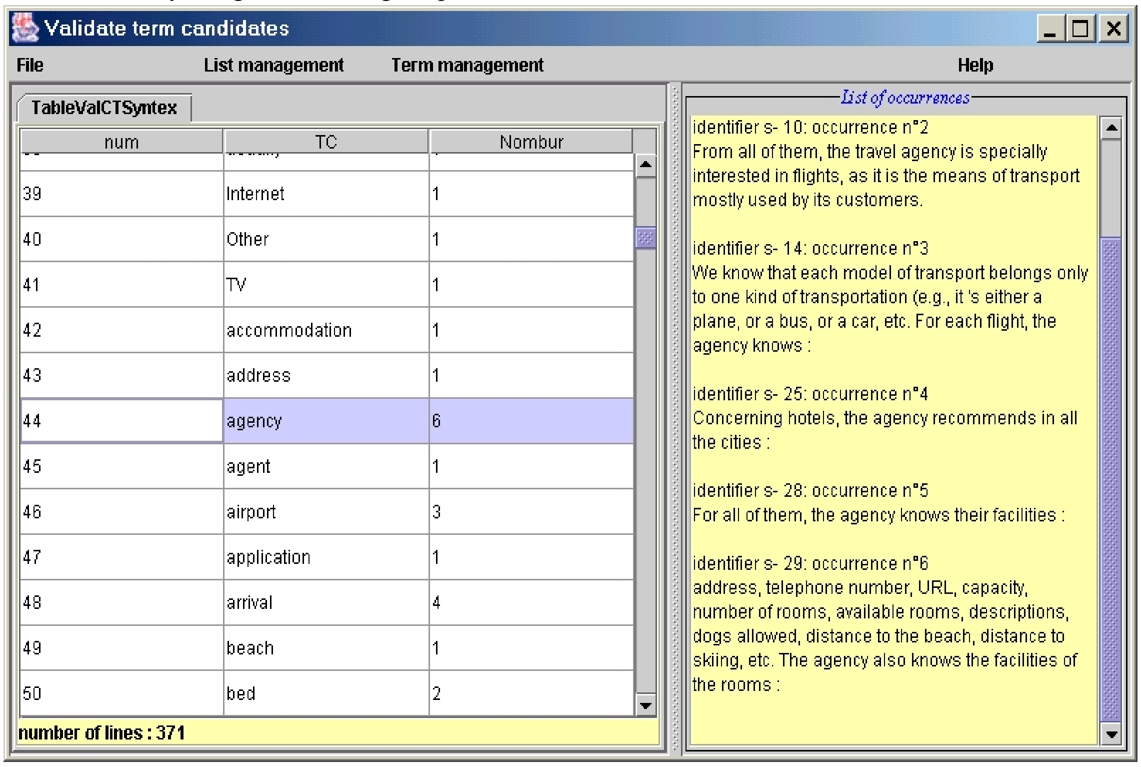
In the following, we describe how we proceed the two knowledge identification tasks, how we organize knowledge in the ontology and we illustrate the kind of consequences when applying the normalization rules. Then, we will list various modeling decisions that we took, whatever the way we identified the knowledge. TO end with, we will report some of the missing knowledge noticed by the classification function.

2. Knowledge identification tasks

2.1. Linear reading

Building concepts just from reading the texts assumes various facts :

1) The ontology builder knows enough domain knowledge to be able to decide which words (nouns, phrases, verbs or adjectives) are domain terms and possible concept or relation labels. In the particular case of this experiment, the domain is familiar to any one and common sense knowledge is almost enough to understand the text. In fact, we can suppose that it was one of the objectives of the writer, that every designer has enough expertise on the domain to model it.



num	TC	Nombur
39	Internet	1
40	Other	1
41	TV	1
42	accommodation	1
43	address	1
44	agency	6
45	agent	1
46	airport	3
47	application	1
48	arrival	4
49	beach	1
50	bed	2

number of lines : 371

identifier s- 10: occurrence n°2
From all of them, the travel agency is specially interested in flights, as it is the means of transport mostly used by its customers.

identifier s- 14: occurrence n°3
We know that each model of transport belongs only to one kind of transportation (e.g., it 's either a plane, or a bus, or a car, etc. For each flight, the agency knows :

identifier s- 25: occurrence n°4
Concerning hotels, the agency recommends in all the cities :

identifier s- 28: occurrence n°5
For all of them, the agency knows their facilities :

identifier s- 29: occurrence n°6
address, telephone number, URL, capacity, number of rooms, available rooms, descriptions, dogs allowed, distance to the beach, distance to skiing, etc. The agency also knows the facilities of the rooms :

2) Concerning the output, a similar implicit assumption is that the ontology builder knows well the way the ontology will be used, the task of the travel agent and how it could assisted with the help an ontology based system. This is much less obvious : in fact, we have tried to represented as much knowledge from the text as possible, without precise information about its relevance and use.

When we read the text linearly, we proceed in one the following ways :

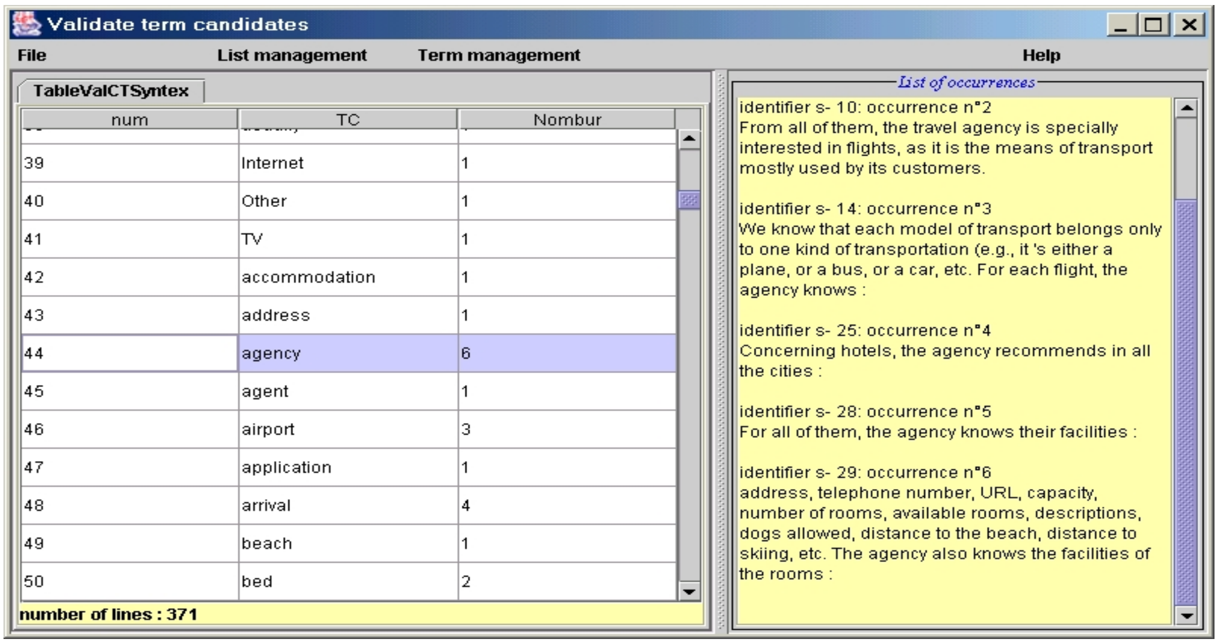
1) systematic inventory : from reading a sentence, we identify some concept names or role labels. It is frequent in this text because the writer prepares the ontology descriptions. For instance, in paragraph 2 we are suggested to define the "means of transport" concept as a class, and plane, car, ferries, trains, … as sub-classes of this concept. This leads to the definition of various concepts and IS-A relations in the ontology.
2) Structuring : some times, we use our domain knowledge to structure some information. We use it to make explicit with more abstract concepts some implicit knowledge in the texts. For instance, in the 5th paragraph about destinations, we are given a lot of examples (instances) and we are free to organize them into classes according to our mind. The same happens when deciding how to represent persons (we have two kinds of persons only : costumers (or clients) and the travel agent).

In both previous cases, the next step is knowledge representation in the ontology, with the definition of a concept (either generic or individual) or a role. See "Knowledge representation steps" below.

2.2. Browsing extracted candidate terms

Browsing Syntex results is generally much more efficient than reading when the domain is huge and the documents are numerous. For instance, if one or several books form the knowledge sources, Syntex criteria for identifying domain terms rapidly leads to find the main domain concepts and relations.

In the case of this project, the linguistic material is not prone to automatic processing. From Syntex we have obtained 372 terms (single words and phrases combing some of these words). Only 74 of them appear more than once, among 50 are relevant domain terms and some 5 or 6 refer to the domain of building ontologies. We can get read of irrelevant terms in a validation frame (see the screen dump below). For a given term, its occurrences are displayed to help decide whether to keep it or not. This work is a fastidious one, but rather fast in this case. It helps reduce the list of possible terms that will be browsed later on in the modeling process. We reduced the list down to 270 terms but many other terms could still be eliminated.



num	TC	Nombur
39	Internet	1
40	Other	1
41	TV	1
42	accommodation	1
43	address	1
44	agency	6
45	agent	1
46	airport	3
47	application	1
48	arrival	4
49	beach	1
50	bed	2

number of lines : 371

identifier s- 10: occurrence n°2
From all of them, the travel agency is specially interested in flights, as it is the means of transport mostly used by its customers.

identifier s- 14: occurrence n°3
We know that each model of transport belongs only to one kind of transportation (e.g., it 's either a plane, or a bus, or a car, etc. For each flight, the agency knows :

identifier s- 25: occurrence n°4
Concerning hotels, the agency recommends in all the cities :

identifier s- 28: occurrence n°5
For all of them, the agency knows their facilities :

identifier s- 29: occurrence n°6
address, telephone number, URL, capacity, number of rooms, available rooms, descriptions, dogs allowed, distance to the beach, distance to skiing, etc. The agency also knows the facilities of the rooms :

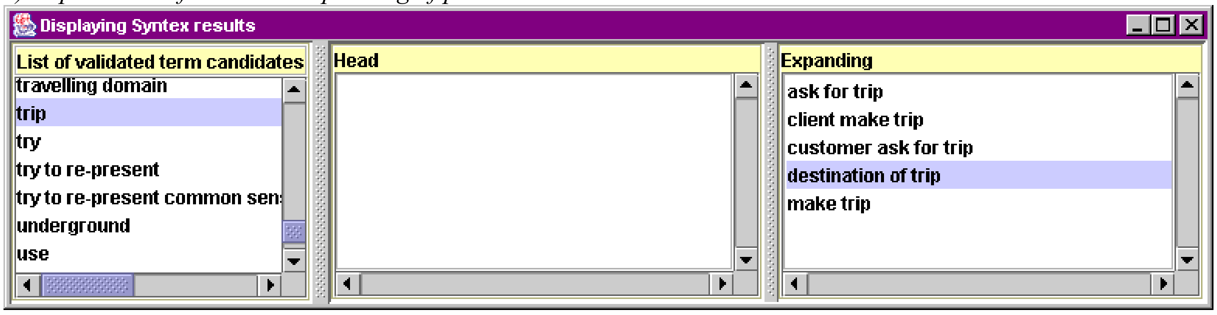
Although not completely adapted here, going from the list of possible (candidate) terms to the ontology is a good means :

1) to check the various use of a term
2) to identify synonyms (transport and transport means)
3) to automatically get comments that enrich the model and explain why some knowledge is represented in a certain way.

