Software Engineering Sub-Ontology for Specific Software Development

Wongthongtham, P., Chang, E. & Cheah, C.

School of Information Systems, Curtin University of Technology, Australia
{pornpit.wongthongtham, elizabeth.chang}@cbs.curtin.edu.au, chancheah@optusnet.com.au

Dillon, T.S.

Faculty of Information Technology, University of Technology Sydney, Australia
tharam@it.uts.edu.au

Abstract

In this paper we propose software engineering sub-ontology. We called it application-specific ontology, for specific software development. It enables remote team members browsing, searching, sharing, and authoring ontological data under the distributed software engineering projects environment. We transform explicit meaningful human knowledge into application-specific ontology, where knowledge structures and semantics are linked, and we go through a formal hand-shaking agreement establishing process before the semantic contents are updated in ontology repositories. The application-specific ontology is used for communication over project agreement to facilitate better, highly consistent communications and formalized domain knowledge sharing. We assume that object-oriented development is deployed in the distributed projects. The knowledge of object-oriented development formed in the application-specific ontology clarifies the object-oriented development concepts in a machine understandable form. Software agent, for example, can be utilised to extract information.

1. Introduction

As project teams engage more in projects that are geographically dispersed, inter-site communications become a key issue that often leads to miscommunications and misunderstandings. Carmel [1] and Van Fenema [2] suggest that traditional mechanisms, such as coordination and control frameworks, combined with appropriate integrated voice, data and video communication technology could be effective methods and tools for sharing and exchanging knowledge in projects. However, little is known of the success of using such communications technologies and methodologies in globally distributed software engineering projects [3]. Furthermore, it is already common knowledge that current development methodologies do not facilitate seamless and effective distribution of development tasks across multiple sites [4]. Therefore, this paper is to explore an application-specific ontology to help in communication in a globally distributed setting. We assume that object-oriented development is used in the projects.

This paper is a response to the challenge problems in the Multi-site Distributed Software Development (MDSD), which were highlighted in [4, 5]. Inter-site communication issues are especially key concerns in large-scale systems development, where the development teams reside in different locations [4-7]. In earlier work we proposed some solutions [4-8], and this paper examines one of these solutions, i.e. application-specific ontology, in detail. In the next section we briefly define ontology and in section 3 we briefly discuss ontology development. Our approach, application-specific ontology, will be explained in section 4 together with its staged process flows in section 5. We conclude this research and identify future work in section 6.

2. Software Engineering Ontology

In recent years, the notion of the 'Ontology' has been gaining prominence, in which Ontology provides the explicit formalization and conceptual specification of a domain or a general knowledge representation. The knowledge conceptualization is modelled in terms of notional entities and their inter-relationships. An ontology, or simply a conceptual knowledge map is only meaningful when it is associated with semantic data instances. From a machine perspective, these instances contain the actual data that are being queried. Therefore one of the main purposes of ontology is to enable communication between computer systems.

Proceedings of the 2005 29th Annual IEEE/NASA Software Engineering Workshop (SEW'05)
0-7695-2306-4/05 $20.00 © 2005 IEEE

We have proposed in [5-8] the software engineering ontology, which divided into two sub-ontologies: Generic ontology and Application-specific ontology. Generic ontology is a set of software engineering terms including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for the software development. It provides the vocabulary for the terms in software engineering. Application-specific ontology is different in that it is an explicit specification of object-oriented development in software engineering for a particular software development project. Set in a dedicated software development project environment, this domain ontology can be used for sharing intra-project communications of project knowledge that have been established by consensus agreement by different project members in dispersed locations. Both generic and application-specific ontologies establish intra and inter project knowledge agreements for enabling knowledge sharing. Application-specific ontologies foster a seamless and virtual: intra project environment for internal browsing, searching, sharing and authoring ontological project data across sites.

