Practical Experiences in Developing Ontology-Based Multi-Agent System

Jarmo Korhonen  
Software Business and Engineering Institute, Helsinki University of Technology, Jarmo.Korhonen@hut.fi  

Pekka Isto  
Industrial Information Technology Laboratory, Helsinki University of Technology Pekka.Isto@hut.fi

Abstract

We have developed a prototype software system which uses software agents, an ontology and a pre-existing database from other software. This paper presents the system, and experiences in developing it. The main purpose was to find how to develop an ontology-based system in a real-world like case where the new software has to co-exist with existing software and database.

1. Introduction

This paper presents practical experiences in developing an ontology-based multi-agent system. The main motivation for this research was to evaluate the maturity of existing ontology technology and to gain experience in applying the technology to a concrete knowledge-intensive problem.

We used a database from an earlier project, BRIEFS [Seitsonen 2002], as the information source. BRIEFS is an information extraction system that collects a database of enterprise agreements, mergers and other operations from news briefs. The database is usually presented in a custom user interface to help corporate strategic planning. There is also another ontology project within the BRIEFS project, which is used for knowledge extraction purposes. We also compared the resulting ontologies that had been developed for different goals and with different tools.

The developed ontology was constrained by the database structure, as the practical research problem was to present that data in ontology-based form. We believe that it is common in practical problems to have existing systems that must be supported by new ontology systems. We also used other published ontologies as inspiration for our own.

In this research, our purpose was to provide a machine-understandable presentation of the database by using ontology, and test the approach within a multi-agent system. The multi-agent system was our chosen architectural model, as it enabled us to better research how separate software systems will in practice use shared information. The ontology is intended for sharing information. Multi-agent systems consist of multiple separate software agents and an agent platform that provides shared infrastructure and messaging interface to the agents. Here the agents use the agent platform to send messages with ontology-based content.

In practice, we implemented our Contract ontology with DAML+OIL, several collaborative software agents using FIPA standard, a simple message passing protocol using the ontology and a mapping from the database to the ontology. In Section 2, we list other work related to this experiment. In Section 3 the tool selection process and results are described. In Section 4 the architecture of the system is described. In Section 5 we describe the implementation experiences. Finally, in Section 6, we conclude our experiences on the ontology development in this practical case.

2. Related Work

In short, an ontology is a specification of a conceptualization [Gruber 1993]. Numerous ontologies covering a wide range of domains have been developed and presented, for examples, see [DAML]. The ontology presented is this paper has its roots in the ontologies developed for the BRIEFS information extraction system [Seitsonen 2002]. The development of our ontology was heavily influenced by the Enterprise Ontology and the methodology used to build it [Utschold 1998].

Enterprise Ontology was formalised into Ontolingua [Farquhar 1995]. Many formal or semiformal notations have been developed for representing ontologies. We based our ontology representation on the ontology languages from the DARPA Agent Markup Language (DAML) initiative [McGuinness 2002]. The first DAML
ontology language DAML-ONT combined properties from web language RDF, frame-based modeling approach and description logics for formal foundation. A merger of DAML-ONT with Ontology Inference Layer (OIL) resulted in the current DAML+OIL project with two released versions of the language. DAML+OIL stands as the starting point for the W3C’s Ontology Web Language (OWL) [Dean 2002], which is part of the Semantic Web Activity.

One of the main motivations behind building shared ontologies is the possibility of reuse. Our ontology was planned to include a DAML implementation of the Simple Time ontology [Zhou 2000], but due to the reasons explained below, the ontologies were never merged.

Ontologies have been used in multi-agent systems in many earlier systems. For example, InfoSleuth [Nodine 1999] is an ontology-based multi-agent system for managing distributed heterogeneous data sources. Furthermore, FIPA standard forum is developing a standard for using ontologies in multi-agent systems.

3. Tools

The tools selections were influenced by our simultaneous needs for prototyping and integration with existing components. Being newcomers to conceptual modeling, we preferred tools that allow fast take-off, ease of use, and conceptual clarity. Since we purported to use the languages from the DAML initiative, a level of support for DAML was required. Reasoning support was preferred to help construct a consistent ontology. We hoped to experiment with the practicability of merging ontologies. Thus, we tried to find editors with merging functionality. As we were integrating a system from existing software components, one of the most important selection criteria for the tools was their interoperability. We needed to have a parser that would be capable of parsing instances according to the ontology implemented with the selected ontology editor. The parser would need to be easy to integrate with the agent platform and the communication language mechanisms available on the platform.