We can browse the list of possible terms according to their frequency, to the alphabetical order or (in the Syntex interface) to the grammatical category (verbs, nouns, noun phrases, verb phrases, adjectives or adverbs) and to the compositional relations (from phrases to their components, or from single words to the phrases there are used in). In this project, we did not use the Syntex interface. We carried out the following explorations :

1) looking for productive terms (that are part of many compound terms)
2) looking for the most frequent single terms: this often leads to major high level domain classes
3) looking for the most frequent noun and verb phrases : this leads to other main domain concepts and some domain relations
4) alphabetical exploration around the first identified terms.

1) Exploration of head and expanding of phrases

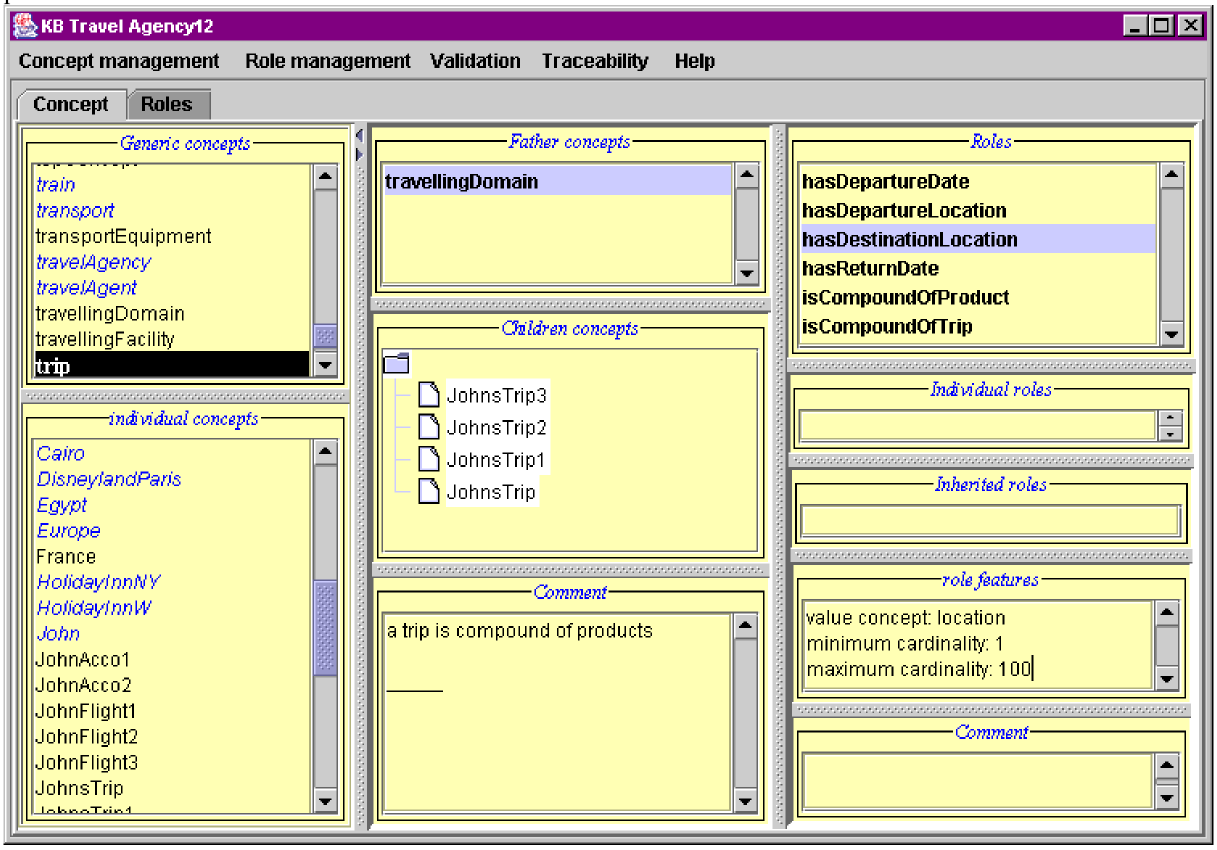

Displaying Syntex results

List of validated term candidates
travelling domain
trip
try
try to re-present
try to re-present common sen
underground
use

Head

Expanding
ask for trip
client make trip
customer ask for trip
destination of trip
make trip

Exploring terms and their constitutive parts (head and expanding) helps to figure out their productivity. Key domain concepts are more likely to be labeled by terms that belong to various domain phrases, that is to say to productive terms. For instance, the visualization of the expanding of the "trip" term helps to define its roles as presented below.


KB Travel Agency12

Concept management | Role management | Validation | Traceability | Help

Concept | Roles

Generic concepts
train
transport
transportEquipment
travelAgency
travelAgent
travellingDomain
travellingFacility
trip

individual concepts
Cairo
DisneylandParis
Egypt
Europe
France
HolidayInnNY
HolidayInnW
John
JohnAcco1
JohnAcco2
JohnFlight1
JohnFlight2
JohnFlight3
JohnsTrip

Father concepts
travellingDomain

Children concepts
JohnsTrip3
JohnsTrip2
JohnsTrip1
JohnsTrip

Comment
a trip is compound of products

Roles
hasDepartureDate
hasDepartureLocation
hasDestinationLocation
hasReturnDate
isCompoundOfProduct
isCompoundOfTrip

Individual roles

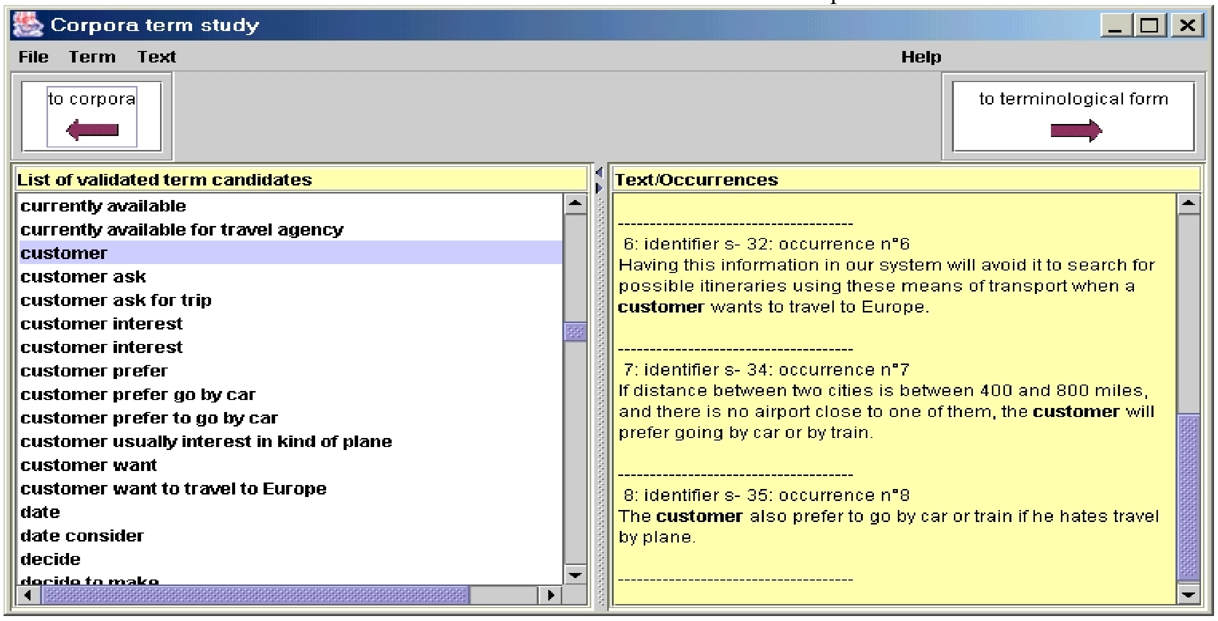
Inherited roles

role features
value concept: location
minimum cardinality: 1
maximum cardinality: 100

Comment

2) Exploration of the most frequent single terms

The screen below is the interface to be used to define a term and then a concept or role in the model.



currently available
currently available for travel agency
customer
customer ask
customer ask for trip
customer interest
customer interest
customer prefer
customer prefer go by car
customer prefer to go by car
customer usually interest in kind of plane
customer want
customer want to travel to Europe
date
date consider
decide
decide to make

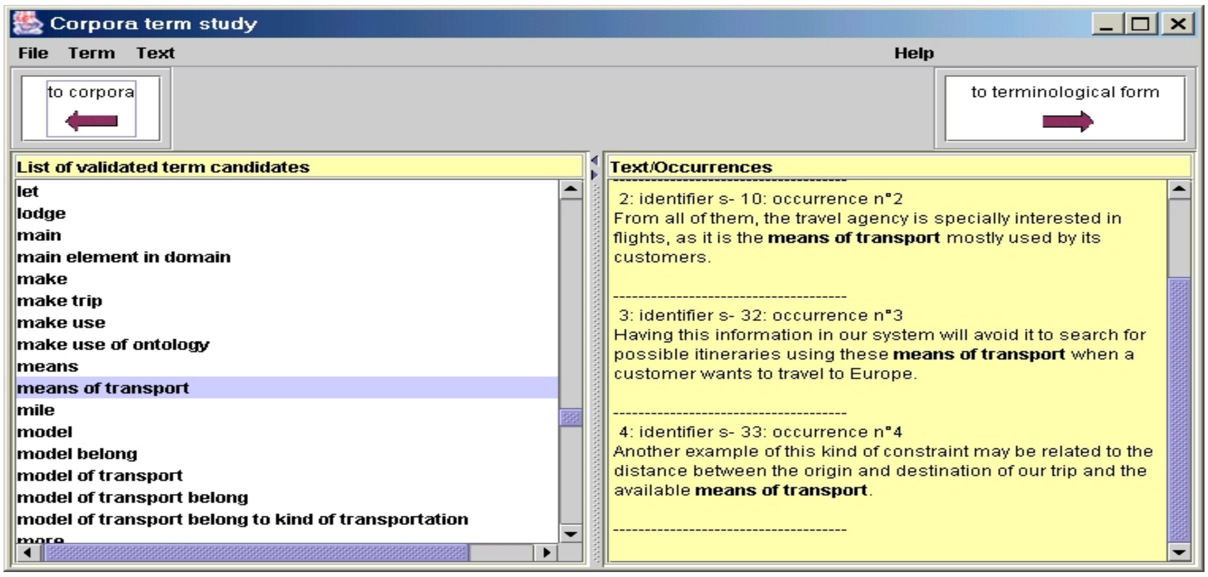
6: identifier s- 32: occurrence n°6
Having this information in our system will avoid it to search for possible itineraries using these means of transport when a **customer** wants to travel to Europe.

7: identifier s- 34: occurrence n°7
If distance between two cities is between 400 and 800 miles, and there is no airport close to one of them, the **customer** will prefer going by car or by train.

8: identifier s- 35: occurrence n°8
The **customer** also prefer to go by car or train if he hates travel by plane.

----

3) Exploration of the most frequent phrases



let
lodge
main
main element in domain
make
make trip
make use
make use of ontology
means
means of transport
mile
model
model belong
model of transport
model of transport belong
model of transport belong to kind of transportation
more

2: identifier s- 10: occurrence n°2
From all of them, the travel agency is specially interested in flights, as it is the **means of transport** mostly used by its customers.