3. Software Engineering Ontology Development

A number of ontology representation languages currently exist; notable among these are Knowledge Interchange Format (KIF)[9], Simple HTML Ontology Extension (SHOE)[10], ISO standard for describing knowledge structures (Topic Maps)[11], Ontology Exchange Language (XOL)[12], Ontology Markup Language (OML)[13], Ontology Inference Layer (OIL[14], DAML+OIL[15]) and Web Ontology Language (OWL)[16]. We have chosen OWL because as it has now become the official W3C standard since the World Wide Web consortium released it in February 2004. There are many ontology development tools for creating ontology including Protégé[17], Oiled[18], OntoEdit[19], OntoLingua[20], and WebODE[21]. Protégé is the most widely known and used tool for creating ontologies and knowledge bases. Protégé is an open-source ontology-development tool developed at Stanford Medical Informatics. This provides an integrated environment to build and edit ontologies and check errors and inconsistencies (using a reasoner). Protégé has a number of different plug-ins including OWL Plug-in. The OWL Plug-in is a complex Protégé extension that can be used to edit and create OWL files and databases. To do querying, we are using RDQL - a query language for Resource Description Framework (RDF) in Jena model [22]. Jena 2 is a Java framework for writing Semantic Web applications and supporting a programmatic environment for OWL [23]. Data held imported ontology can be accessed, retrieved and modified using RDQL query language. We use UML to model ontology because there is a lack of graphical notation of modelling ontology. There are benefits for using the same paradigm for modelling ontologies and knowledge. Even standard UML cannot express advanced ontology features such as restrictions, cannot easily conclude whether the same property was attached to more than one class and cannot create a hierarchy of properties. However, it is a kind of agile modelling method for ontology design.

4. Software Engineering Sub-Ontology

A schematic overview of our approach is shown in Figure 1 illustrating a transformation of concepts to the ontology and its instances. The software engineering concepts are transformed to the ontology as domain knowledge in the form of man-machine-interoperable. Team members model their project system, i.e. project specific knowledge which is obviously based on domain knowledge or software engineering concepts. The project specific knowledge specially meets a particular project need and will be put into application-specific ontology as instance knowledge in form of machine-readable. Therefore, instance knowledge varies based on its use for a particular project. Once created, it is available to be shared among the teams through the Internet. All team members, regardless where they are, can query the semantic linked data instances and use them as the common communication and knowledge basis of raising discussion matters, questions, analyzing problems, proposing revisions or designing solutions, whatever.

Domain knowledge of 'object class' diagram concept in object-oriented development is presented in Figure 2 and 'use case' diagram concept is shown in Figure 3. Class diagram for some project design shown in Figure 4 can be transformed into application-specific ontology as instance knowledge.

