Automated Ontology Construction from Scenario Based Software Requirements Using Clustering Techniques

Mohammad Moshirpour[1], Seyedehmehrnaz Mireslami [1], Reda Alhajj[2], Behrouz H. Far[1]

[1] *University of Calgary, Department of Electrical and Computer Engineering, 2500 University Dr NW, Calgary, AB, Canada, T2N 1N4*

[2] *University of Calgary, Department of Computer Science, 2500 University Dr NW, Calgary, AB, Canada, T2N 1N4*

{mmoshirp, smiresla, alhajj, far}@ucalgary.ca

Abstract

Ontologies have been utilized in many different areas of software engineering. As software systems grow in size and complexity, the need to devise methodologies to manage the amount of information and knowledge becomes more apparent. Utilizing ontologies in requirement elicitation and analysis is very practical as they help to establish the scope of the system and facilitate information reuse. Moreover ontologies can serve as a natural bridge to transition from the requirements gathering stage to designing the architecture for the system. However manual construction of ontologies is time consuming, error prone and subjective. Therefore it is greatly beneficial to devise automated methodologies which allow knowledge extraction from system requirements using an automated and systematic approach. This paper introduces an approach to systematically extract knowledge from system requirements to construct different views of ontologies for the system as a part of a comprehensive framework to analyze and validate software requirements and design.

Keywords: Software requirements; Scenario-based software engineering, Ontology, K-means clustering, Emergent behavior

1. Introduction

The goal of this research is to establish a comprehensive framework to analyze and validate software requirements and design documents [1-6]. It is suggested in the literature that detecting unwanted behavior prior to the implementation phase is about 20 times less costly than finding them during the deployment phase [7]. However manual review of requirements is usually inefficient and error prone, thus devising automated methodologies to analyze software requirements, architecture and design documents will be greatly beneficial. In [2] we demonstrated that using ontology in these methodologies can greatly increase the their efficiency and accuracy.

As software systems grow more complex, software engineers and developers need to devise methodologies to manage the amount of information and knowledge. Using Ontologies is an efficient solution for this issue as ontologies provide means to store the knowledge generated during the software development process [8]. However manual construction of ontologies, particularly for larger systems, can add a great deal of overhead to the software development process and it is error prone and subjective. Therefore it is highly desirable to devise methodologies to automate the process of ontology construction.

There is no consensus what should constitute software requirement and design ontology. As part of our model based software validation framework we have devised two distinct views of the ontology, namely the static and dynamic views.

The static view of ontology is a tree structure, where the elements are components of the system and are related to each other in this ontology based on their hierarchy within the system. The static view is currently constructed by the domain expert. In the static view, the domain expert classifies all system components into a finite set of categories.

The dynamic view of ontology, on the other hand, represents interactions among system components. This view describes the aspects of the system which can change with time. The dynamic view of the ontology is typically represented using finite state machines [9]. This view is also currently constructed by the domain expert. In this paper the goal is to automate construction of these two ontology views using clustering technique.

The organization of this paper is as follows: in Section 2, a brief background on ontologies and their use in software engineering is provided. Section 3 contains the case study of mine-sweeping robot. In Section 4 the automated methodology of ontology construction is outlined and demonstrated using the case study. Section 5 illustrates the developed tool for this methodology and conclusions and future work are provided in Section 6.

2. Background

Some background knowledge with regards to ontologies as well as scenario-based software engineering is provided in this section.

2.1. Ontology in Software Engineering

A commonly accepted definition for an ontology is as follows: "An ontology is an explicit and formal specification of a conceptualisation of a domain of interest" [10]. Thus two key points about ontology are established: (1) The conceptualisation is formal; therefore it allows reasoning by computers; and (2) A practical ontology is designed for a specific domain [10]. Ontology consists of concepts (also known as classes), relations (also known as properties), instances and axioms. A more concise definition of an ontology is as a 4-tuple <C, R, I, A>, where 'C' is a set of concepts, 'R' a set of relations, 'I' a set of instances and 'A' a set of axioms [11].

The importance of ontology has been recognized in a variety of areas such as semantic web applications, knowledge engineering, database design and integration [10, 12]. Depending on the application, different views of ontology as well as different ontology languages and tools are utilized [9, 13, 14].