Several ontology editors provide a level of support for DAML. KSL Ontolingua Editor [Farquhar 1997] supports DAML with its DAML-ONT. The DAML-ONT metaconcepts are based on several layers of metaconcepts build on top of KIF. That makes it very difficult to build DAML level ontologies since the concept definition errors may come from any of the several layers. A good understanding of the system is required to be able to successfully build non-trivial ontologies. Also, the user interface of the web based Ontolingua Editor is not very user friendly in terms of responsiveness and graphical appeal when compared to other editors implemented in compiled native or Java code. The strenght of Ontolingua Editor is support for distributed collaboration, but since the development of our ontology was made by a closely working team, that functionality was not needed. OntoEdit is a modular ontology engineering environment [Sure 2002]. Version 2.0 has support for DAML+OIL, but we did find the GUI not entirely intuitive. General axioms must be written in frame logic.

The development of our contract ontology was done with OILed v2.2 [Beckhofer 2001] and Protégé-2000 v.1.6.1 [Noy 2001]. OILed was the main implementation tool. It provided adequate features for building the ontology and verifying its consistency with FaCT reasoning system [Horrocks 2000]. Protégé-2000 with OntoViz tab v1.5 was used to print the ontologies at various stages of development.

We tested several available DAML parsers with ontologies taken from the DAML ontology library [DAML] and other sources. The tests included Jena v1.2.0 [McBride 2000], DAML API v0.6 [MARIA] and DAMLJessKB v010817b [Kopena 2002]. We included OILed editor in this test in order to get understanding of its capability of importing foreign ontologies. The test ontologies were fed to the tools and the number of DAML classes successfully parsed was compared to the actual number of classes in the test ontology. The results can be seen in the Table 1. The results are mixed. None of the tools managed to successfully parse all of the ontologies and we got a number of complete failures. Even when the parsers manage to successfully parse all the classes, they may return a large number of other concepts as RDF triples in addition to the DAML classes. Fortunately, our Contracts ontology seemed relatively easy to parse successfully.

<table>
<thead>
<tr>
<th>Ontology</th>
<th>#</th>
<th>DAML API</th>
<th>DAML Jess</th>
<th>Jena</th>
<th>OILed</th>
</tr>
</thead>
<tbody>
<tr>
<td>contracts-2001-03</td>
<td>29</td>
<td>0</td>
<td>29</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>contracts-2000-12</td>
<td>29</td>
<td>0</td>
<td>3</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>daml+oil-ex.daml</td>
<td>26</td>
<td>26</td>
<td>28</td>
<td>Fail</td>
<td></td>
</tr>
<tr>
<td>TestOntology</td>
<td>9</td>
<td>18</td>
<td>8</td>
<td>Fail</td>
<td>9</td>
</tr>
<tr>
<td>test.daml</td>
<td>1</td>
<td>Fail</td>
<td>5</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>lego-ldraw-representations</td>
<td>3</td>
<td>47</td>
<td>72</td>
<td>Fail</td>
<td></td>
</tr>
<tr>
<td>calendar-ont.daml</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td></td>
</tr>
<tr>
<td>DublinCore</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Fail</td>
<td>1</td>
</tr>
<tr>
<td>ksl-daml-desc.daml</td>
<td>23</td>
<td>0</td>
<td>3</td>
<td>23</td>
<td>0</td>
</tr>
</tbody>
</table>
We decided to use DAMLJessKB, since it was more robust in the tests than Jena and included inference capability provided by Jess [Friedman-Hill 1998]. However, since Jess is a production system and DAMLJessKB provided the results of parsing as RDF triples instead of DAML classes, we needed to reimplement an approximation of the semantics of the ontology with Jess rules. Since our agent platform was Java based, we needed to instantiate Java objects from the DAML+OIL encoded content in the FIPA-OS messages exchanged between the agents. Jess provided a good degree of Java integration to the rule base, but the rules needed to instantiate suitable Java representations of the DAML+OIL concepts had to be written manually for each concept. This was one of the major difficulties in developing our system. Our system would have needed a proper Java based DAML+OIL parser and writer with integration to description logics inference engine such as FaCT.