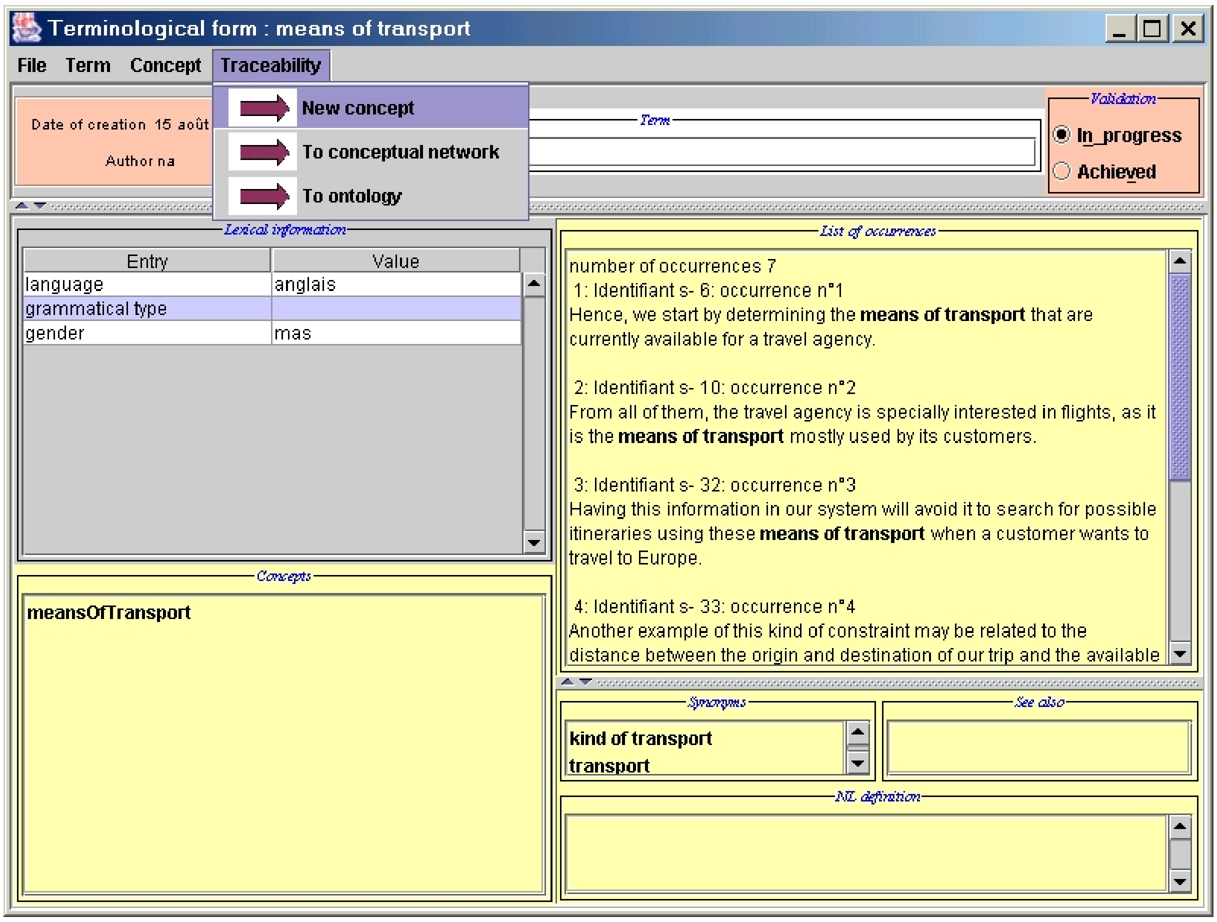
3: identifier s- 32: occurrence n°3
Having this information in our system will avoid it to search for possible itineraries using these **means of transport** when a customer wants to travel to Europe.

4: identifier s- 33: occurrence n°4
Another example of this kind of constraint may be related to the distance between the origin and destination of our trip and the available **means of transport**.

Defining a new term

When a terms in the list is considered a concept label, the ontology builder press on the “to terminological form” arrow, defines the corresponding term and then the corresponding concept. Here are the screens used for the definition a “means of transport” as a concept label.

The first screen is the terminological form where synonyms and new occurrences can be added, some linguistic information may be given. Here for instance, synonyms are “kind of transport” and even “transport”. SO the occurrences of these words have been added to those of the phrase “means of transport”.


Terminological form : means of transport

Date of creation 15 août

Author na

Term

Validation: [✓ In_progress] Achieved

Lexical information

Entry	Value
language	anglais
grammatical type	
gender	mas

List of occurrences

number of occurrences 7

1: Identifiant s- 6: occurrence n°1
Hence, we start by determining the **means of transport** that are currently available for a travel agency.

2: Identifiant s- 10: occurrence n°2
From all of them, the travel agency is specially interested in flights, as it is the **means of transport** mostly used by its customers.

3: Identifiant s- 32: occurrence n°3
Having this information in our system will avoid it to search for possible itineraries using these **means of transport** when a customer wants to travel to Europe.

4: Identifiant s- 33: occurrence n°4
Another example of this kind of constraint may be related to the distance between the origin and destination of our trip and the available

Concepts

meansOfTransport

Synonyms

kind of transport
transport

See also

NL definition

From this screen, a concept can be defined in the model thanks to the option “define a concept” in the pop-up menu “Traceability”. At this time of the process, the ontology builder only knows that these words are important as domain knowledge labels, but he does not know yet how the corresponding knowledge should be represented in the ontology.

The next steps are described in the “representations steps” paragraph.

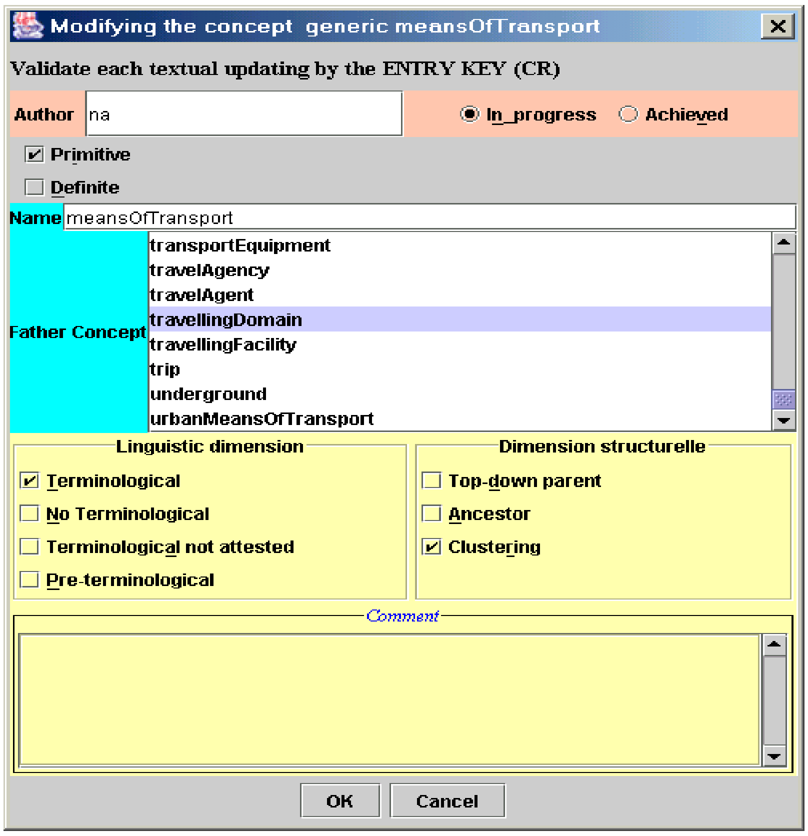
3. Knowledge representation steps

A concept may be created either from a terminological form or from the ontology editor (“create” option in the “concept” pop-up menu).

Each time a concept is created, a concept editor opens (screen bellow). The user must specify several properties :

1) The concept super class (father concept with the link isKindOf) in the hierarchy. The list of existing concepts is proposed. If the father concept has not been defined yet, the user can either enter its name and then define it or select TopConcept (the root of the hierarchy). For instance here, meansOfTransport is a class under travellingDomain. Terminae representation language allows multiple super classes and multiple inheritance of the roles. So several concepts can be selected as father in the proposed list.
2) whether the concept is terminological or not according whether it comes from the text or not. For instance, structuring concepts added from the builder own domain knowledge (e.g. country, urbanMeansOfTransport) are not terminological. This is a purely informative property that no impact on knowledge representation and formalization.
3) If the concept is built up from a terminological file, part of the occurrences may be cut and pasted to comment the concept. This help to easily store some design justification. Any other comment can be added too.
4) Whether the concept is primitive or defined. This refer to the formal representation with a description logic that is behind the Terminae interface. As long as formalization is no longer possible in Terminae, we did not check this property.

5) Whether the concept has been design following a bottom-up (ascendant) or top-down (descendant) process, or for structuring or gathering reasons. This property is also just for information. It keep tracks of one of the reasons that led to the concept definition : looking for a more generic class of various existing concept (bottom-up or gathering) or trying to list of the possible sub-classes of a given concept (top-down). For instance, the concept urbanMeansOfTransport has been characterized as a bottom-up one because it is the super class for a list of concepts (cityBus, taxi, etc.). NaturalSite is characterized as "top-down" because it is a way to list the sub classes of pointOfInterest.
6) When this first part of the definition is completed, the user can check the "OK" button. He will know later that this concept does not need to be checked again.


Modifying the concept generic meansOfTransport

Validate each textual updating by the ENTRY KEY (CR)

Author na   [✓ In_progress] Achieved

[✓] Primitive
[ ] Definite

Name meansOfTransport

Father Concept:
transportEquipment
travelAgency
travelAgent
travellingDomain
travellingFacility
trip
underground
urbanMeansOfTransport

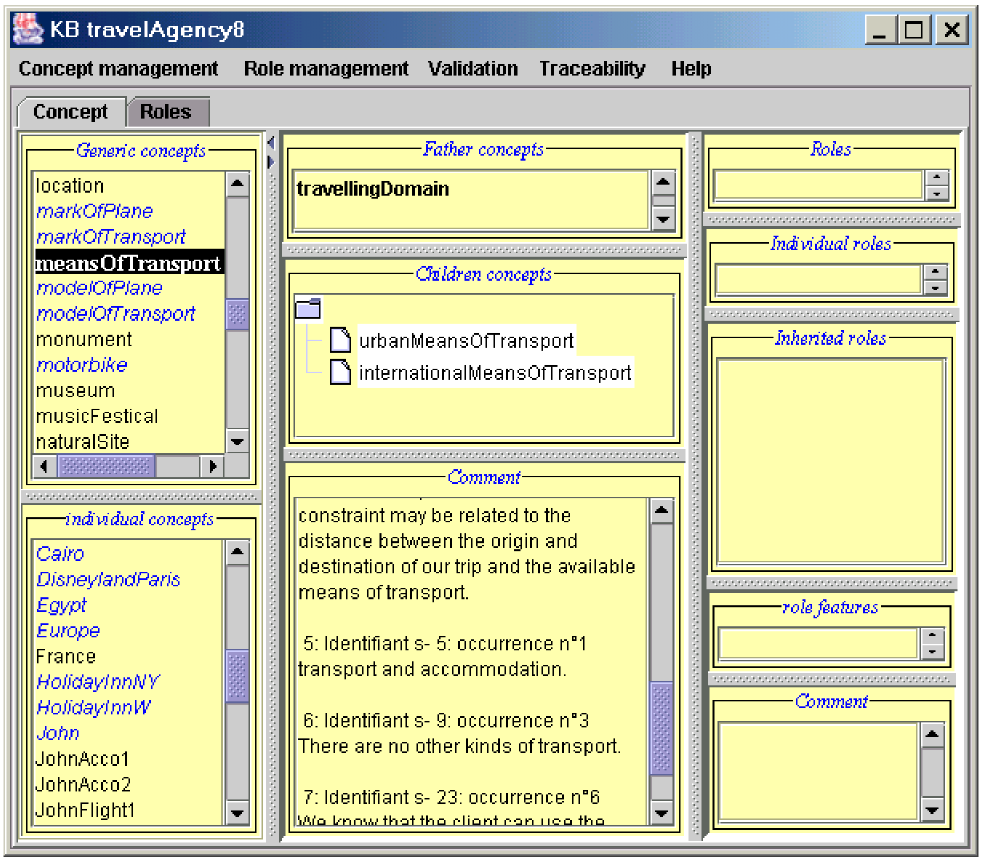
Linguistic dimension
[✓] Terminological
[ ] No Terminological
[ ] Terminological not attested
[ ] Pre-terminological

Dimension structurelle
[ ] Top-down parent
[ ] Ancestor
[✓] Clustering

Comment

OK Cancel

The concept is then inserted in the ontology and available from the ontology editor. The concept creation opens the ontology browser. The concept "meansOfTransport" is listed in the generic concept list of the ontology (left part of the screen bellow). Roles can be added later on.