Ontologies have been used in many different areas of software engineering [8, 9, 14-18]. As software systems grow more complex, software engineers and developers need to devise methodologies to manage the amount of information and knowledge. Ontologies provide means to organize the knowledge generated during the software development process [8]. The work presented in [8] describes a Semantic Reuse System (SRS), which uses ontologies that are represented using the knowledge representation language of the semantic web for software development knowledge reuse.

Ontologies have been utilized in Software Development Environments (SDE) [17]. An SDE is a computational system which provides support for the construction, management and maintenance of a software product [19]. SDEs provide integrated case tools, guidance to the software process and common repositories to the development teams. However SDEs do not provide any knowledge with regards to the domain of the related tasks [17]. To resolve this issue, the concept of SDE was extended by introducing domain and task knowledge to it in order to direct software engineers through software development phases; which lead to the development of the Domain-Oriented Software Development Environment (DOSDE) [20]. One of the main components for the infrastructure of DOSDE are ontologies.

The work in [16] presents examples of using ontologies in each stage of the software development lifecycle. For instance, in the requirement and design stage, [16] suggests the use of ontology for describing requirements specification document and formally representing requirements knowledge [21, 22]. The end-to-end use of ontologies in analysis, design and implementation, is appropriate for rapid application development. Moreover, using this approach the interoperability of ontologies with other components or applications is improved [16]. In addition, using a web-based knowledge representation format, allows engineers to discover sharable domain models from both internal and external repositories [16]. In addition, ontologies are utilized in software maintenance, testing, coding support and code documentation [16, 18].

In addition to building ontologies to address different needs in software engineering; some research suggests extracting ontologies from software design artifacts such as software object models [14]. The work of [14] suggests that ontologies are closely related to modern object-oriented software engineering. Therefore object oriented techniques can be adapted for the development of ontologies [14]. Furthermore this research states that there are similarities between descriptive ontologies and database schemas [14]. It can be concluded that artifacts from object-oriented design can be a starting point for building ontology for a given system. Some of the differences between ontologies and object-oriented modeling are as follows [14]:

- Ontology is theoretically based on logic and as such ontology allows automated reasoning. However object-oriented modeling does not include inference.
- Ontology and object-oriented modeling differ in the treatment of properties. Ontologies give properties priority, while object oriented modeling does not. For instance, ontologies allow the inheritance of properties.

However it is important to note that despite these differences, object-oriented modeling techniques and UML are accepted as practical ontology specification methods due to the similarities mentioned and their extensive use in the industry and the multitude of existing models in UML [14].

2.2. Different Views of Ontology

There are different classifications of ontologies in the literature [12, 23, 24]. For instance the work of [24] classifies ontologies based on the nature of the real-world issues as follows:

- *Static ontologies*: These ontologies are used to describe things that exist, their attributes as well as relations between them. In this classification, it is assumed that the world is made up of entities which are unique. Static ontologies are often represented using tree structures [25].
- *Dynamic ontologies:* These ontologies describe aspects of the modeled world which are capable of changing with time. To model dynamic ontologies it may be necessary to use finite state machines (FSM), Petri nets, et. Examples of terminology commonly included in this category include: process, state or state transition [9].
- *Intentional ontologies:* Such ontologies describe the aspects of the world of motivations, goals, belief or choices of the involved agents. Some common terms used in association with these types of ontology include aspect, object support or agent.

2.3. Ontology Tools

Ontology tools were developed later in the mid-1990s [13]. Ontology tools can be categorized as follows [13]:

- Tools which are specifically designed to work with a particular ontology language. Therefore these tools are developed as ontology editors for specific languages. Some examples of these kinds of tool are SWOOP [26] and KAON2 [27] which support OWL.
- Tools which have an extensible architecture and their model is independent of ontology languages. Such tools provide a set of ontology-related services and are easily extended to other modules to supply more functions. Some examples of these kinds of tools are Protégé [28], WebODE [29] and OntoEdit [30].

