

# Design of a Knowledge-Based System with A UML-Based Ontology Model

Xin Wang    Howard J. Hamilton    Christine W. Chan

Department of Computer Science  
University of Regina  
3737 Wascana Parkway, Regina, SK, Canada, S4S 0A2

{wangx, hamilton, chan}@cs.uregina.ca

**Abstract.** An ontology is a comprehensive knowledge model which enables the developer to practice a “higher” level of reuse, knowledge reuse. In order to achieve knowledge reuse, we propose forging a closer mapping between the knowledge and software models in the development process. In this paper, we first present UML as an ontology modeling language and then describe the process of deriving the system model from the UML-based ontology model.

**Keyword**    Ontology    Knowledge-based System    Domain model

**Classification:** 2 year’s work

## 1. Introduction

To support software reuse, we need to move our focus from modeling a single system to modeling families of systems by identifying “reusable” solutions. An important benefit of using an ontology during development is that it enables the developer to practice a “higher” level of reuse than is usually the case in software engineering, that is, knowledge level reuse. Moreover, an ontology enables the developer to reuse and share application domain knowledge using a common vocabulary across heterogeneous software platforms and programming languages. It also enables the developers to concentrate on the structure of the domain and the task at hand and prevents them from being distracted by implementation details.

Currently, researchers in the knowledge-based system community are exploring a variety of languages and platforms to help acquire and represent ontologies, e.g., KIF-based language such as Ontolingua [1], KL<sub>ONE</sub>-based language such as LOOM [2], and knowledge acquisition tool such as Protégé[3]. Knowledge representation paradigms underlying these languages and tools are diverse: frame-based, description logic, first and second order predicate calculus, object-oriented, etc. [4]. However, these language and tools lack both a wide user communities outside AI research laboratories and a systematic approach to guide the ontology model into the software model of the knowledge-based system.

The objective of our study is to investigate stronger coupling between the knowledge engineering and software engineering phases of development of knowledge-based systems to make better reuse of the knowledge. This paper presents our investigation of using a UML-based domain ontology model to facilitate the design of knowledge-based system. We propose a two-step approach to the design of a knowledge-based system, consisting of ontology model, domain model and system model. The ontology model characterizes the overall knowledge relevant to a family of systems, the domain model describes the information relevant to system building in the domain and is thus an instance of the ontology model. The system model finally instantiates the domain model for a specific system. In the following discussion, the domain of petroleum remediation techniques is used as our application example.

The paper is organized as follows: Section 2 briefly explains our rationale for integrating the knowledge and software models. Section 3 describes how a UML-based ontology can be used to build a specific system model. Some concluding remarks are given in section 4.

## 2. Integration of Knowledge and System Model

An *ontology* is an explicit specification of a conceptualization [5]. An ontology refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of

explicit assumptions regarding the intended meaning of the vocabulary [5]. In the simplest case, an ontology describes a hierarchy of concepts related by subsumption relationships; in more sophisticated cases, suitable axioms are added to express other relationships between concepts and to constrain their intended interpretations.

The approach that we present in this paper is intended to enable users and software engineers better understand an ontology model by using a language that they can understand. Better comprehension will allow the ontology to be validated more quickly and completely. At the same time, the model needs to include all the knowledge required for problem solving so that developers can use it for constructing the software system that is to solve the problem. Our approach is based on reconsidering the stages in the development process. Ideally, the software development process should consist of two completely separate stages: a Knowledge-centered Stage, which is related to the domain knowledge and users' needs, and a System-centered Stage, which finds a computer-based solution to satisfy the users' needs. From this perspective, the ontology model is mainly related to the first stage, while the software model is related to both stages but emphasizes the second. At present, ontology modeling cannot produce good software because of its lack of software-centered support. Hence, a bridge is needed to cross the gap in modeling between the ontology and a real knowledge-based system.