Generic concepts: location, markOfPlane, markOfTransport, meansOfTransport, modelOfPlane, modelOfTransport, monument, motorbike, museum, musicFestical, naturalSite

Father concepts: travellingDomain

Children concepts: urbanMeansOfTransport, internationalMeansOfTransport

Comment:
constraint may be related to the distance between the origin and destination of our trip and the available means of transport.

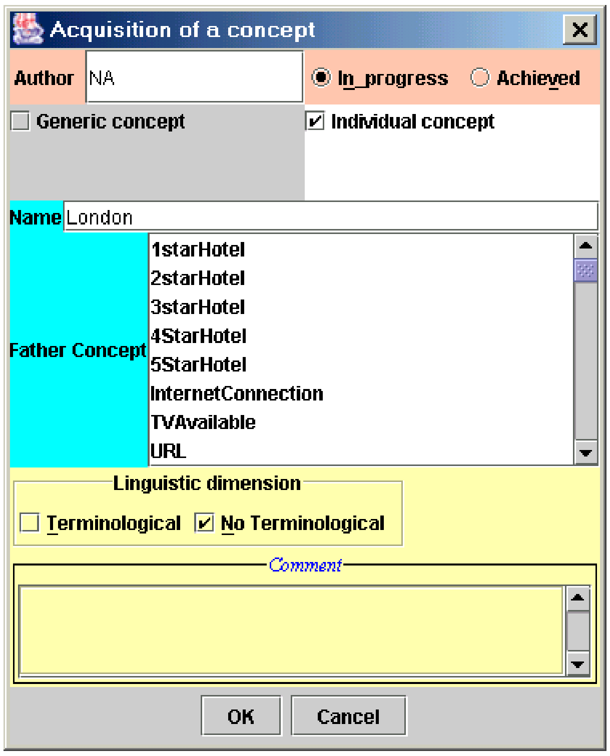
5: Identifiant s- 5: occurrence n°1
transport and accommodation.

6: Identifiant s- 9: occurrence n°3
There are no other kinds of transport.

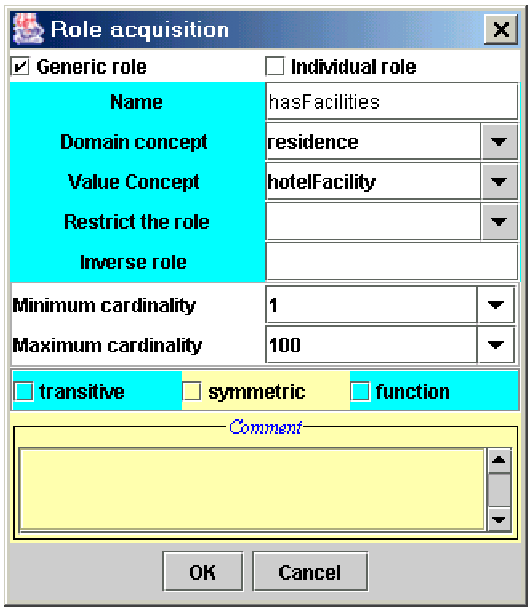
7: Identifiant s- 23: occurrence n°6
We know that the client can use the

individual concepts: Cairo, DisneylandParis, Egypt, Europe, France, HolidayInnNY, HolidayInnW, John, JohnAcco1, JohnAcco2, JohnFlight1

Creating an individual concept (an instance) is done through a similar editor but it requires less information shown below. On the left side of the ontology editor here above, the two kinds of concepts are show in two different alphabetical lists : generic concepts are in the upper part whereas individual ones are in the bottom list.



Author: NA — [● In_progress] ○ Achieved
☐ Generic concept [☑ Individual concept]
Name: London
Father Concept: 1starHotel, 2starHotel, 3starHotel, 4StarHotel, 5StarHotel, InternetConnection, TVAvailable, URL
Linguistic dimension: ☐ Terminological [☑ No Terminological]



[☑ Generic role] ☐ Individual role

Name	hasFacilities
Domain concept	residence
Value Concept	hotelFacility
Restrict the role	
Inverse role	
Minimum cardinality	1
Maximum cardinality	100

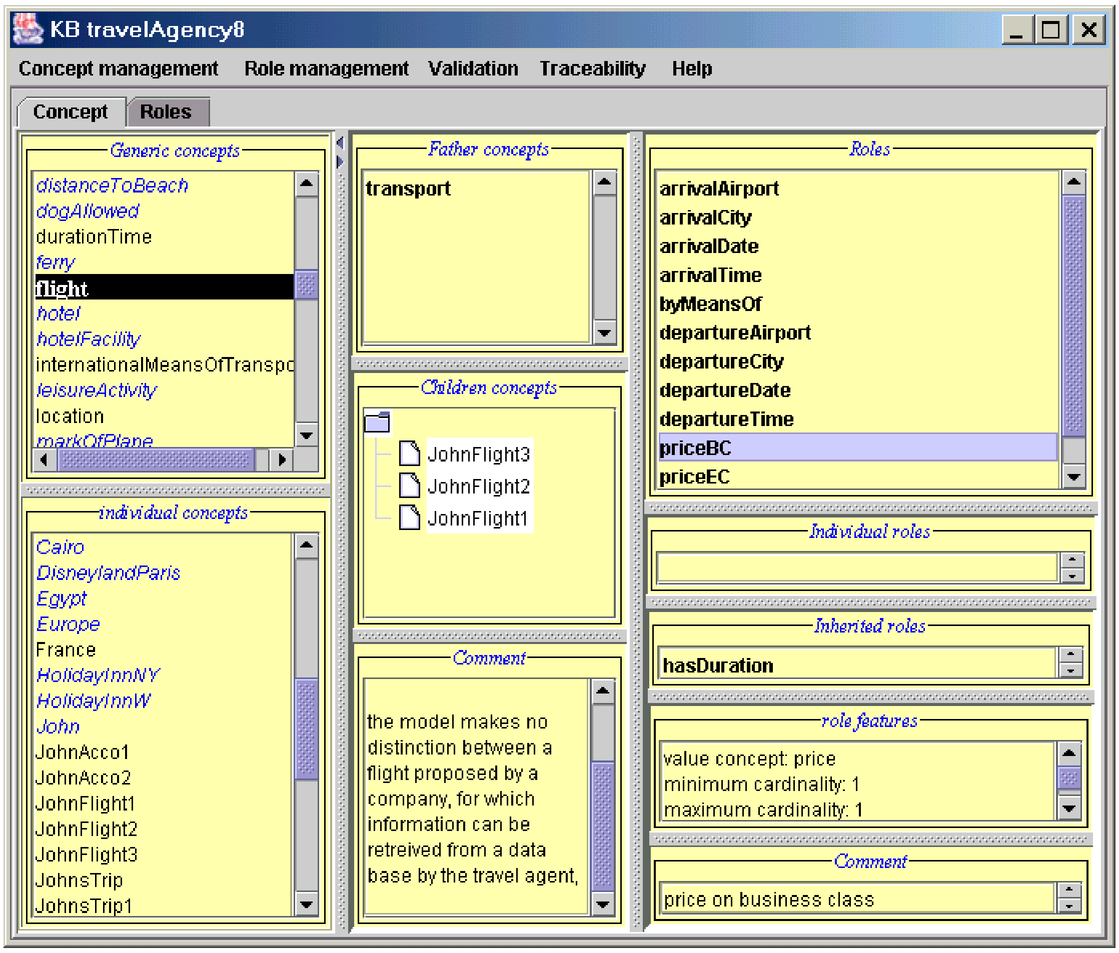
☐ transitive ☐ symmetric ☐ function

At any time, roles can be added that set relationships between concepts. The ontology builder may decide to add a role after having read the text (linear reading) or a term occurrence (when browsing terms). For instance, reading the 6th paragraph leads to define the concept hotelFacilities and to a role that connects the concept hotel with hotelFacilities. A role can be either generic or individual. Generic roles are associated to a concept and inherited by its sub-concepts; their value concept (destination) is a generic concept. Individual roles are specific to a concept, they are no inherited and can be associated individual concepts; their value concept can be either generic or individual.

We show above the role editor. The editor proposes the list of existing concepts to select the concept value (value concept). A new concept can be defined from there if required. A role can restrict an inherited role of a more generic concept in the hierarchy. It means that the new role associated to the specific concept will have a more restricted concept as the value concept. The option « restricts role » proposes to select one of the inherited roles of the current concept. Then the value concept must be a subclass or an instance of the value concept of this inherited role.

Some of the information associated to roles (symmetry, transitivity, functional, inverse role) is not interpreted formally. The only relation between roles is the "Inverse role" relation. Cardinality indicates the minimal and maximum number of associated roles of this type that a concept may have. Cardinality is used to check that an individual concept of this class as at least zero or one or no more than one or many roles of this types towards other individual concepts.

After a concept has been assigned roles, its selection in the ontology editor makes it possible to see all of its roles and the related concepts, its comments and sub-concepts.



Generic concepts: distanceToBeach, dogAllowed, durationTime, ferry, flight, hotel, hotelFacility, internationalMeansOfTranspo, leisureActivity, location, markOfPlane

individual concepts: Cairo, DisneylandParis, Egypt, Europe, France, HolidayInnNY, HolidayInnW, John, JohnAcco1, JohnAcco2, JohnFlight1, JohnFlight2, JohnFlight3, JohnsTrip, JohnsTrip1

Father concepts: transport

Children concepts: JohnFlight3, JohnFlight2, JohnFlight1

Comment: the model makes no distinction between a flight proposed by a company, for which information can be retreived from a data base by the travel agent,

Roles: arrivalAirport, arrivalCity, arrivalDate, arrivalTime, byMeansOf, departureAirport, departureCity, departureDate, departureTime, priceBC, priceEC

Inherited roles: hasDuration

role features:
value concept: price
minimum cardinality: 1
maximum cardinality: 1

Comment: price on business class

4. Limitations of Terminae representation language

Terminae representation language suffers from some limitations, mainly because constraints and relations between roles are not some of the primitives. Here are some of the other missing primitives :

1) operators such as OR, NOT to represent relations between concepts

2) existence operators like ONE-OF to represent a set of individuals as possible role values ; another solution would to enrich the possible types of role values : at the moment, it must be a concept, whereas in Protégé2000 for instance, it can be an instance, a value picked in a selected set or a generic type like String, Boolean, etc.
3) concrete types, as integer, string, …
4) more generally, axioms or relational expressions out of the language

For these reasons, we were not able to easily represent the kind of constraints defined in the 6th paragraph, about the way to go from America to Europe and so on. They could have been stored at least as comments.

