The Use of Ontologies as a Backbone for Software Engineering Tools

Dirk Deridder and Bart Wouters [*]
Programming Technology Lab
Vrije Universiteit Brussel, Brussels, Belgium
email: {dirk.deridder,bart.wouters}@vub.ac.be
http://progwww.vub.ac.be

Abstract

The last few years research efforts concerning ontologies in computer science and artificial intelligence have been focussed on methodologies, formalisms and tools to build and to browse ontologies. Although everybody is convinced of the power and use of ontologies, until now no real prove exists of their potential in the domain of Software Engineering.

In every phase of software engineering, communication is one of the major activities (in duration as well as importance). Throughout the whole software development life cycle natural language is one of the major means to describe the different artifacts used.

One of the most burdensome problems associated herewith is the natural language ambiguity. Very often mistakes are made because words can be interpreted in a variety of ways. This could be due to the fact that those persons work in another domain, speak another language or simply because the word has multiple meanings and the context doesn't single out the correct significance or meaning.

As a consequence of such a misunderstanding vast parts of a system might have to be rebuild, remodeled, re-coded and re-debugged resulting in unanticipated delays and increased costs.

[*]This Research was partially supported by Wang Global and the Brussels Capital Region, Belgium (CCOOS project)

This paper will reveal that by adapting existing techniques and applying knowledge from the domain of Computer Linguistics and Artificial Intelligence i.e. ontologies and ontology-related techniques, we can improve the creation, verification and validation of software artifacts created during the software development life cycle. By integrating an ontological engine into a CASE-tool, we will demonstrate our point.

1 Introduction

In 1998 Gruber made the observation that

> ...in the AI ontology community, there is a dearth of well-developed applications reported in literature. One possible explanation is that the research community has been more focussed on the creation of tools and methodologies for building ontologies, and the building of particular ontologies, than on putting them to use in applications. That ontologies will ultimately deliver value thus remains largely a matter of faith ... [Usc98]

This citation expresses our feelings. It is in anticipation to this urge, that we decided to do some experiments with respect to the use of ontologies in applications. And, more precisely, applications in the world of Object-Oriented Modeling.

It is not our intention to start writing yet another paper on how to represent and/or model ontologies and on how to acquire the knowledge to store into these ontologies. We believe already enough people are doing research in this aspect of the ontological domain. The sole goal of this paper is to focus on the use of ontologies in SE applications. We will assume for the occasion that an ontology formalism and means of storing already exist.

1.1 Ontologies

The first notions of ontologies can be traced back to in philosophy where they were amongst others defined as

> the metaphysical study of the nature of being and existence [Lab98]

This definition was later adopted by the artificial intelligence community to refer to a set of concepts or terms that can be used to describe some area of knowledge or build a representation of it [ST99].

In this community, the most general and most accepted definition for an ontology is the one given by Gruber:

> An ontology is a specification of a conceptualization. [Gru92, Gru94]

In neither of both communities however, a consensus has been reached about what an ontology exactly is in terms of requirements. Resulting from this lack of consensus a whole set of definitions can be found in literature - often ambiguous, overlapping or interpretable in a multitude of ways - [GG95]. As a consequence of this ambiguity, there are as many formalisms to represent ontologies as there are definitions. Every one of them is in one way or another related to one or more other formalisms, and definitions only introducing small changes or additions. Recently though, efforts have been done to come to one formalism to represent everything in a certain universe of discourse. A nice example of this effort can be found in [MOF].

1.2 Integrating ontologies into CASE tools

In [EP98] an overview is given of a number of desirable CASE tool properties. We will point out some of them for supporting our discussion. The most important amongst them is that a CASE tool should provide a central repository. The authors state that all the collected information about a model should be stored centrally, which enables the tool to perform a number of interesting tasks. These are consistency checking, critiquing, reporting, and element or diagram reuse. A striking example given in this context is that a repository facilitates renaming a class and having this change broadcasted throughout the entire model. We believe that this is a very weak example of the power of such a central repository, but unfortunately it is a good indication of how case tools are currently making use of it.