The most common formalisms used to represent ontologies are the Knowledge Interchange Format (KIF) and KL\_ONE style knowledge representation languages. KIF provides a Lisp-like syntax for expressing sentences of first order predicate logic and also provides extensions for representing definitions and meta-knowledge [6]. KIF is a highly expressive and low-level language for representing ontologies. KL\_ONE principally provides a language for the explicit representation of conceptual information based on the idea of structured inheritance networks [7]. KIF and KL\_ONE are little known outside the knowledge engineering community, as are their descendants [1][2]. Furthermore, since they are based on the logic, they are difficult to learn and knowledge encoded in them is difficult for software engineers to understand.

Protégé-2000 [3] is an extremely nice software package and the best known existing software that we know of for creating ontologies. Protégé assists software developers in creating and maintaining explicit domain models, and in incorporating those models directly into program code. The construction of the implemented ontology model has been described in [8]. However, while Protégé can be used for constructing an ontology model, the implemented representation of the ontology is usually not well understood by the software engineer. Hence, the implemented ontology cannot be directly translated into a system model, and a bridge is needed between the ontology model and a knowledge-based system. That is, we need a human-readable representation language to represent ontology, and a systematic method to reuse the ontology to help generate the system model.

### 3. Using the UML-based Ontology

In this section, we propose a two-step approach to the design of a knowledge-based system, consisting of ontology model, domain model and system model. The whole process of using a domain model as a bridge between the ontology model and the system model is illustrated in Figure 1. As we mentioned in the introduction, the *ontology model* characterizes the overall knowledge relevant to the

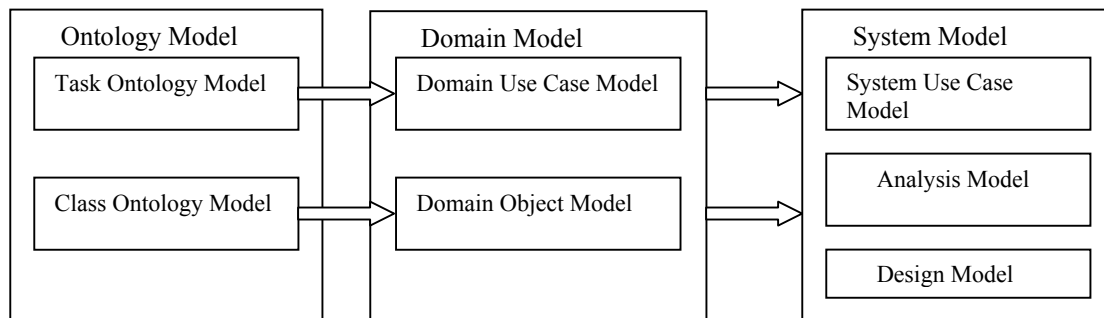


Figure 1. Domain model as the bridge between the ontology model and the system model

systems, the *domain model* describes the information relevant to system building in the domain and is thus an instance of the ontology model. The *system model* finally instantiates the domain model for a specific system. The ontology model includes two parts: a task ontology model and a class ontology model. The *task ontology model* provides a systematic vocabulary of the terms used to solve problems associated with tasks in this domain. The *class ontology model* describes the classes, attributes and the relationships in this domain. Both of them are application-independent. The domain model is an application-dependent model which serves as a bridge between the ontology model and the real system model. It consists of a domain use case model and a domain object model. The *domain use case model* embodies the intention of the system and a set of steps to achieve that intention. The *domain object model* presents the static structure of the application, including the information of the classes, and their relationships. The domain use case model and the domain object model are derived from the task ontology model and the class ontology model, respectively. The system model includes three parts: system use case model, analysis model and design model. The system model is application-dependent, which describes the whole application from the different views. It is generated from the domain model.

In the following discussion, first we introduce UML with OCL as the ontology modeling language, then the approach of limiting the ontology to the system model will be presented.