Another limitation is that recent changes in the knowledge representation (like having individual concepts as role value even for generic concepts) make it now impossible to have a formal translation of an ontology in Terminae description logic.

5. Design decisions

In this section, we report our design decision according to the influence of the knowledge representation. Given the Terminae primitives, we still have the choice to represent an information in various ways. We motivate here some of our decisions. Some other decisions come from the application of differentiation rules. We will illustrate this normalization process in section 6.

5.1. Preliminary remarks

The text given to build the ontology intends clearly a target application which is not detailed. Although implicit, the objectives of the application lead to set some relations that would not be actually correct in a general ontology. They are acceptable because operational for the objectives. We clearly build a task ontology, not a generic one.
When representing a car as a means of transport, it is explicitly intended that it is a point of view no more detailed; if another point of view has to be taken into account, as the one of an automobile constructor, the modeling must be reconsidered.
Another illustration concerns rentalCar. It is a sub-concept of urbanMeansOfTransport and not a sub-class of car as it would be in a generic ontology. We could have used the multiple inheritance but this would have led to an inconcistency : a rentalCar would have been also an internationalMeansOfTransport. This is not false but not explicitly proposed in the text.

5.2. Choice between generic or individual concept

We report here two examples that led to two different decisions. In both cases, the starting knowledge is an enumeration of nouns. No significant syntactical (or linguistic) indication can be exploited. The choice comes from semantic and even pragmatic reasons.

1. Paragraph 3 "… the means of transport that are …. We will have in our ontology the following ones : planes, trains, cars, ferries, motorbikes and ships".
   We know that we have to represent in the ontology that "planes, ferries, …" are some means of transport (linguistic indicator : "the … that are available … are the following"). We have the choice between defining individual concepts or generic concepts related to the concept meansOfTransport. If we read the following of the page, we notice that we need to refer to several specific and real planes used for specific flights. So these real planes will be instances, whereas the notion of plane is considered here as a class, and requires to be a generic concept.

2. Paragraph 4 "For each flight, the agency knows : the arrival date, the departure date, the arrival city …"
   We have here another enumeration. From reading, we know that all the terms in this enumeration refer to some properties of a flight. In Terminae, properties are represented with roles, that connect the concept with another concept called the value concept. So we have to define as many roles and related concepts as there are properties in this list. Decisions must be made when defining the value concepts, that can be either generic or individuals.
   In this case, we decided to define the following roles and only generic concepts :

- the roles priceBC (price in Business Class), priceFC (price in first class) and priceEC (price in economy class) have the same value concept, price, which is a generic concept, and can be instanciated with specific price values;
- The roles departureTime and arrivalTime have the same value concept absoluteTime for similar reasons;
- The roles departureDate and arrivalDate also have the same concept value date;
- On the opposite, departureAirport and arrivalAirport have the same value concept airport;

In all those cases, we did not feel the need to differentiate the two kinds of airports, times, dates as classes because it has no meaning. These properties define roles. The same concrete object can play the two roles at different times. The decision is sometimes much more complex and hard to make, as described in the next section.

1) In the last paragraph, we had a similar problem with the various hotel classes. We define a role numberOfStar on hotel with value in starNumber. starNumber individuals are 1*, …5*. If we define each hotel class as an instance (individual concept), we can express that the Holiday Inn hotel in New York is a 4 star hotel thanks to the role numberOfStar. But then these individual concepts will never be used in the ontology as role value. Another solution is to define the various hotel classes as as many generic concepts (oneStarHotel, twoStarHotel, …) that are sub-concepts of hotel. Then The Holiday Inn hotel may be an instance of 4StarHotel. It is important to do so if we need to explicitly specify some of the facilities available in a 3 star hotel that make them different from a 4 star hotel for example. If the system does not need to do so because hotel classes and corresponding services are well known by the customers, it is no use defining concepts and the role numberOfStar of hotel is enough. We decided to define generic sub-concepts oneStarHotel, …, fiveStarHotel of hotel, with corresponding numberOfStar value.

5.3. Concepts or roles ?

Some knowledge may be represented either by concepts or roles. The choice may be difficult to make.

First example

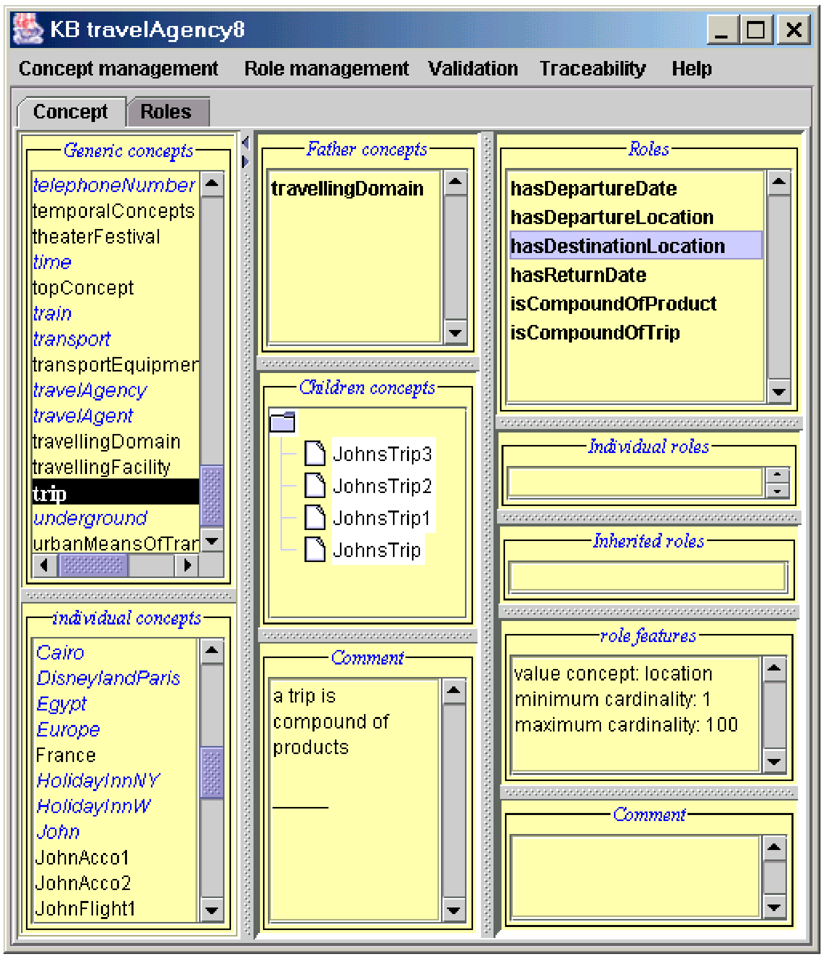
2) For instance, if it is important for the customer to know the model of plane of the flight, it must be one of the characteristics of a flight, or accessible from a flight (by the plane); this is expressed by the role meansOfTransport with value plane, which is inherited from the concept transport. So meansOfTransport is both the label of a generic concept and a role of the concept flight.

Second example (paragraph 5)

3) Destination are complex, including cities as points of interest located near a city with airport or even a continent. We first decided to call them all destination, to link this concept to trip. A destination may be a specific pointOfInterest as StatueOfLiberty, a city (and that implies a city to be a destination, that is not very correct), or otherDestination as Europe; each destination is linked to a correspondingCity. The problem is that otherDestination is a very general concept with many possible interpretations. It has to be made more precise.
1) In fact, all destinations are locations and destination is rather the label of a role of the trip concept. So we ended by defining the location concept, with sub-concept such as city, country, continent, pointOfInterest. Such concepts have connecting roles to mean that a city belongs to a country, and that this country is part of a continent. This helps refine with a customer the destination associated to his trip: if he wants to go to Europe, the travel agent can suggest him various European countries or cities or points of interest located in Europe.

5.4. Instance definition leads to modification in generic concept definitions

4) For defining Paris as being a destination including the city and Disneyland, we define DisneyLand as an individual concept of pointOfInterest. In fact, more generally, we decide that any location can have various points of interest, and we feel the need to classify them into classes so that the customer whishes may be refined.
5) It appears also that a trip may be compound of several trips : a trip to Europe includes trips to London, to Paris (city or DisneyLand) and to Madrid. We need to express a recursive definition: a trip may be compound of trips.



telephoneNumber
temporalConcepts
theaterFestival
time
topConcept
train
transport
transportEquipmer
travelAgency
travelAgent
travellingDomain
travellingFacility
trip
underground
urbanMeansOfTran

travellingDomain

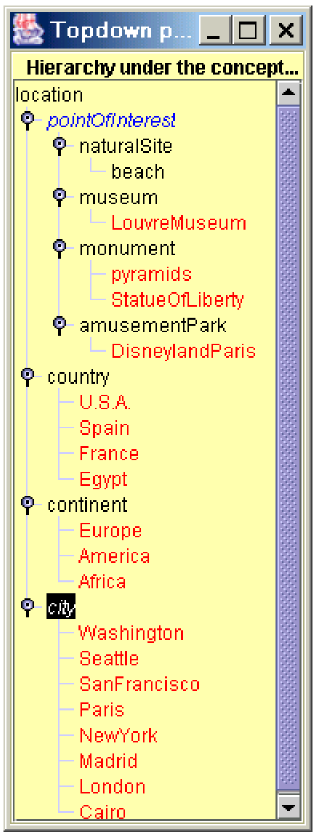
hasDepartureDate
hasDepartureLocation
hasDestinationLocation
hasReturnDate
isCompoundOfProduct
isCompoundOfTrip

JohnsTrip3
JohnsTrip2
JohnsTrip1
JohnsTrip

Cairo
DisneylandParis
Egypt
Europe
France
HolidayInnNY
HolidayInnW
John
JohnAcco1
JohnAcco2
JohnFlight1

a trip is
compound of
products

value concept: location
minimum cardinality: 1
maximum cardinality: 100



location
pointOfInterest
naturalSite
beach
museum
LouvreMuseum
monument
pyramids
StatueOfLiberty
amusementPark
DisneylandParis
country
U.S.A.
Spain
France
Egypt
continent
Europe
America
Africa
city
Washington
Seattle
SanFrancisco
Paris
NewYork
Madrid
London
Cairo

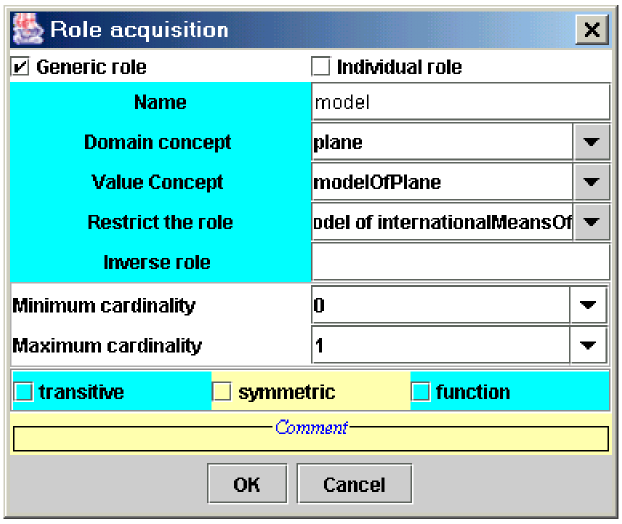
5.5. One or several concepts ?