More interesting would be to provide a means to solve what is known as the language ambiguity problem, for instance. This is a widely known problem in all areas where multiple agents use the same resources and artifacts. The language ambiguity problem is a widely known problem whenever a natural language participates in the software engineering process. Generally it boils down to the situation where different agents use the same term to denote a different concept. As Furnas already pointed out in [FLGD87]: "the probability of two subjects, picking the same term for a given entity ranged from 7 to 18%".

This is where enhancing the central repository with an ontological engine will be particularly useful. As this enhancement will promote the repository to be an active component instead of a passive one, this will improve the capabilities of the tool in completing the earlier mentioned tasks. Further, we predict that this list of tasks that might be performed by our case tool will have to be extended once this 'activation' has occurred.

For our experiments, we will integrate an ontological engine into the Rational Rose CASE tool. The data used by the ontological engine will be based on a slightly modified version of WordNet 1.6

An ontological engine can be classified into two categories according to the way it uses the ontological data, i.e. ontology-driven and ontology-based.

Ontology-driven engines use the ontological data as an active shared enterprise memory. The engine retrieves data from the ontology within a given context, and uses it to guide the software engineer in performing the task at hand. During this process, the software engineer has to provide feedback allowing the engine to partially automate recurring tasks like e.g. mappings between ontological data and UML.

Ontology-based engines are the inverse of ontology-driven engines. In this approach the ontology is used as a passive component, only needed to verify and lookup data.

The benefits of both approaches will be demonstrated in the next section.

2 Experiments

Our experiments are situated in the context of applications concerned with computer hardware, e.g. applications that support the sale of computer parts, stock management, hardware testing, support and helpdesk application for computer hardware etc.

In the ontology-driven experiment we will assume we have to build a new application for a help-desk. The application is supposed to assist the help-desk officer in going through a number of standard questions concerning the computer hardware. Depending on the feedback, a certain solution is proposed. It is obvious that we want this application to have a model of as many computer parts and relations between them as possible.

To built this application, the ontological engine will propose the engineer with known information about computers and their parts, subclasses etc. Based on the feedback of the engineer, UML diagrams will be automatically generated in Rational Rose.

In the ontology-based experiment, we will perform a reverse engineering of some diagrams of an existing help-desk application, with the same functionality and structure as in our previous example. With some exceptions, like unknown terms and concepts and some relations in the UML diagrams that are not compliant to the data in our ontology.

Both approaches are based on the idea that, the more the ontological engines are used, the more complete and detailed the ontologies will become as the ontology is enriched and completed during its use. The profits will only be noticeable after a certain period of time.

2.1 Setup of Experiments

2.1.1 Wordnet

Wordnet is a semantic word database based on psycholinguistic principles. Wordnet groups synonymous word senses into single units ("synsets"). Noun senses are organized into a deep hierarchy, and the database also contains part-of-links, antonym links and others. Approximately half of WordNet synsets have brief informal definitions [KL, MBF$^+$93].

For these experiments, the content of WordNet 1.6 is sufficient, however we are aware of its shortcomings. For future, more complex experiments we are currently looking at more appropriate content providers.

2.1.2 Rational Rose

Rational Rose is one of the most widely known and used CASE tools available at the moment. It integrates a graphical component to draw UML diagrams, a source code generator that generates amongst others C++, java and VB code based on the UML diagrams, and a reverse engineering component that allows the building of UML diagrams based on source code. Up until a certain level, round trip engineering is possible. For the following experiments we are only interested in the class diagram features of Rational Rose.

2.2 Overview

2.2.1 Ontology-driven experiment

An ontology-driven engine uses the ontological data to drive the creation of all the artifacts in the SDLC (Software Development Life Cycle), i.e. diagrams, classes, patterns, frameworks, components, etc. This means that, based on the contents of the ontology suggestions will be made to the user. That way previously made models and decisions can be reused accordingly. Thus we could state that an ontology-driven approach uses the ontology as an active component.