2.4. System Requirements

The way of expressing the system requirements depends heavily on the target domain. In this research the focus is on distributed software systems that may lack a centralized controller. Multi-agent systems are typical examples. We assume that the system requirements are defined using scenario-based specifications. Scenarios allow stakeholders to describe system's behavior using partial stories. Generally scenarios have been represented using two different notations in the literature; Sequence Diagrams (SD) of UML and Message Sequence Charts (MSC) standardized by the International Telecommunications Union (ITU) [31, 32]. In this paper we use the latter to express system requirements.

3. Case Study

To demonstrate the proposed methodology of automated ontology construction, the case study of a mine sweeping robot is used [5]. The robot's mission is to navigate through a maze-like course, which resembles the layout of the streets of a city, for which the robot has no map and has to investigate it based on utilizing its sensory information, i.e. ultrasonic and/or GPS data. At the same time, it has to identify and mark the location of mines. For this prototype version it is assumed that mines emit infrared signal which is detectable via the infrared sensor. In order to provide the robot with more computation power and additional control for the motors and different types of sensors, two multi-core CPU units are utilized. The units are built on separate boards connected via a simple but reliable connection protocol. The two CPUs interact using the client-server architecture. As there is no sophisticated operating system in charge of the control and scheduling of the processes and threads, the design of the robot must account for the proper management of all processes and their interactions in a logical manner

For the sake of simplicity, let's assume that the robot has only two sets of sensors: an ultra-sonic which is used for navigation purposes, and an infrared sensor to detect mines. Both of these sensors are connected to the client CPU. Thus the client receives signals from sensors, processes the message and sends them to the server CPU to act upon accordingly. The processes *Client Controller* and *Server Controller* are in charge of the motors responsible for the wheels on the left and right sides of the robot. Each process is also responsible for sending and receiving messages to and from the other. The server controller is also in charge of the motor of the mechanism which dispenses a flag on the location in which a mine has been detected. The *ULTS Motor Controller* process is responsible for the motor in charge of the rotation of the ultra-sound sensor which is necessary for navigation through the maze. The functionality of the robot is shown using scenarios as illustrated in Figures 1-3.

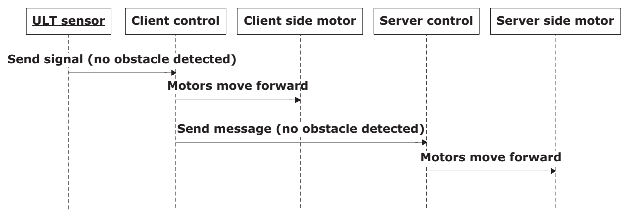

Figure 1. No obstacles detected; robot is moving forward

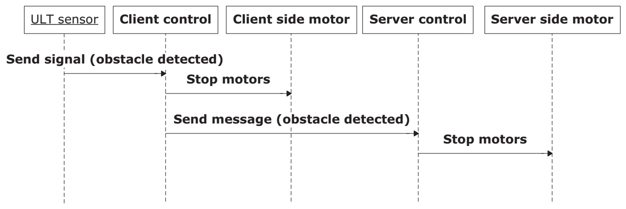

Figure 2. ULT sensor detects an obstacle; robot is halted

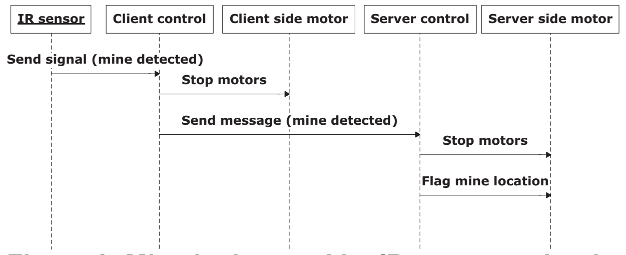

Figure 3. Mine is detected by IR sensor; robot is halted

4. Methodology

As demonstrated in [2], ontologies can be used to further automate the analysis of software requirements expressed using scenarios. To reduce overhead it is greatly beneficial to automate the process of ontology construction. Conveniently, system requirements expressed using scenarios are used to automate the ontology construction approach.

The process of automated construction of ontologies is done in two steps of clustering system components, and building the ontology based on the clustering results as outlined in this section.

4.1. Clustering System Components

In order to categorize the system's components, in this paper, it is proposed to employ clustering. This helps to better identify the states of the components in the behavior model state machines. Each component is represented by two variables: x and y. Variable x represents the number of messages sent and Variable y represents the number of messages received. The components and the associated variables are given as the inputs to the proposed algorithm shown in Figure 4.