We illustrate this with paragraph 3 “customers are usually interested in the kind of plane they will fly on …”.

1. A first choice can be to identify mark and model as two different properties of the concept plane (planeModel and planeMark). As long as we cannot give “string” as the class value of these roles, we have to define 2 concepts : planeMark and planeModel.
2. Another solution would be not to differentiate mark and model (for instance, AirbusA320 would give the two information with a single individual concept). The notion of planeModel would include both the mark and the type of plane. We would have a single role and a single concept.
3. A third solution would be to consider that the mark is an information associated to a model, not to a plane, the model being the only information associated to a plane. Then the plane concept would have planeModel as a role, and planeModel would have planeMark as a role.
4. The reserve solution is also possible : consider the mark as the information associated to a plane, and the model as a role of a mark. Although selected in a first time, this solution is not very relevant: an invividual mark would have as roles all the possible plane models of this mark. So it is not easy to represent that a given plane (an individual concept) that has a given mark (individual role) is then related to a specific model.

So we choose the third solution because it seems to be more general and adequate. It allows to know the model of a plane in a first time, and to precise the mark if required. The relations between plane, markOfPlane, and modelOfPlane are delicate to establish. The difficulty is that the need of modeling mark and model of a plane appears when defining means of transport although this notion concerns a specific trip in plane, not plane as means of transport.

The same difficulty appears with other means of transport. For this reason, we decided to generalize the mark and model roles to any means of transport. This means that the roles were associated to the concept meansOfTransport and modelOfTransport rather than plane and modelOfPlane. As a sub-concept, plane inherits of the modelOfTransport role. So we restricted it with the value modelOfPlane, which is a sub-concept of modelOftransport. The same happens for MarkOfTransport and markOfPlane.

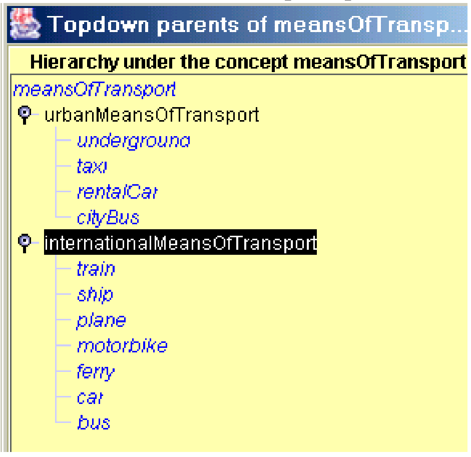


☑ Generic role	☐ Individual role
Name	model
Domain concept	plane
Value Concept	modelOfPlane
Restrict the role	odel of internationalMeansOf
Inverse role	
Minimum cardinality	0
Maximum cardinality	1

☐ transitive ☐ symmetric ☐ function

5.6. Define structuring concepts or not ?

When defining a flight, it seems time to define some structuring concepts : an agency sells a trip which is compound of products, that are either transport or accommodation. Transport by means of plane is a flight. We added another concept to illustrate what other transports could be roadJourney.
Because the concepts that refer to means of transport are different to go from a location to a destination location and to move inside this destination location, we differentiate two classes of means of transports: internationalMeansOfTransport and urbanMeansOfTransport. This information is not explicitely in the text. So the labels of these structuring concepts may not be the best one as long as rental cars are not specifically urban.
We present here bellow the correspond part of the concept hierarchy.



Hierarchy under the concept meansOfTransport

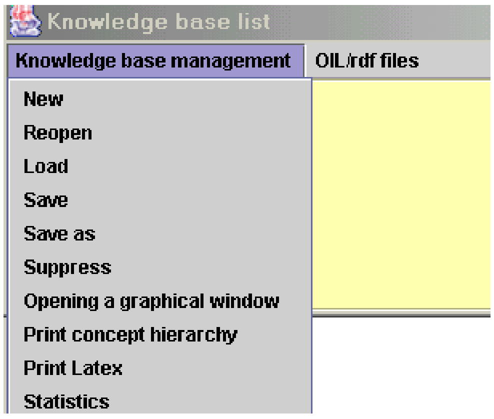
```
meansOfTransport
urbanMeansOfTransport
    underground
    taxi
    rentalCar
    cityBus
internationalMeansOfTransport
    train
    ship
    plane
    motorbike
    ferry
    car
    bus
```

5.7. Checking the model : what's in the concepts ? what does the ontology look like ?

Terminae offers several means to have a more concise view on the ontology. The default of the concept and role editors is to give a split view.

- the ontology editor provides a rather concise view of each generic or individual concept : we can see its specific roles, its inherited roles, its sub-concept and its father in the hierarchy ; for each role, we can know its value ;
- several options available from the KN management pop-up menu of the ontology list manager help see the whole ontology (see screen copy bellow)
    - the whole hierarchy can be seen and printed thanks to "printing the concept hierarchy"

- a Latex file can be printed : it proposes a frame like presentation of all the generic and individual concepts with their associated roles and values. We present here bellow an extract of this file.



```
:topConcept :travellingDomain :trip
Concept primitif
T
TDS
*****rôles *******
isCompoundOfTrip trip
isCompoundOfProduct product
hasReturnDate absoluteDate
hasDestinationLocation location
hasDepartureLocation location
hasDepartureDate absoluteDate
******************************************************
:topConcept :travellingDomain :trip :JohnsTrip3
NT
******************************************************
:topConcept :travellingDomain :trip :JohnsTrip2
NT
****** rôles individuels*******
isCompoundOfProductJT22 JohnAcco2
isCompoundOfProductJT21 JohnFlight2
hasDestinationJT2 Washington
departure NewYork
******************************************************
:topConcept :travellingDomain :trip :JohnsTrip1
NT
****** rôles individuels*******
returnDateJT1 April112002
isCompoundOfProductJT12 JohnAcco1
isCompoundOfProductJT1 JohnFlight1
destinationLocationJT1 NewYork
departureLocationJT1 Madrid
departureDateJT1 April52002
******************************************************
:topConcept :travellingDomain :trip :JohnsTrip
NT
****** rôles individuels*******
returneDateJT April152002
isCompoundOfTrip3 JohnsTrip3
isCompoundOfTrip2 JohnsTrip2
isCompoundOfTrip1 JohnsTrip1
destinationLocationJT U.S.A.
departureDateJT April52002
departureCityJohnsTrip Madrid
```

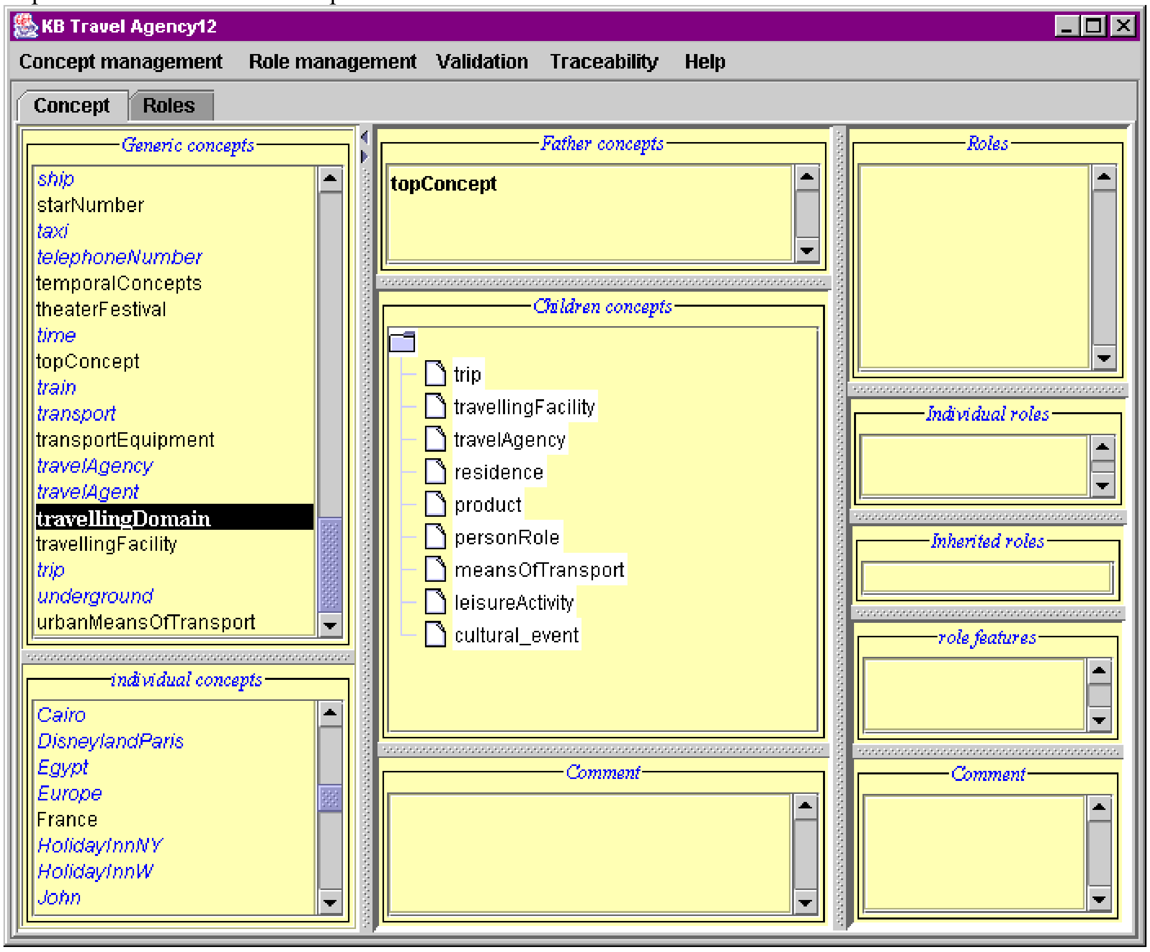
6. Normalization

Once most of the concepts found in the knowledge sources and required by the application needs have been added to the ontology, Terminae suggests to check the model according to differentiation rules. These rules lead to make explicit the modeling decisions. The knowledge engineer may require to look for additional knowledge back in the documents or from the expert. The differentiation rules require that for any given concept, the following information should be made explicit in the model (thanks to roles with Terminae representation language) :

- the concept must have at least one common role with its father concept (generally an inherited role);
- the concept must have at least one specific role that make it different from its father concept;
- the concept must have at least one share property (role) with its brother concepts (this role may be an inherited one);
- the concept have at least one specific property that make it different from its brother concept (this may be a specific role or value of an inherited role).

The application of these rules leads either to enrich the model or to eliminate some useless concepts or to reorganize the hierarchy with some intermediary concepts.