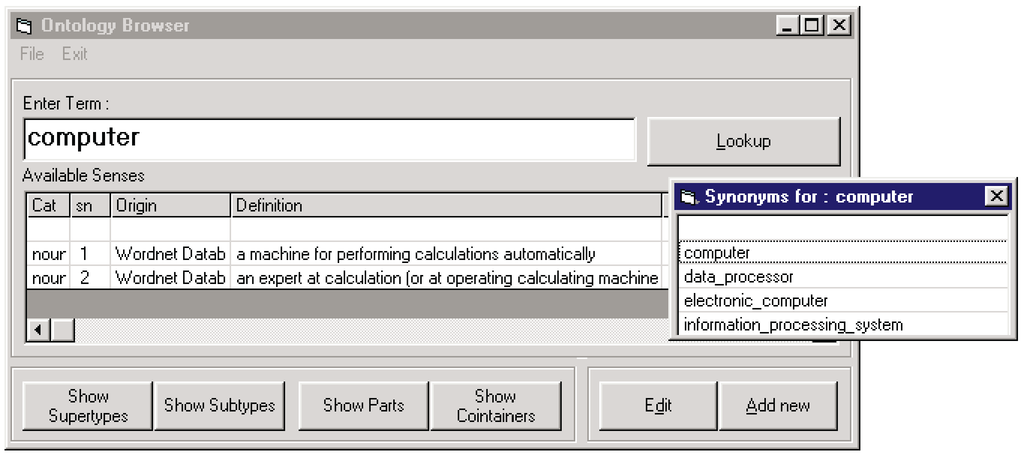
When software engineers, working within the same problem domain, start working on a certain project, very often (not to say always) a feeling of "having done all this already before" rears its head. This would not be such a problem if they were able to retrieve those pieces of diagrams, source code and components and could reuse them.

Nowadays this is where the shoe pinches. Diagrams and models are built, converted to source code and maybe some kind of documentation is written. After this, everything is put into the cupboard (often referred to by CASE-vendors as repositories). Nobody knows how or where to find the things they need, so the next time someone wants to build a similar application, the whole process starts all over again from scratch. Under the most favorable circumstances the engineer can build on his own or a colleague's experience. But most of the time, a worst case scenario occurs, where neither of these experiences are available. Thus starting another lengthy SDLC, filled with obstacles.

Using the ontology as an "active shared enterprise memory", the SDLC can be vastly optimized and improved. Thus minimizing the previously mentioned problems.

To fulfill his task, the software engineer will startup the ontological engine from within Rose 98. As shown in figure 1, the engine will open a browser window which entry point of interaction. From within this browser, the engineer will have access to all the information contained in the ontology, e.g. synonym lookup, browsing part-of (figure 2) and is-a hierarchies (figure 4) , etc.

For the example at hand, the synonyms of computer are displayed. The engineer chooses to model the concept of a computer, and selects the concept computer in the sense of "a machine for performing calculations auto-



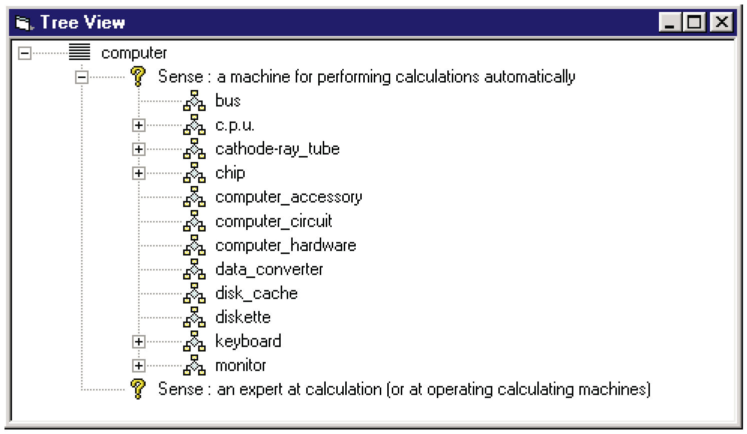
Enter Term : computer

Available Senses

Cat	sn	Origin	Definition
nour	1	Wordnet Datab	a machine for performing calculations automatically
nour	2	Wordnet Datab	an expert at calculation (or at operating calculating machine

computer
data_processor
electronic_computer
information_processing_system

Figure 1: View on the browser



```
computer
    Sense : a machine for performing calculations automatically
        bus
        c.p.u.
        cathode-ray_tube
        chip
        computer_accessory
        computer_circuit
        computer_hardware
        data_converter
        disk_cache
        diskette
        keyboard
        monitor
    Sense : an expert at calculation (or at operating calculating machines)
```