We had to choose between using a FIPA-standard compliant agent platform or some other, for example KQML-based Jackal platform. As we see that ontologies can be most useful in distributed computing environment, we decided that a message-passing system would best fit in such environment. And because we wanted the participants to share data using an ontology, we needed a platform that did not put any restrictions on the message content format. We decided to use a FIPA-compliant multi-agent system because it fulfilled that criterion. FIPA specifications also already had some notion of ontologies, even though they have not yet been finalized.

We chose to use FIPA-OS agent platform because it is open source and without commercial licensing restrictions. In technical sense, our agent messaging protocols were so simple that practically any FIPA platform would have sufficed.

4. Architecture

Our goal was to develop an ontology and use it in several different communication cases between different agents. This lead us to define three sets of agents: entity agents for managing different subsets of the data, query agents for offering query services using one or more entity agents to get the actual data, and insert agent that was responsible for adding new data to the database through the entity agents. Figure 1 shows the planned architecture of the multi-agent system.

![Figure 1 Agent system overview](image)

The entity agents were each responsible for one class of entities in the ontology. Company agent managed the entities that were in the company table of the database, including enterprises, standard bodies, governments etc. Agreement agent managed agreements in the database, including mergers, acquisitions, joint ventures and product purchases. Person agent managed person data that was gathered by the BRIEFS system to the database. Mainly, this data included person name and different companies that they had worked for. The database contained text type key fields, so we were able to separate the entity agents without risking reference integrity provided by the BRIEFS information extraction system.

Insert agent integrated to the BRIEFS system. Its task was to send new entity data to the entity agents for inserting to the database.

Query agents were responsible for user interface. They provided query services by contacting one or more entity agents for information, and then integrating this information as the query result. For example, the company deals agent queried company agent for company information, agreement agent for agreements relating to that company, and then again the information of the other participants in the resulting agreements. The notification agent in particular was intended for alerting the user for new interesting agreements.

Communication between the agents was straightforward, the only exception being that we used our ontology to code the actual data or query terms. For example, to query company information from the company agent, we sent a company entity with only the company name field filled, and got back the full company entity in the response message.

We have implemented the entity agents and one query agent, Company Deals agent. Only architectural
difference to the planned architecture was that entity agents used a filtered version of the database, not the original database. In particular, all communication between agents used the ontology.

5. Development experiences

5.1 Ontology

The ontology design was helped by multiple tools and examples that were available in Internet. As noted in Section 3, we used OILed and Protégé-2000 as our ontology editors. Initial version of our ontology was based on Enterprise ontology [4] and we also intended to include parts of the time ontology [8]. The merging of the ontologies could not be made, since OILed does not have merging capability and Protégé-2000 has only support for OIL. We just decided to omit the time reasoning functionality from our prototype. OILed proved to be a very good tool for designing and implementing our ontology. Integration with FaCT facilitated rapid development as the development versions of the ontology could be easily tested for consistency. Our experiments with DAML+OIL constructs convinced us rapidly to avoid axiomatic definitions and stick to relatively straightforward frame-like class hierarchies. It can be very difficult and non-intuitive to rely on axiomatic definitions, while class definitions are quite intuitive. Furthermore, our parser could handle class definitions relatively easily, but axiomatic definitions would have had to be approximated with the Jess reasoning engine adding significant complexity and overhead into the system. Protégé-2000’s OIL and OntoViz tabs were used mainly for printing ontologies exported from OILed. However, as expected, conversion to pure OIL resulted in some loss of information.

Using ontology in agent communication required both a parser and writer. This was the most challenging part of the implementation. DAMLJess required that the ontology had to be practically defined again as rules in the knowledge base. Once that was done, however, it was possible to read and write Java Bean objects to ontology objects and vice versa. The system also automatically checked the validity of the documents, which is very helpful feature if all agents are not known beforehand. Unfortunately, the rule system of our ontology was quite complex, and it took a long time for each read and write cycle. Furthermore, as a query went from user to query agent, to entity agent, and back to query agent, the conversions were made many times. The overall system performance was very poor due to the slowness of the rule engine. In real production systems there needs to be some ways to reduce this overhead.

5.2 Database

Designing an ontology for existing system instead of from scratch was quite challenging, and we had to tune our ontology a lot to fit it with the database model. The database was implemented with MySQL. It contained four tables with non-normalized structure that was suited for the information extraction routines of BRIEFS system.