### 3.1 Specifying the ontology by using UML-OCL

Unified Modeling Language (UML) can be used to visualize, specify, construct, and document the artifacts of a software-intensive system [9]. However, despite its expressiveness, a UML graphical model, such as a class model, cannot by itself support a precise and complete specification. Usually, users of the UML can use Object Constraint Language (OCL) to specify constraints and other expressions attached to their models. OCL is a formal language that has the expressiveness of first order logic and is easy to read and write [9]. Henceforth, the combination of UML with OCL is referred to as UML-OCL.

The four basic elements identified in the ontology model are classes, relationships, constraints and processes. A detailed discussion of how to represent the ontology in UML-OCL is presented in [10]. To summarize the results in [10], the concepts in the domain ontology can be represented by the class notation, the relationships among the classes can be represented by the class diagrams, the task or domain process can be represented by using the activity diagram either from the activity view or from the operation view and the OCL expression can be used to represent the constraints. An additional advantage of the UML-OCL formalism is that each symbol in the UML notation is well-defined with well-understood semantics. Hence a model represented in UML-OCL can be easily interpreted unambiguously by other people or tools.

### 3.2 Mapping the ontology model into the system model

Generally, an ontology model comprises the task ontology and the class ontology. The *task ontology* characterizes the domain tasks for the domain. The *class ontology* characterizes the domain static knowledge, including the classes, attributes, constraints and the relationships in the domain. The domain tasks manipulate these knowledge items. The ontology model is application-independent. These application-independent components are useful building blocks for different applications and even for different but similar domains, but they are not easily adaptable and when reused, they may carry extra baggage or extra functionality due to their general nature [11]. For example, for developing the ontology for the petroleum contaminate remediation selection domain, the knowledge of the different media, including water, soil, air and groundwater were acquired and represented. Each particular medium is associated with specific chemical contaminations, standards, remediation methods and processes. However, when a particular application knowledge-based system is developed, usually only part of the knowledge will be used. For an expert system that supports eliminating the contaminants in the air, for example, only the part of the ontology model that concerns the medium of air will be used. That is, the generic or application-independent ontology model needs to be trimmed or tailored for the requirements of a specific system model.

**Step1: Mapping the ontology model into the domain model.** A domain model is introduced to help limit the general ontology model. A domain model organizes the knowledge that exists in the ontology model and the requirements for the new system, all notations in the domain model are represented by the standard UML notation with the slash. It is an application-dependent model which serves as a bridge

between the ontology model and the real system model. It enhances the software engineers' understanding of the problem tackled by the software. The requirements for the application can be derived from the domain model and it may also form the basis for reengineering. The creation of precise, well-scoped domain models for the system is of central importance and should benefit from software-engineering principles that make the models both explicit and manipulable. A domain model consists of a domain use case model and a domain object model. The domain use case model embodies the intention of the system and a set of steps to achieve that intention. It includes two elements: domain actors and domain use cases. The domain object model presents the static structure of the application, including the information of the classes, and its relationships. The domain use case model and the domain object model are derived from the task ontology model and the class ontology model, respectively.

**Step2: Mapping the domain model into the system model.** A system model includes three parts: a system use case model, an analysis model and a design model. In this model, the system functional requirement and static object structure are not in separate sub-models any more. They will work together to describe different views of the system.

The *system use case model* describes how the expert system is related to the users and experts. It is based on the domain use case model. The system use case model contains the actors and the system use cases. The model is defined in terms of use cases. A use case describes the behavior of the system, including the interaction between the actor and the system.

An *analysis model* is an object model that describes realization of the use cases and serves as an abstraction of the design model. The goal of an analysis model is to create a preliminary mapping of the required behavior onto modeling elements in the system. In most cases, it omits details of the design model to provide an overview of system functionalities. The analysis model eventually transitions into the design model, and the analysis classes directly evolve into design model elements.