Figure 2: Partof relations in the ontology

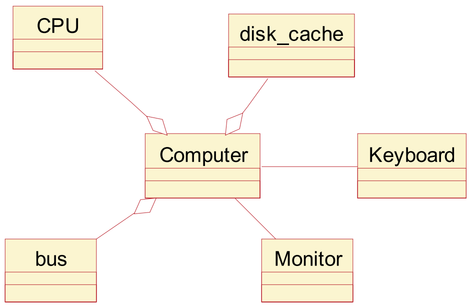
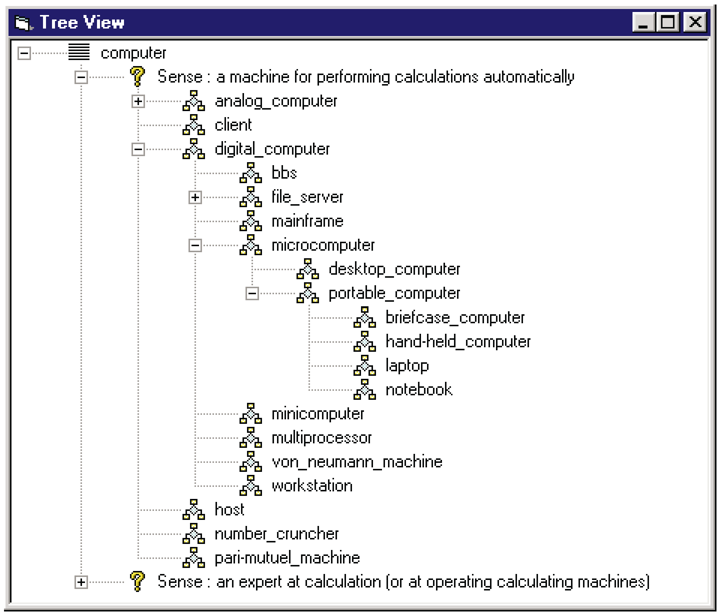


Figure 3: Partof relations in Rational Rose 98



```
computer
  Sense : a machine for performing calculations automatically
    analog_computer
    client
    digital_computer
      bbs
      file_server
      mainframe
      microcomputer
        desktop_computer
        portable_computer
          briefcase_computer
          hand-held_computer
          laptop
          notebook
      minicomputer
      multiprocessor
      von_neumann_machine
      workstation
    host
    number_cruncher
    pari-mutuel_machine
  Sense : an expert at calculation (or at operating calculating machines)
```

Figure 4: isA relations in the ontology

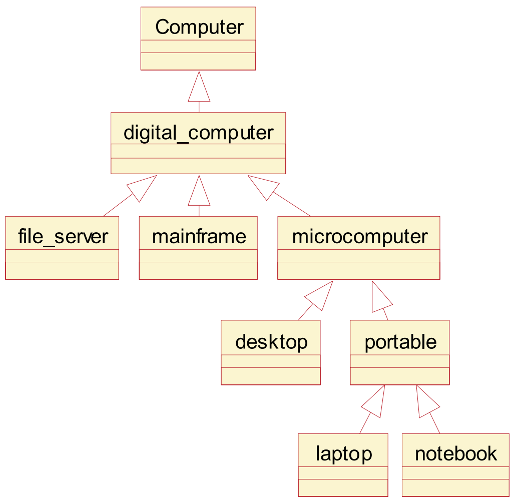


Figure 5: Subtype relations in Rational Rose 98

matically”. If he finds the two already available senses not suitable for his purposes, he is able to extend the ontology.

Next he browses the parts-of (or meronyms) of the computer concept in the selected sense. Here he can decide which parts he wants to model (figure 2), and how he wants to represent them in Rational Rose. This step is necessary, since it is possible to represent a meronymy relationship in UML as an attribute, an aggregation, a composition, or an association. A possible result is shown in figure 3. The definitions of every modeled concept, coming from the ontology, is also copied into the documentation section of the classes. That way providing an initial (naive) means for (semi-)automatical documentation of class diagrams. We are currently exploring the possibilities and potential of this feature. The same process is followed to model the is-a (or holonyms and hypernyms) relationship (figures 4 and 5.