The proposed algorithm, which is a customized clustering Algorithm, clusters the system components into five clusters based on the number of messages interacted by a component. First, system components are clustered based on Variable x into two clusters. The components with x=0 are clustered into Cluster 1 and the rest are further clustered into two clusters based on Variable y such that the components with y=0 are put in Cluster 2. Then, the remaining system components are clustered based on Variable x where the components with x=1 are clustered into Cluster 3 and the rest are further clustered based on variable y such that the components with y=1 are put in Cluster 4. Finally, the remaining components are clustered in Cluster 5. The algorithm outputs clusters, Cluster 1, Cluster 2, Cluster 3, Cluster 4 and Cluster 5.

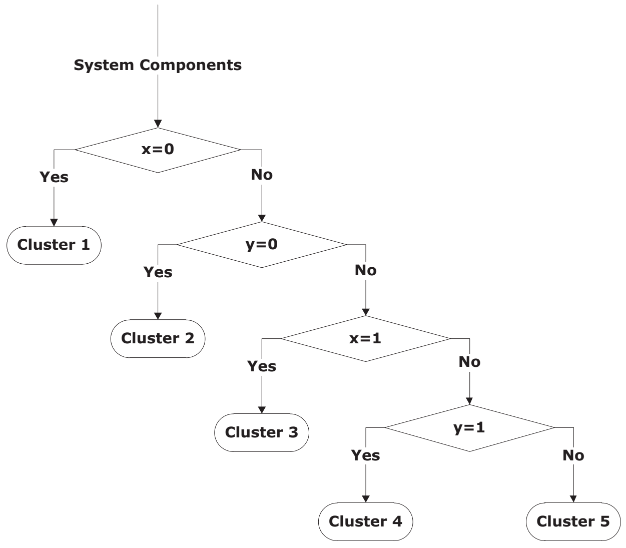

Figure 4. The flowchart of the customized clustering algorithm

Performing clustering for the system components helps to identify the type of each component as well as the type of state of each component. Therefore, Cluster 1 includes the motors since they only receives the signals. These components can only be final states in the behavior model state machines. The components that are clustered in Cluster 2 are considered as sensors that only send information and can only be represented as initial states in the state machines. Finally, the components in clusters Cluster 3, Cluster 4 and Cluster 5 are the controllers that send and receive several messages. The components cannot be initial or final states in the behavior models.

In order to show the effectiveness of the proposed algorithm, it is applied on the case study used in this paper (mine sweeping robot) when all the scenarios are considered. The components are modeled with x and y variables as shown in Table 1.

Table 1. Components modeled

Component Name	x	y
IR sensor	1	0
ULT sensor	2	0
Client control	6	3
Client side motor	0	3
Server control	3	3
Server side motor	0	3

These components are then given as the inputs to the proposed algorithm which is implemented in MATLAB R2010a. The output clusters are presented in Table 2.

Table 2. Clustering Results

Cluster Name	Components
Cluster 1	Client side motor, Server side motor
Cluster 2	IR sensor, ULT sensor
Cluster 3	-
Cluster 4	-
Cluster 5	Client control, Server control

As it can be seen in Table 2, the components in clusters Cluster 1 and Cluster 2 are motors and sensors, respectively while clusters Cluster 3, Cluster 4 and Cluster 5 contain control components.

4.2. Ontology Construction

To determine both the structure and they behavior of the software system, both the static and dynamic view of the ontology is needed.

The clustering results guide the domain expert to classify all system components into a finite set of categories. In our case study, the categories include: Sensors, Controllers and Motors as illustrated in Figure 5.

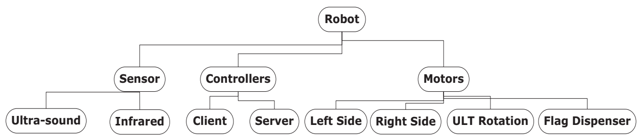

Figure 5. Static view of ontology for the mine-sweeping robot

As is the case with state machines, the dynamic view of ontology will have 3 different types of states as follows:

- *Start State* - Resembles the category of components that will only send messages, but do not receive any messages (e.g. Sensors in the case study).
- *Final State* - Resembles a category of components in which a type sending or receiving a message results in completing a task. For instance, in the case of the mine-sweeping robot, the motors are recognized as final states.
- *Transition State* - is a state which is determined to be not a start nor a final state. In this case study, the controllers fall under this category.