Example of not normalized concepts



Generic concepts:
ship
starNumber
taxi
telephoneNumber
temporalConcepts
theaterFestival
time
topConcept
train
transport
transportEquipment
travelAgency
travelAgent
travellingDomain
travellingFacility
trip
underground
urbanMeansOfTransport

individual concepts:
Cairo
DisneylandParis
Egypt
Europe
France
HolidayInnNY
HolidayInnW
John

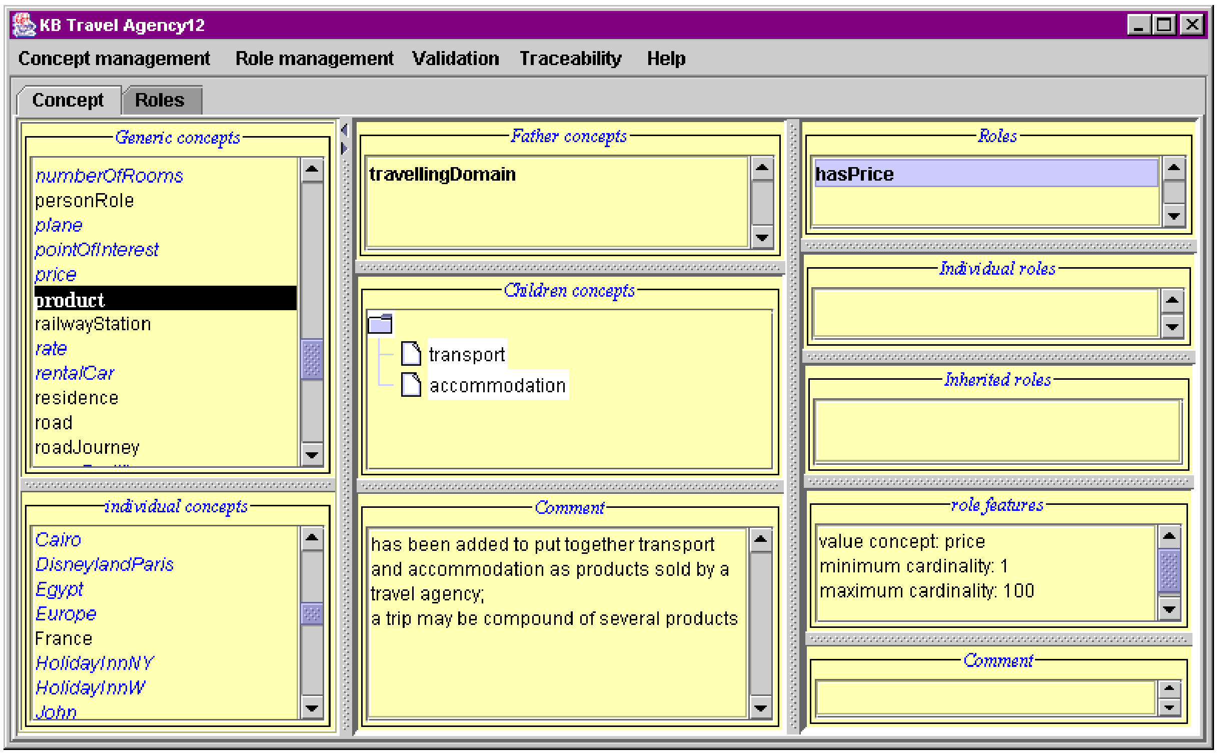
Father concepts:
topConcept

Children concepts:
trip
travellingFacility
travelAgency
residence
product
personRole
meansOfTransport
leisureActivity
cultural_event

The concept travelingDomain has no specific role yet, that would make it different from the root topConcept. In fact, this concept is an artificial means to inform the reader that, from this concept and below, all the information is structured according to the point of view induced by the travel agency application.

Example of differentiated concepts of the hierarchy
In the example bellow, the role hasPrice has been added to the concept product in order to stress the commonalities between a product, an accommodation and a transport. This role also contributes to differentiate this concept from other children of the travelingDomain concept. The children concepts transport and

accommodation are different because they have their own roles (byMeansof -> meansOfTransport and hasDuration are specific roles to transport; firstNight and lastNight are specific roles of accommodation).



Generic concepts: numberOfRooms, personRole, plane, pointOfInterest, price, product, railwayStation, rate, rentalCar, residence, road, roadJourney

individual concepts: Cairo, DisneylandParis, Egypt, Europe, France, HolidayInnNY, HolidayInnW, John

Father concepts: travellingDomain

Children concepts: transport, accommodation

Roles: hasPrice

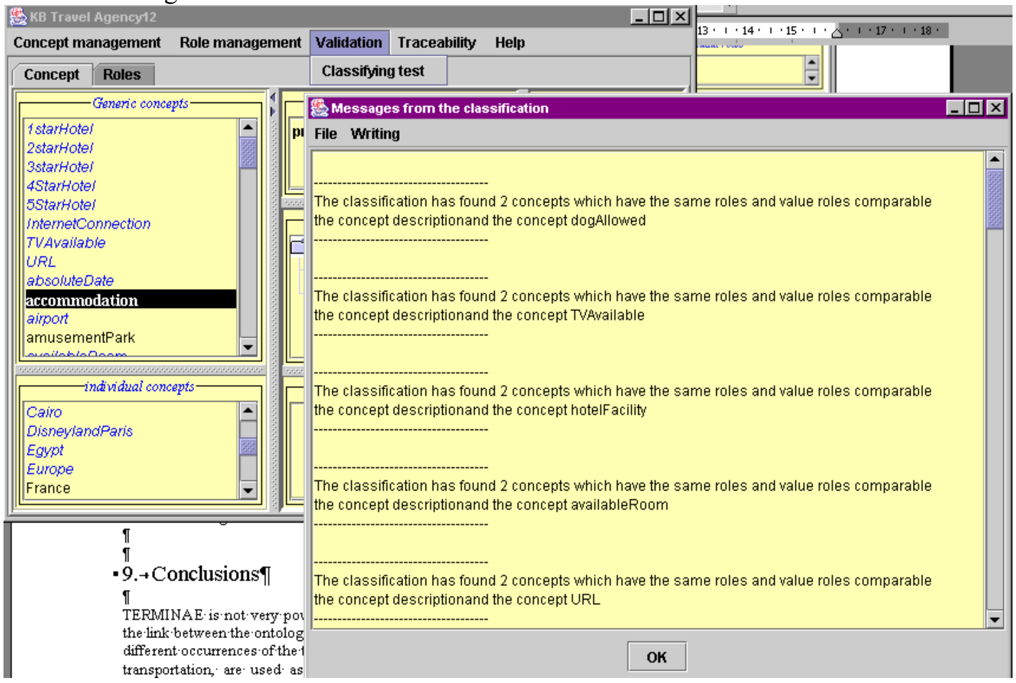
Comment: has been added to put together transport and accommodation as products sold by a travel agency;
a trip may be compound of several products

role features:
value concept: price
minimum cardinality: 1
maximum cardinality: 100

7. Concept classification

The validation option of the KB editor proposes a classification program. A report is then displayed to the user that can notice all the errors left in the model. The classifier expects each concept to be different from all the other, whether because it has a specific role or a specific value concept of a common role. Implicitly, the algorithm assumes that a concept is not worth being defined if it is not syntactically different from the other ones. Its label is not enough as a difference.

The screen copy bellow shows a report obtained before a systematic differentiation of our ontology. Most of the errors come from missing differentiation roles.



Generic concepts: 1starHotel, 2starHotel, 3starHotel, 4StarHotel, 5StarHotel, InternetConnection, TVAvailable, URL, absoluteDate, accommodation, airport, amusementPark

individual concepts: Cairo, DisneylandParis, Egypt, Europe, France

```
----------------------------------------
The classification has found 2 concepts which have the same roles and value roles comparable
the concept descriptionand the concept dogAllowed
----------------------------------------

----------------------------------------
The classification has found 2 concepts which have the same roles and value roles comparable
the concept descriptionand the concept TVAvailable
----------------------------------------

----------------------------------------
The classification has found 2 concepts which have the same roles and value roles comparable
the concept descriptionand the concept hotelFacility
----------------------------------------

----------------------------------------
The classification has found 2 concepts which have the same roles and value roles comparable
the concept descriptionand the concept availableRoom
----------------------------------------

----------------------------------------
The classification has found 2 concepts which have the same roles and value roles comparable
the concept descriptionand the concept URL
----------------------------------------
```

9. Conclusions

TERMINAE is not very pov
the link between the ontolog
different occurrences of the t
transportation, are used as

This control is optional and an ontology may be left with some errors according to these criteria. For instance, from the available document in our experiment, many knowledge is missing that would help to differentiate a hotel from a bed and breakfast, or to differentiate formally all the hotel facilities. May be these facilities should better be represented has individual concepts of the hotelFacility concept.

8. Output of TERMINAE

TERMINAE proposes various format for the output ontology and the terminological forms.
The ontology is stored by default in XML, and can also be exported in OIL or OIL-RDFs. It can printed as a LaTex file.
Each terminological form is stored in XML format .

9. Conclusions

TERMINAE is not very powerful as a representation language but rather as a guiding tool. Its main interest is the link between the ontology and the texts. For instance, the terminological form meansOnTransport gives the different occurrences of the term in the text, and it says that other terms in the text, kind of transport and kinds of transportation, are used as synonyms. A natural language definition can be given, which completes the conceptual definition. That helps the user to understand the underlying modeling of the ontology, and the modeling point of view.

We have shown the process from lists of terms to terminological forms and then to the ontology as an illustration of the guidance provided by the system. It would have been more powerful with a larger input set of texts. We are well aware of the need to have a formal model checking at the end.

10.References

AUSSENAC-GILLES N., BIEBOW B. & SZULMAN S., (2002), Modélisation du domaine par une méthode fondée sur l'analyse de CORPUS. In *Ingénierie des connaissances.* Paris : Eyrolles, à paraître.

BIÉBOW B. & SZULMAN S. (1999). TERMINAE: A linguistic-based tool for the building of a domain ontology, Proc. of the *11th European Workshop, Knowledge Acquisition, Modelling and Management (EKAW 99)*, Dagstuhl Castle (G), Springer Verlag, 49-66.

BIEBOW B. & SZULMAN S. (2000), Terminae : une approche terminologique pour la construction d'ontologies du domaine à partir de textes. *Actes de RFIA2000, Reconnaissances des Formes et Intelligence Artificielle*, Paris (F).

S. Le MOIGNO, J. CHARLET, D. BOURIGAULT, P. DEGOULET, M.-C. JAULENT (2002), Terminology Extraction from Text to Build an Ontology in Surgical Intensive Care. In *Proceedings of the ECAI2002 workshop on NLP and ML for Ontology Engineering*. Lyon (F). July 22-23, 2002.

SZULMAN S., BIEBOW B. & AUSSENAC-GILLES N. (2002), Structuration de Terminologies à l'aide d'outils d'analyse de textes avec TERMINAE, *TAL*, Paris : Hermès. Vol43,N°1. 2002.

Evaluation experiment for the editor of the WebODE ontology workbench

Óscar Corcho, Mariano Fernández-López, Asunción Gómez-Pérez

Facultad de Informática . Universidad Politécnica de Madrid
Campus de Montegancedo, s/n. 28660 Boadilla del Monte. Madrid. Spain
{ocorcho, mfernandez, asun}@fi.upm.es

Abstract. We summarize our design decisions on the conceptualization of a travelling ontology, when building it with the ontology editor of the WebODE ontology engineering workbench. This ontology editor is composed of a set of HTML forms, a graphical taxonomy editor called OntoDesigner and an axiom editor called WAB (WebODE Axiom Builder).