2.2.2 Ontology-based experiment

An ontology-based engine uses the ontological data as a kind of reference work. The engine will try to map the semantical information contained in a UML class diagram onto the information contained in the ontological data.

This mapping enables the engine to verify and validate the information contained in the UML diagrams. Thus, in this case, we could state that an ontology-based approach uses the ontology as a passive component.

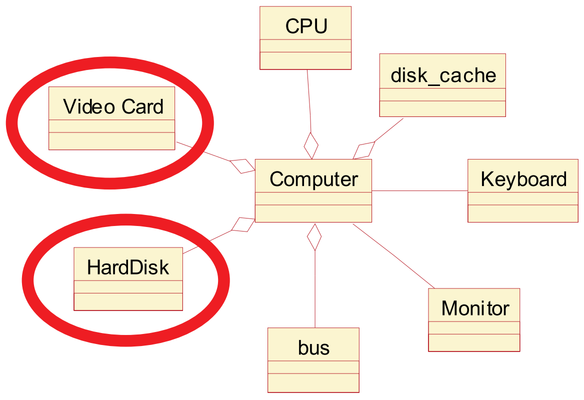


Figure 6: Extended partOf hierarchy in Rational Rose 98

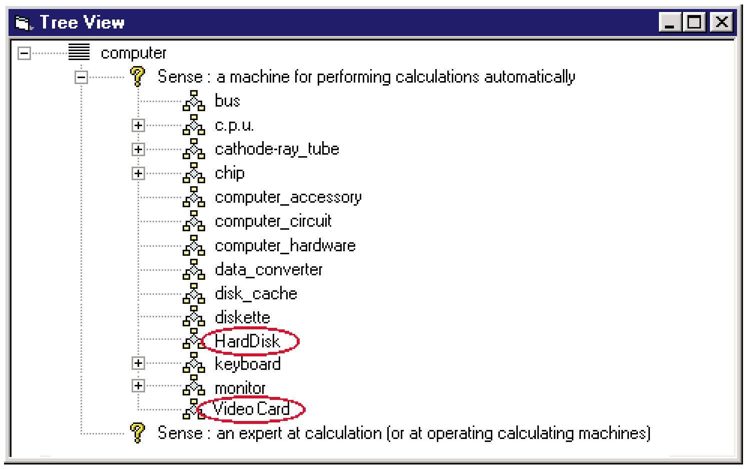
The benefits of this approach lay in the fact that semantic conflicts can be detected automatically. Oblivions, ambiguities and conflicts are reported to the software engineer. This allows him to take appropriate steps immediately instead of in a later phase of the SDLC, where expensive and time-consuming remodeling and rebuilding are unavoidable.

The first part of this task can be completed without any interventions from the software engineer. It will mainly consist of verifying whether the information contained in the UML diagram can be mapped onto the corresponding ontological information. Two kinds of mapping errors can occur. On the one hand a chunk of information could be found in the UML diagram that cannot be found in the ontology. On the other hand, it is possible that the information in the UML diagram is conflicting according to the ontology.

In the first case, the engineer should add the new information to the ontology. In the latter case, the engineer should decide which party (UML diagram or ontology) is the correct one. The erroneous party should be corrected.

If the ontology would contain links towards projects that use this erroneous data, engineers maintaining those applications should be warned about the (potential) failure.

Figure 6 shows part of a UML diagram on which information has been found, currently not present in the ontology. When checking this is mentioned to the engineer. Figure 7 shows the corrected ontology.



```
Tree View
computer
  Sense : a machine for performing calculations automatically
    bus
    c.p.u.
    cathode-ray_tube
    chip
    computer_accessory
    computer_circuit
    computer_hardware
    data_converter
    disk_cache
    diskette
    HardDisk
    keyboard
    monitor
    Video Card
  Sense : an expert at calculation (or at operating calculating machines)
```

Figure 7: Extended partOf hierarchy in the ontology

One should note that whenever models are developed using the *ontology-driven* approach, and on condition they were not altered manually, no conflicts should occur and no oblivions should be detected.