Similarities between entity relationship model and ontology class hierarchy helped to find the essential classes for the ontology, but database contained implementation specific features that are not relevant in ontology. In this case one particular problem was the duplicates in the database that were inserted by the information extraction routines. We had to manually make a filtered copy of the database to get unique company, person and agreement identities that could be used with the ontology.

Even though usually databases do not contain duplicates, it is not uncommon that older production databases contain some odd structures and errors. It should be possible to use these databases with ontologies even with some incorrect data. Fortunately, our experiences also show that standard integrity constraints of normalized databases are adequate for ontology use as well. When we filtered duplicates from the database, it was easily usable with the ontology.

5.3 Queries

Queries in the system were generated by dedicated query agents. Query agent would get the attributes from the user, fill partially an instance document with given attributes and send it to correct entity agent. The entity agent would then receive the message and find needed information from the partial instance document. The instance document attributes are used to make the database query.

Query results are then sent back to the query agent as zero or more instance documents, filled fully with all information available in the database. All results were returned in one message.

We designed but did not implement queries based on other than direct attribute values. For example, query of employer history of a person (Person History agent) would require using time ontology with no direct counterpart on the database. Query agents could provide these special queries as their services.

All references were in text format in both the database and in the ontology. In a real production database, queries based on references would also have to be developed. As database foreign keys and ontology-based class references
are quite different, this may be nontrivial to implement. Simple data types were easy to map from database to ontology.

5.4 Agent platform

FIPA-OS provided relatively easy-to-use base classes for agent development. As we used a simple request-response protocol, we adapted the sample classes to query and entity agent base classes. These base classes were then extended to provide entity agents for each entity and query agents for different queries. This was relatively straightforward.

The message passing infrastructure of FIPA-OS was text-based. We used the ontology writer to serialize a Java bean object to the DAML+OIL format. The resulting string was put inside message body, and the receiver would then parse it back to a Java class. This simple scheme enabled us to use ontology to pass data between different agents.

This simple case did not benefit from the ontology-based communication. More complex problems, for example in cases where the participants do not know of each other beforehand, could benefit from the ontology. In this case, adding new query agents would have been possible without changing the existing agents. As a real-world example, a new subcontractor could be added to a distributed supply-chain system more easily when using shared purchase order ontology than with elaborate EDI protocols. Ontology can help to make a very flexible distributed computing environment.

6. Conclusions

We think that OILed with FaCT is an excellent tool for developing a small but non-trivial ontology. OILed is intended to be a simple notepad-like editor, but we would like to see it to have some functionality for merging ontologies. Reuse is one of the main motivations for ontologies and it requires the ability to merge ontologies or parts of ontologies. OILed is now an open source project, so it is possible to contribute to the development and use the existing source code for derivative works. The OILed source code could provide DAML+OIL parser and Corba interface to FaCT and thus solve the main knowledge representational disparity with in our system. Most of the tools and platforms considered in this study are under development, and while this paper indicates their status at the time of the experiment, anyone interested in using them for a particular task should of course reassess the current releases from his or hers point of view.

The knowledge extraction ontology within BRIEFS has similar concepts as the ontology developed in this project. The main difference seems to come from the fact that we were only able to see successful information extraction results in the database. The BRIEFS ontology, on the other hand, has all concepts that the software tries to extract. For example time of events was seldom present in the database, and thus it was not a key concern for us. However, the structures of the ontologies were very different. The goal clearly affects how the ontology will be formulated, at least in this case.

The tools for this kind of system were of varying quality and state of completeness. Agent platform was satisfactory for this simple system, but development was not very easy. Ontology tools were mostly at starting point, only ontology editors were completely usable. Most notable is that we did not find any support at all for making an ontology-based system that needs to co-exist with legacy systems or databases.

Ontology engineering in itself was relatively straightforward. In this case, we had to reverse-engineer the BRIEFS database to find the concepts. After finding the concepts, the relationships between them could be defined with most editors, and the ontology language was expressive enough.

The actual implementation tools were more lacking, and at the time of development, it would not have been possible to develop a real application using ontology in communication between different modules. We hope that the new W3C recommendation, OWL language, will encourage development of more and better tools.

7. Acknowledgments

We thank Seppo Törmä, Markku Syrjänen and Lasse Pajunen for ideas. Further thanks to Kenneth Kronholm and Zhao Deze for implementation help.

8. References


[MARIA] DAML API
  http://sylveste.r.va.grci.com/mariasite/codipsite/Tools/Components.html