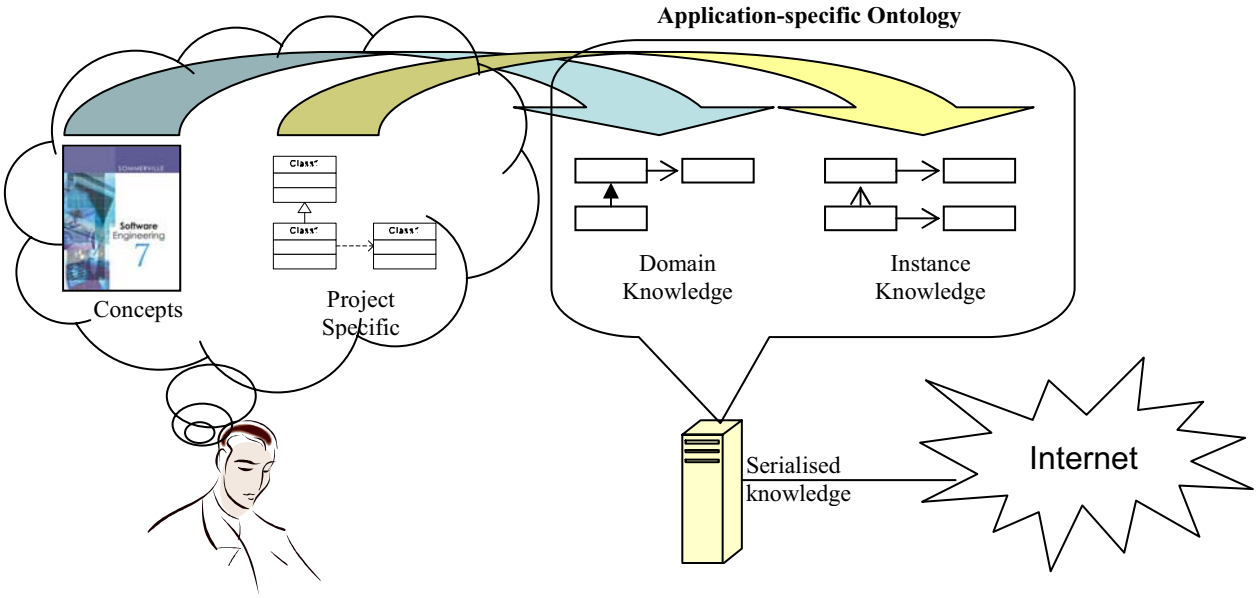

Figure 1: An overview of our approach

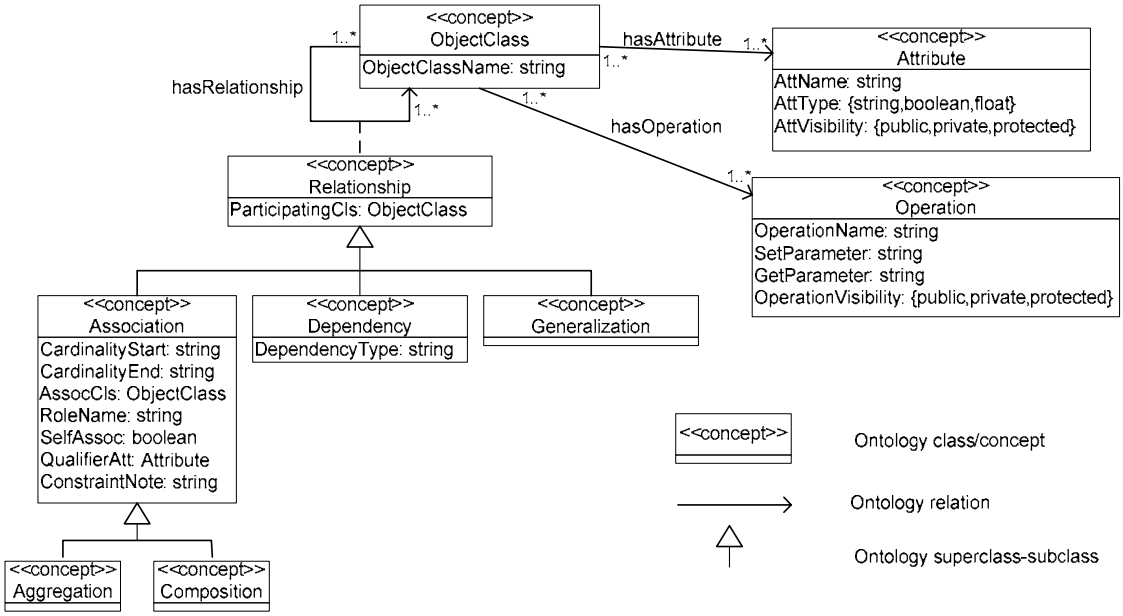

Figure 2: Meta-model of domain knowledge of 'object class' concept in object-oriented development

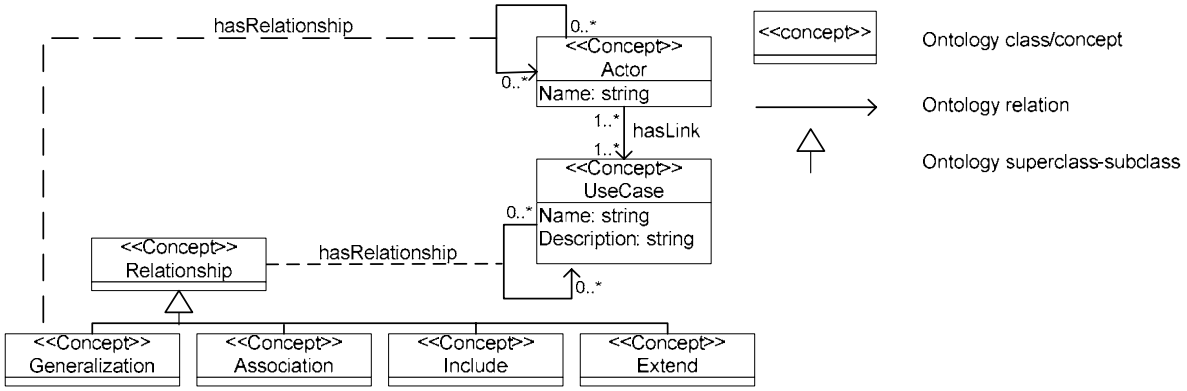

Figure 3: Meta-model of domain knowledge of 'use case' concept in object-oriented development