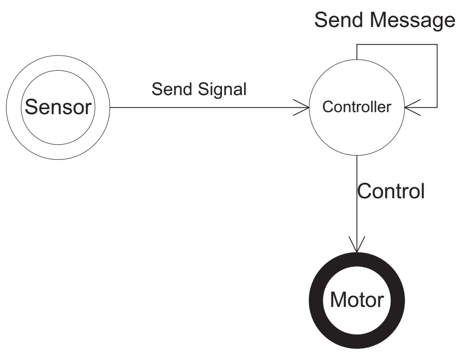

Figure 6. Static view of ontology for the mine-sweeping robot

Based on the clusters which were obtained in Section 4.1, the dynamic view of the ontology is constructed as shown in Figure 6. The constructed ontologies will be used to analyze system requirements and to detect unwanted behavior as shown in [2].

5. Developed Tool

The comprehensive framework to analyze and validate software requirements and design documents developed in this research [1-6] has been developed into a software package as shown in Figure 7. This project has been programmed in C# using Microsoft Visual Studio 2010.

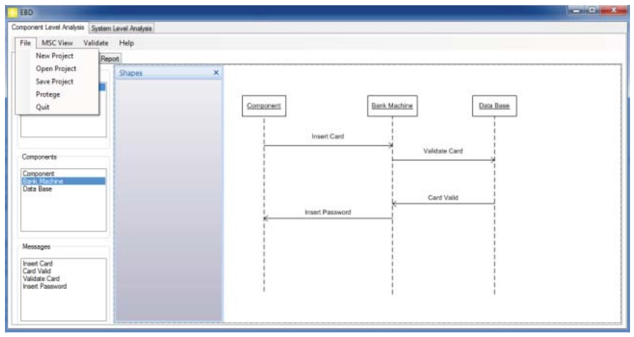

Figure 7. User interface

To start the analysis process, the user must either import an existing project or start a new project. Each project will consist of two separate files consisting of MSCs and ontologies. Upon importing the project, the user can browse through the MSCs using Microsoft Visio (Which is embedded within the GUI), and browse through the ontology using the ontology editor Protégé which is linked to this software. The user will then be able to click on validate and see the analysis results in the report which will be generated.

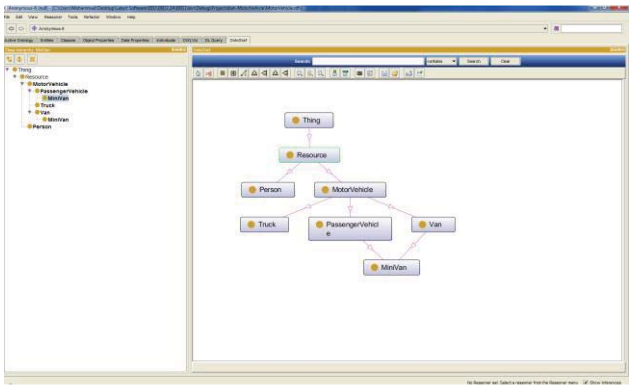

Figure 8. Protégé is used to illustrate ontolgoies

There were several ontology editors to choose from such as Protégé, TopBraid Composer, SWOOP, Jena, etc. The most important points in the selection criteria were as follows:

1. Export capabilities
2. UI linkable to external application
3. Popular and commonly used
4. Reproduction rights

By far, Protégé [28] is the most popular of the editors researched. It has export capabilities, and a UI that is linkable to external applications and one has right to reproduce and sub-licence protégé subject to third part claims. One downside is that Protégé is in Java which poses certain challenges with linking it with our application which is being developed in C#. The user can make new ontologies or open existing one using Protégé which is linked to this tool as shown in Figure 8. Using an existing plug-in named "OntoGraf" the user can view a graphical representation of the system's ontology.