3 Conclusion

In this paper we have demonstrated two ways to integrate ontologies into a CASE-tool, enhancing the possibilities and extending the functionalities of this tool.

As the results of the experiments indicate, it is now possible to move on from basic ontology research (representation and formalization issues) towards research focussed on advanced applications of ontologies.

It is clear that both proposed approaches require an initial effort to fill in the ontology with domain specific data. The longer the ontology is used, the more domain knowledge is present in the ontology, and the less effort will have to be spend in adding data to the ontology. Consequently development of applications, by making use of the proposed CASE tool enhancements, will take a lot less time and debugging effort as more and more (established and verified) knowledge is reused.

4 Acknowledgements

Thanks to Joke Reumers and to Wim Lybaert for proofreading the final draft of this paper.

We would especially like to thank Anyouzoa Njimolu Alain Gaetan for his enthusiastic support, his help with the experiments and his remarks after proofreading some drafts of this paper.

5 Future Work

In this paper we proposed to integrate an ontology in a CASE tool, to help the engineer in designing and verifying his model, allowing him to extend his ontology. Apart from this kind of integration, ontologies could be integrated into several other kinds of tools, and offer a lot of help there too. One could think about reverse engineering tools, where it is necessary to try to understand existing (legacy) applications, browsers and programming environments, etc.

Another research topic could be the formalization of use cases with the help of ontologies. When software engineers start developing applications, the first phase is knowledge acquisition. Together with the client all functionalities of an application are discussed and written down into use cases [JCJO92]. These use cases are the basic documents on which developers and programmers implement the application. This approach has one vast drawback: use cases are written down in natural language and are thus informal, very often ambiguous, incomplete and not automatically processable. This makes them hard to handle, hard to browse and hard to encode. By formalizing use case with the help of ontologies, this drawback could be remedied. By linking use cases to an ontology, we would have both the advantages of a natural language (i.e. readability of use cases for client) and the advantages of something formal to be able to (semi-)automatically do some verifications and checks. At the moment, some students are trying to find a solution to this topic at the Programming Technologies Lab of the Vrije Universiteit Brussels, Belgium.

Finally, reverse engineering tools could benefit from an integrated ontology. Tools like DUPLOC, developed at the university of Bern, Switzerland and extended versions of the Refactoring Browser in VisualWorks that are analyzing source code, could benefit from the use of ontologies. Especially lexical information like e.g. synonym-detection and conceptual information like e.g. kindOf relations, could mean a useful extension to such tools.

References

[EP98] Hans-Erik Eriksson and Magnus Penker. *UML Toolkit.* Wiley, 1998.

[FLGD87] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11), 1987.

[GG95] Nicola Guarino and Pierdaniele Giaretta. Ontologies and Knowledge Bases. IOS Press, Amsterdam, 1995.

[Gru92] T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In *Proceedings International Workshop on Formal Ontology*, 1992.

[Gru94] T. R. Gruber. Towards Principles for the Design of Ontologies Use for Knowledge Sharing. In *Proceedings of IJHCS-1994*, volume 5 of 6, pages 907–928, 1994.

[JCJO92] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard. *Object-Oriented Software Engineering, A Use Case Driven Approach.* Addison-Wesley Publishing Company, revised fourth printing edition, 1992.

[KL] Kevin Knight and Steve K. Luk. Building a Large-Scale Knowledge Base for Machine Translation. knight@isi.edu, luk@isi.edu.

[Lab98] Princeton University Cognitive Science Lab. Wordnet 1.6. online thesaurus, 1991-1998.

[MBF+93] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. *Introduction to WordNet: an on-line lexical database, 5 papers on WordNet.* University of Princeton, 1993.

[MOF] Meta-Object Facility Tutorial. http://www.dstc.edu.au/Research/Projects/MOF/Tutorial.html.

[ST99] William Swartout and Austin Tate. Ontologies. *IEEE Intelligent Systems*, pages 18–19, January/February 1999.

[Usc98] Mike Uschold. Where are the Killer Apps? In *ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods*, 1998.