Ontology class is one of the most essential concepts in ontology modelling. Note that ontology class as it is defined in OWL (owl:class) is different from traditional UML class or object-oriented programming language class concept. In this paper, ontology class represents a concept for grouping resources with similar characteristics. For example in Figure 2 object class in UML is an ontology class (more precise – a class) that classifies many instances e.g. (in Figure 4) Customer, RentalCustomer, InsuranceRegisteredDriver, RentalRegisteredDriver. All object classes have some characteristics i.e. name, attribute, operation, and relationship which are represented by properties i.e. ObjectClassName (Datatype Property), hasAttribute (Object Property), hasOperation (Object Property), and hasRelationship (Object Property) respectively. These properties can have values that are of certain type; ObjectClassName can be a string, hasAttribute can be Attribute (another ontology class). Attribute then classifies concrete attributes (its instances): CustomerID, FirstName, LastName, etc. (see Figure 4).

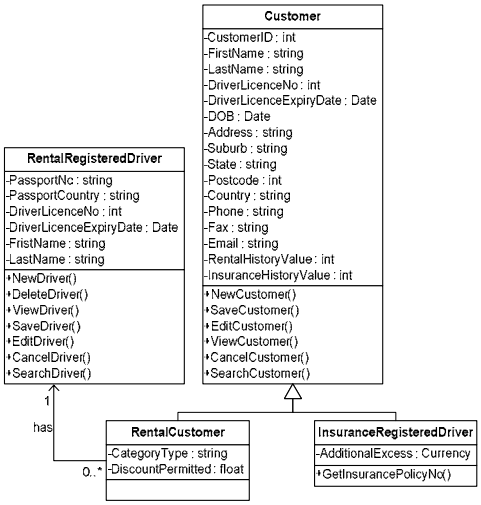

Figure 4: An example of class diagram put into the ontology as instance knowledge

Ontology class attributes are represented through properties. A property is a relation between a subject resource and an object resource. It might look similar to a concept of attribute in traditional in object-oriented sense. Nevertheless, they are different; ontology class property is stand-alone. It does not depend on any class like attributes in UML. In OWL, an ontology class property can be defined even if no classes are associated with it. OWL classifies two types of properties i.e. owl:ObjectProperty whose range can be only an instances and owl:DatatypeProperty whose range can be only a datatype value. OWL also defines additional characteristic on properties i.e functional, inverse functional, transitive, and symmetric property. These can further refine the property. Both ontology classes and ontology class attributes may be constructed in a

superclass-subclass hierarchy. Subclasses are subsumed by their superclasses and can encapsulate its superclasses.

An instance of an ontology class is also known as an individual. Class diagram for example shown in Figure 4 can be transformed into application-specific ontology as instance knowledge in the following way:

- ObjectClassName in ontology class of ObjectClass is 'Customer'.
- ObjectClass ontology class has relation named hasAttribute with Attribute ontology class. Attribute ontology class contains properties e.g. AttName is 'CustomerID', AttType is 'int', and AttVisibility is 'public' etc.
- ObjectClass ontology class has relation named hasOperation with Operation ontology class. Operation ontology class contains properties e.g. OperationName is 'NewCustomer', SetParameter and GetParameter is none and OperationVisibility is 'public' etc.
- ObjectClass ontology class e.g. 'Customer' has relation named hasRelationship with another ObjectClass ontology class. The hasRelationship is Generalisation ontology class which ParticipatingCls is 'RentalCustomer' (an ObjectClass ontology class).

Likewise 'use case' diagram in some project design can in the same way convert into application-specific ontology as instance knowledge using the ontology modelling of domain knowledge of 'use case' shown in Figure 3. The domain knowledge and instance knowledge will then be shared as serialised knowledge through the Internet. Project team members can then use it to create consistent understanding within.

5. Software Engineering Sub-Ontology Platforms

If we are to model the said ontology development approach as a system, this is how the man-machine system interfaces work. A user at any one site logs a project matter in the system. Figure 5 shows such an example of the text transcription which under circumstance of development teams are geographically distributed and team members are involved in many projects simultaneously made thing difficult.