1 Introduction

This paper presents the results of using the ontology editor of the WebODE ontology engineering workbench to conceptualize an ontology in the domain of travelling and lodging. This is the first experiment for the evaluation of ontology tools' editors, performed in the context of the Special Interest Group (SIG) on Enterprise-Standard Ontology Tools of the European IST OntoWeb thematic network (IST 2000-29243). This experiment is described in section 2.5 of the OntoWeb Deliverable 1.3 [6].

In section 2 we will briefly describe the WebODE ontology environment, its ontology editor and its knowledge model. Section 3 will present the design decisions that we have made to model this ontology in WebODE, focusing on those pieces of knowledge that we have been able to model and on those pieces of knowledge that we have not been able to model with it. Section 4 briefly comments on the formats used to deliver the ontology: the XML representation of WebODE and RDF(S). Finally, section 5 will present some conclusions that can be derived from this experiment.

2 The WebODE ontology engineering workbench

WebODE [4] is an ontology engineering workbench developed by the Ontology Group at the Technical University of Madrid (UPM). It is the successor of the ontology design environment ODE [2].

WebODE is easily extensible and scalable, supported by an application server. The core of WebODE is its ontology editor. Ontologies are browsed and edited either with HTML forms (which allow editing ontology components and which provide "copy&paste" functionalities) or with a graphical user interface, *OntoDesigner* (which allows managing different views, where we can edit concept taxonomies with subclass-of relationships, disjoint and exhaustive subclass partitions and part-of relationships, and ad-hoc binary relations, and where we can either show or hide the different kinds of relationships in the ontology to highlight parts of it). The ontology editor also provides constraint checking capabilities, axiom and rule creation and parsing (with the WAB editor [4]), documentation in HTML, ontology merge, and ontology exportation and importation in different formats (XML, RDF(S), OIL, DAML+OIL, CARIN, Flogic, Java and Jess). Finally, its built-in inference service uses Prolog and a subset of the OKBC protocol [3].

2.1 WebODE's knowledge model

The WebODE's knowledge model [1] is based on the intermediate representations proposed in Methontology [6]. It allows modelling **concepts** and their **attributes** (both class and instance attributes), **concept taxonomies**, **disjoint** and **exhaustive** class partitions, **ad-hoc binary relations** between concepts, **properties of relations**, **constants**, **axioms** and **instances** of concepts and relations.

Bibliographic references can be attached to any of the aforementioned ontology components. Besides, it is possible to import terms from other ontologies. **Imported terms** are referred to by means of URLs.

Finally, the WebODE's knowledge model supports **views** and **instance sets**. Views highlight specific parts of the ontology in OntoDesigner. Instance sets make possible to populate a conceptual model for different applications or scenarios, maintaining different, independent instantiations of the same conceptual model in WebODE.

3 Conceptualization of the travelling ontology in WebODE

The ontology that we present has been conceptualized using Methontology. Methontology proposes to conceptualize the ontology using a set of tabular and graphical intermediate representations (IRs), and recommends the following order to assure the consistency and completeness of the knowledge already represented. First, we must identify the main concepts in the ontology and build the concept classification tree. Second, we create the ad-hoc binary relations between concepts in the same taxonomy or in different taxonomies. Then, we add the class attributes and instance attributes to the concepts, and finally we create axioms and rules. This is just a recommendation: this process is not necessarily sequential.

Therefore, our first task consisted of extracting concept taxonomies and their ad-hoc relations from the ontology description. We created five different **views**:

?? `Trip` view. A `customer` makes one or more `trips`, which use some kind of `transport` and `accommodation`. Here we understand by trip a combination of one or several transports and (possibly) an accommodation. That is, in the example, John will make three different trips: the one from Madrid to NY, the one from NY to Washington DC, and the one from Washington DC to Madrid.

?? `Means of transport` view, which contains all the concepts relevant to means of transport, classified by `air`, `ground` and `sea transportation`.
?? `Plane` view, which presents the concept taxonomy under the concept `plane`, which is contained in the previous view. This is done since the concept plane is the only one being more specialized in the ontology.
?? `Location` view, which contains the concepts `city`, `airport` and `importantPlace`, and their ad-hoc relations: a city may have several `nearest airports`, and several `important places` worth to visit.
?? `Accommodation` view, which contains the concepts related to accommodation. The most general concept is `accomodation`, which specializes in `hotels` and `bed and breakfasts`, as proposed in the NL description of the example.

In these views we modelled, if possible, disjoint and exhaustive partitions instead of simple subclass-of **concept taxonomies**. For instance, in the accommodation view we created an exhaustive partition of the concept `hotel` in the different hotel categories (from 1 star to 5 stars). And in the same view, we created a disjoint partition of the concept `accommodation` in `hotel` and `bed and breakfast`, since there are other types of accommodation that have not been included in the ontology.

Another important design decision related to the concept taxonomies was that of transport. We defined two different concept taxonomies for transports, considered as services (`flight`, `city bus`, `taxi`, `rental car`, etc.) and means of transport (`plane`, `car`, `bus`, `underground`, etc.). Both taxonomies are connected with the ad-hoc relation `usesTransportMean`, which is defined between the most general concepts in both taxonomies (from the concept `transport` to the concept `transportMean`) and specialized in some of the more specific concepts. For instance, between the concepts `taxi` and `car`, between `flight` and `plane`, etc.

Besides, another important issue is how to define flights from one city to another. We classified flights according to the air company in charge of them (this does not prevent a specific flight being subclass of several air companies' flights, in case of joint flights), and created a class for each flight code, that is, `aa0415`, `us1453`, etc. Instances of these concepts will be the specific flights in a specific date.

Once that we defined concept taxonomies and **ad-hoc relations** between concepts, we deepened in the description of concepts by defining concept attributes. We have selected the attributes that we considered most relevant for each concept, according to the description provided in the example. For each attribute, we provided its NL description, its value type, its minimum and maximum cardinalities, and its measurement unit, precision, minimum and maximum values for numerical attributes.

Class attributes define properties that describe the concept. For instance, the `number of stars` of each kind of hotel, the `economy`, `first` and `business class standard prices` of each kind of flight, the `air companies` of a flight or the `typical cruise speed` of a kind of airplane. These attributes have also their corresponding values.

Instance attributes define properties that will take their values in instances of the concept. For instance, in the concept `accommodation` we defined as attributes the `address`, `URL`, `phone number`, `number of rooms`, `number of available rooms`, `dogs allowed`, `distance to the beach` and `distance to a ski resort`. In the concept `flight`, we defined the `air company`, and the `departure` and `arrival dates`. In the concept `place`, we defined `longitude` and `latitude`, which can be used later to compute distances.

Later, we moved to the **logical axioms**, which are defined in first order logic, according to the WAB syntax. We created the following seven axioms:

?? Two axioms stating that the business class standard price of a flight is always more expensive than the first class standard price of the same flight, and for stating that the first class standard price of a flight is always more expensive than the economy class standard price of the same flight.

?? One axiom to obtain a flight's arrival city from the flight's arrival airport

?? Two axioms to establish that the only kind of transport that can be used for going from America to Europe, and vice versa, is a flight.

?? One axiom to state that in every kind of city transport the arrival city and departure city are the same.

?? One axiom to obtain the preferences of a customer for a trip, depending on the distance between two cities, as presented in the NL description of the example.

Many other constraints could have been inferred from the NL description of the problem. However, we have tried to restrict to the most relevant ones and those defined explicitly in the text.

A **bibliographic reference** was used to obtain many of the NL descriptions of concepts, attributes and relations: the `Merriam-Webster on-line`.

Finally, we created two **instance sets**: one for an agency in New York and another one for an agency in Madrid. We have created all the instances in the first one: an instance for `John`, three instances for John's trips, instances for the two hotels to be used, for the cities that he will visit, for the `Statue of Liberty` and for John's flights. We also created the instances of relations between these instances.

As a summary, we created in this ontology 58 concepts (organized in concept taxonomies with 23 subclass-of relations, 6 disjoint and 3 exhaustive partitions), 19 class attributes, 28 instance attributes, 21 relations, 0 constants, 1 reference and 7 axioms. We created 23 instances in the New York Agency instance set.

We found the following difficulties when modelling our ontology in WebODE:

Enumerated types cannot be represented in WebODE, in the sense of allowed values for an attribute in a class. For instance, they would be useful for representing the allowed values of the attribute hotel chain in the concept hotel, or for the continents in the attribute continent of the concept city.

We cannot represent attributes attached to ad-hoc relations. For instance, the number of rooms of each room type in a hotel. To represent this, we should create an intermediate concept that represents the relation, and define the corresponding instance attributes in it. However, we lose clarity and legibility in the representation.

WebODE cannot compute distances between places. This calculation must be done by external systems (such as inference engines or traditional software systems), which would be in charge of creating the corresponding instances of the concept `distance`.

4 Formats in which the ontology has been delivered

We have this ontology in two formats, automatically generated from the WebODE ontology editor: (1) **WebODE's XML** (in which we have all the components of the ontology that we have presented in the previous section), and (2) **RDF(S)**. In the transformation to RDF(S), we lose much of the knowledge of the WebODE ontology, since the RDF(S) knowledge model is less expressive than WebODE's. Hence, in RDF(S) we do not represent partitions, some attribute's information (cardinalities, measurement units, precision, minimum and maximum values), axioms and views.

5 Conclusions

In this experiment we have developed an ontology from a short NL description of the problem to be solved. It is clear that the ontologies that will be presented in this workshop will be very different from each other, since the problem description left open many modelling issues, so as to allow exploiting each ontology tool features.

This experiment is the starting point of a set of experiments that can be conducted by the ontology community. The domain of the experiment can be enriched, and also this experiment can be used for multiple purposes. For instance, it can be used: (1) to evaluate the tools' knowledge models, so that we can determine which components can be represented in each tool and which components cannot be represented, (2) to evaluate the possibilities of integrating the output generated by these tools with other ontology techonology (parsers, inference engines, etc.); (3) to analyze the possibilities of interoperability among tools; (4) to evaluate the usability of each tool; etc.

Acknowledgements

This work has been partially funded by the OntoWeb thematic network (IST-2000-29243) and by a FPI (*Formación de Personal Investigador*) grant from UPM.

References

1. Arpírez JC, Corcho O, Fernández-López M, Gómez-Pérez A (2001) *WebODE: a scalable ontological engineering workbench*. First International Conference on Knowledge Capture (K-CAP 2001). Victoria, Canada.

2. Blázquez M, Fernández-López M, García-Pinar JM, Gómez-Pérez A (1998). *Building Ontologies at the Knowledge Level using the Ontology Design Environment*, Proceedings of the Eleventh Knowledge Acquisition Workshop, KAW98, Banff, 1998.
3. Chaudhri VK, Farquhar A, Fikes R, Karp PD, Rice JP (1997) *The Generic Frame Protocol 2.0*. Technical Report, Stanford University.
4. Corcho O, Fernández-López M, Gómez-Pérez A, Vicente O (2002) *WebODE: an integrated workbench for ontology representation, reasoning and exchange*. 13th International Conference on Knowledge Acquisition and Knowledge Management (EKAW'02). Sigüenza. Spain.
5. Fernández-López M, Gómez-Pérez A, Pazos J, Pazos A (1999) *Building a Chemical Ontology using methontology and the Ontology Design Environment*. IEEE Intelligent Systems and their applications 4(1):37-45.
6. Gómez-Pérez A (editor) (2002) *Deliverable 1.3: A survey on ontology tools*. OntoWeb deliverable.