The *design model* precisely describes the use case realizations combined with the user interface information and the data structures. A use case realization represents the design perspective of a use case. It is an organization model element used to group a number of artifacts related to the use case design. Use cases are separated from use case realizations so each can be managed individually and so the design of the use case can be changed without affecting the baseline use case. For each use case in the use-case model, there is a use case realization in the design model with a dependency (stereotyped <<realizes>>) to the use case. For each use case realization, one or more interaction diagrams are used to depict objects and their interactions.

In the system model, the whole system can be viewed from different perspectives [6]. The use case view of a system encompasses the use cases that describe the behavior of the system as seen by its end users, analysts, and testers. This view primarily includes the functional requirements of the system, i.e., the services that the system should provide to its end users. The design view of a system encompasses the classes, interfaces and collaborations that form the vocabulary of the problem and its solution. The analysis model is based on the domain object model and the system use case model. The design model is dependent on the analysis model, which is concerned about the details of implementation. The domain object model provides the static object structure for the analysis model, including the class hierarchy and the relationships among the classes. The system use case model is derived from the domain use case model. It consists of the dynamic or behavioural aspects of the system, including use case diagrams and activity diagrams. The analysis model includes both static and dynamic aspects of the system.

## 4. Conclusion

To support software reuse, we need to move our focus from modeling a single system to modeling families of systems by identifying "reusable" solutions. An ontology is a knowledge model that can be used to enhance knowledge sharing and reuse. In software engineering, models enable developers to visualize a system and to specify the structure or behavior of a system. However, even though both software engineers and knowledge engineers use model-based methodologies, they often employ different modeling languages and tools. Hence, their models are often not directly translatable.

This paper investigates bridging the gap between ontological models and system models with a domain model. Our study reveals that UML and OCL are good representation mechanisms for the domain ontology. The two-step approach described in this paper can be applied not only in the knowledge-based system, but also various other kinds of systems. The necessary condition of applying this method is the

ontology is broad enough to cover the domain of application and it is available in a human-readable format.

Future work should investigate whether the UML-OCL-based representation has equivalent representational power to existing ontology representation formalisms. The proposed approach should also be evaluated in the context of a development of a full knowledge-based system.

#### References:

1. Farquhar, A., Fikes, R., Rice, J., The Ontolingua Server: A Tool for Collaborative Ontology Construction, *Proceedings of KAW96*, Banff, Canada, 1996.
2. MacGregor, R., Inside the LOOM classifier, SIGART bulletin, #2(3):70-76, June, 1991.
3. H. Eriksson, R. W. Ferguson, Y. Shahar, & M. A. Musen. Automatic Generation of Ontology Editors. *Twelfth Banff Knowledge Acquisition for Knowledge-based systems Workshop*, Banff, Alberta, Canada, 1999.
4. Corcho, O., Gomez-Perez., A., Evaluating Knowledge Representation and Reasoning Capabilities of Ontology Specification Language, Proceedings of ECAI-00 Workshop on Applications of Ontologies and Problem-Solving Methods, 2000.
5. Guarino, N. and Giaretta, P. 1995. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (ed.), *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*. IOS Press, Amsterdam: 25-32.
6. Genesereth, M.R., Gikes, R.E., Knowledge Interchange Format Version 3.0 Reference Manual, 1992.
7. Brachman, R. J., Schmolze, J. G., An Overview of the KL-ONE Knowledge Representation System, *Cognitive Science* 9, 1985, pp.171-216.
8. Chen L., Chan C.W. Ontology construction from knowledge acquisition. Pacific Knowledge Acquisition Workshop (PKAW 2000), Sydney, Australia, December, 2000
9. Booch G., Rumbaugh J., Jacobson I. *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
10. Wang X., Chan C.W. Ontology modeling in UML. *Proceedings of 7th International Conference on Object-Oriented Information Systems (OOIS2001)*, Springer-Verlag, Calgary, Canada, Aug, 2001:59-68.
11. Jorge L. Diaz-Herrera, Domain Engineering, *The Handbook of Software Engineering and Knowledge Engineering*, 2001 (from <http://www.ksi.edu/seke/hand.html>).