6. Conclusions and Future Work

It is suggested in the literature that detecting unwanted behavior prior to the implementation phase is about 20 times less costly than finding them during the deployment phase [7]. This research provides a cost-effective solution for the development of distributed systems by establishing a comprehensive framework to analyze and validate software requirements and design documents in an automated manner [1-6].

Utilizing ontologies is an important part of this framework, as ontologies provide the means to classify and reuse information for software systems. However manual construction of ontologies adds a considerable amount of overhead to the project and it is error prone.

This paper provides a methodology to automate the process of ontology construction for a given software system using clustering techniques. Clustering is applied to software requirements which are expressed using scenario-based specifications.

This paper provides theoretical, empirical and practical contributions. Establishing a methodology for automated ontology construction using clustering techniques is the theoretical contribution. Developing a case study of the mine-sweeping robot has empirical implication and making a user friendly prototype tool to analyze software requirements based on ontologies is a practical contribution.

Future works includes further development of the prototype tool and developing larger case studies to test the efficiency of the automated ontology construction approach.

Acknowledgements

This research was partially supported by a grant from Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] M. Moshirpour, "Model-Based Detection of Emergent Behavior In Distributed and Multi-Agent Systems from Component Level Perspective," in *Department of Electrical and Computer Engineering*. vol. Master of Science Calgary: University of Calgary, 2011.

[2] M. Moshirpour, R. Alhajj, M. Moussavi, and B. H. Far, "Detecting Emergent Behavior in Distributed Systems using an Ontology-based Methodology " in *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011)*, Anchorage, Alaska, USA, 2011, pp. 2407 - 2412

[3] M. Moshirpour and B. H. Far, "Formal Verification of Lack of Existence of Illegal Scenarios in the Requirements of Distributed Systems," in *The IASTED International Conferences on Informatics 2010 Software Engineering and Applications (SEA 2010)*, Marina del Rey, USA, 2010.

[4] M. Moshirpour, A. Mousavi, and B. Far, "Model Based Detection of Implied Scenarios in Multi Agent Systems," in *Proceedings of the International Conference on Information Reuse and Integration*, Las Vegas, USA, 2010.

[5] M. Moshirpour, A. Mousavi, and B. H. Far, "Detecting Emergent Behavior in Distributed Systems Using Scenario-Based Specifications," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering* San Francisco Bay, USA, 2010.

[6] M. Moshirpour, A. Mousavi, and B. H. Far, "A Technique and Tool to Detect Emergent Behavior of Distributed Systems Using Scenario-Based Specifications," in *Proceedings of the International Conference on Tools with Artificial Intelligence*, Arras, France, 2010.

[7] R. F. Goldsmith, *Discovering Real Business Requirements for Software Project Success*. Norwood MA: Artech House, Inc., 2004.

[8] B. Antunes, N. Seco, and P. Gomes, "Using Ontologies for Software Development Knowledge Reuse," in *2nd Workshop on Building and Applying Ontologies for the Semantic Web (BAOSW 2007)* Guimarães, Portugal, 2007.

[9] F. Ruiz and J. R. Hilera, "Using Ontologies in Software Engineering and Technology," in *Ontologies for Software Engineering and Software Technology* C. Calero, F. Ruiz, and M. Piattini, Eds. New York: Springer, 2006, pp. 49-102.

[10] P. Warren, R. Studer, and J. Davies, "Introduction " in *Semantic Web Technologies - Trends and Research in Ontology-based Systems* J. Davies, R. Studer, and P. Warren, Eds. Chichester, West Sussex, England: Wiley, 2006, pp. 1-8.

[11] *Handbook on Ontologies. International Handbooks on Information Systems*, 2nd ed. New York: Springer, 2004.

[12] N. Guarino, "Formal Ontology and Information Systems," in *Proceedings of Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, 6-8 June 1998, pp. 3-15.

[13] O. Corcho, M. Fernández-López, and A. Gómez-Pérez, "Ontological Engineering: Principles, Methods, Tools and Languages," in *Ontologies for Software Engineering and Software Technology*, C. Calero, F. Ruiz, and M. Piattini, Eds. New York: Springer, 2006, pp. 1-48.

[14] W. V. Siricharen, "Ontologies and Object models in Object Oriented Software Engineering," *IAENG International Journal of Computer Science,* vol. 33, 13 February 2007 2007.

[15] C. González-Pérez and B. Henderson-Sellers, "An Ontology for Software Development Methodologies and Endeavours," in *Ontologies for Software Engineering and Software Technology* c. Calero, F. Ruiz, and M. Piattini, Eds. New York: Springer, 2006, pp. 123-151.

[16] H. Happel and S. Seedorf, "Applications of Ontologies in Software Engineering," in *2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), held at the 5th International Semantic Web Conference (ISWC 2006)* Athens, Georgia, 2006.

[17] K. M. de Oliveira, K. Villela, A. R. Rocha, and G. H. Travassos, "Use of Ontologies in Software Development Environments," in *Ontologies for Software Engineering and Software Technology* C. Calero, F. Ruiz, and M. Piattini, Eds. New York: Springer, 2006, pp. 275-309.

[18] N. Anquetil, K. M. d. Oliveira, and M. G. B. Dias, "Software Maintenance Ontology," in *Ontologies for Software Engineering and Software Technology* C. Calero, F. Ruiz, and M. Piattini, Eds. New York: Springer, 2006, pp. 153-173.

[19] A. W. Brown, A. N. Earl, and J. A. McDermid, *Software Engineering Environments: Automated Support for Software Engineering*. New York: McGraw-Hill, 1992.

[20] K. M. de Oliveira, A. C. da Rocha, A. Regina, C. Rocha, G. H. Travassos, and C. S. d. Menezes, "Using Domain-Knowledge in Software Development Environments," in *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering*, Kaiserslautern, Germany, 1999, pp. 180-187.

[21] B. Decker, E. Ras, J. Rech, B. Klein, and C. Hoecht, "Self-organized Reuse of Software Engineering Knowledge supported by Semantic Wikis," in *Workshop on Semantic Web Enabled Software Engineering (SWESE) held at 4th International Semantic Web Conference* Galway, Ireland 2005.

[22] V. Mayank, K. N, and M. Austin, "Requirements Engineering and the Semantic Web: Part II. Representation, Management, and Validation of Requirements and System-Level Architectures," University of Maryland, Technical Report February 12, 2004 2004.

[23] G. V. Heijst, A. T. Schreiber, and B. J. Wielinga, "Using Explicit Ontologies in KBS Development," *International Journal of Human-Computer Studies,* vol. 46, pp. 293–310, 1997.

[24] I. Jurisica, J. Mylopoulos, and E. Yu, "Using Ontologies for Knowledge Management: An Information Systems Perspective," in *Proceedings of 62nd Annual Meeting of the American Society for Information Science (ASIS99)*, Washington, D.C., 1999, pp. 482–496.

[25] J. Delgado, I. Gallego, S. Llorente, and R. Garc´ıa, "IPROnto: An Ontology for Digital Rights Management," in *The 16th Annual Conference on Legal Knowledge and Information Systems (JURIX-03)*, Amsterdam, The Netherlands, 2003, pp. 111-120.

[26] A. Kalyanpur, B. Parsia, and J. Hendler, "A Tool for Working with Web Ontologies," *International Journal of Semantic Web and Information Systems,* vol. 1, pp. 36-49, 2005.

[27] U. Hustadt, B. Motik, and U. Sattler, "Reducing SHIQ Description Logic to Disjunctive Datalog Programs," in *Proceedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning (KRR'04)*, Whistler, Canada, 2004, pp. Whistler, Canada.

[28] N. F. Noy, R. W. Fergerson, and M. A. Musen, "The knowledge model of Protege-2000: Combining interoperability and flexibility," in *International Conference on Knowledge Engineering and Knowledge Management (EKAW'00)* Juan-Les-Pins, France, 2000.

[29] J. C. Arpirez, O. Corcho, M. Fernández-López, and A. Gomez-Perez, "WebODE in a nutshell," in *AI Magazine*. vol. 24, 2003.

[30] 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)*, Sardinia, Italy, 2001.

[31] "ITU: Message Sequence Charts. Recommendation, International Telecommunication Union. ," 1992.

[32] "Unified Modeling Language Specification. Version 2. Available from Rational Software Corporation," Cupertino, CA, 2006.