The Question Platform stage is shown in Figure 6. During this interacting session, users need to specify what they mean of object class or attribute or operation or relationship. For example 'InsuranceRegisterdDriver' is object class so users choose it by highlighting the text of 'InsuranceRegisterdDriver' and then select type as a class to indicate that it is an object class. Also users can select other types i.e. relationship, attribute, operation to be shown. The result of its UML-like diagram is like shown in Figure 4.

The Suggestion platform and Solution platform interactions are similar, as both platforms will involve modifying instance knowledge. A user interacts with the Suggestion platform to propose instance chances, and pending on authorized approval made through the Solution platform, the proposed changes become solution changes. Until such status change, the instance changes get updated in the ontology repository. Figure 7 is a screenshot of a Suggestion/Solution platform interaction, when one can add, delete and modify instance knowledge. After the user has progressed the changes through both the platforms, its UML-like diagram will be shown as well. Those UML-like diagrams will help them to have a clear understanding as well as help them recognize the instance knowledge when they work in other and many more all at the same time.

I am struggling to understand why we need it. I think the system will be simpler for people to understand if we deleted the insurance registered driver.

My reasons for this are that the insurance registered driver is a sub type of the customer. This means that for every insurance registered driver object there must be a corresponding customer object. However, in the customer object we store values like customer type, insurance history value and rental history value. It does not make sense to have these values for the insurance registered driver. I also think people will be confused because we have the rental registered driver as an association with the rental customer (which is a sub type of the customer) but the insurance registered driver is a sub type of the customer.

Figure5: An example of plain text communication

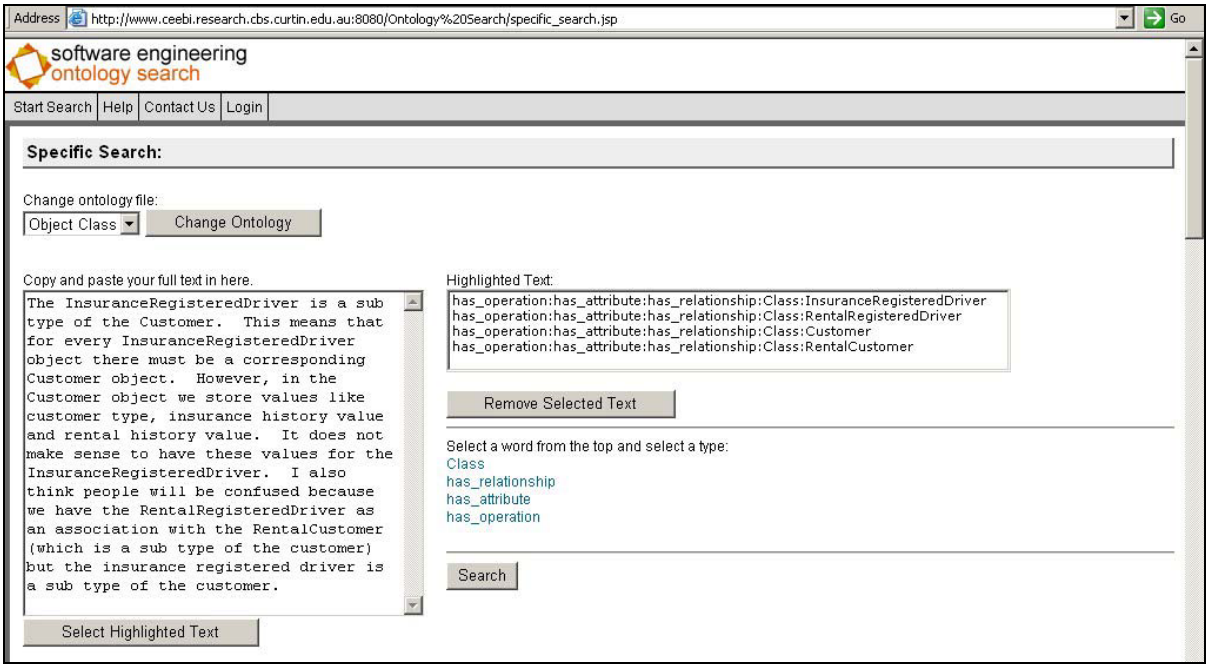

Figure6: Question platform

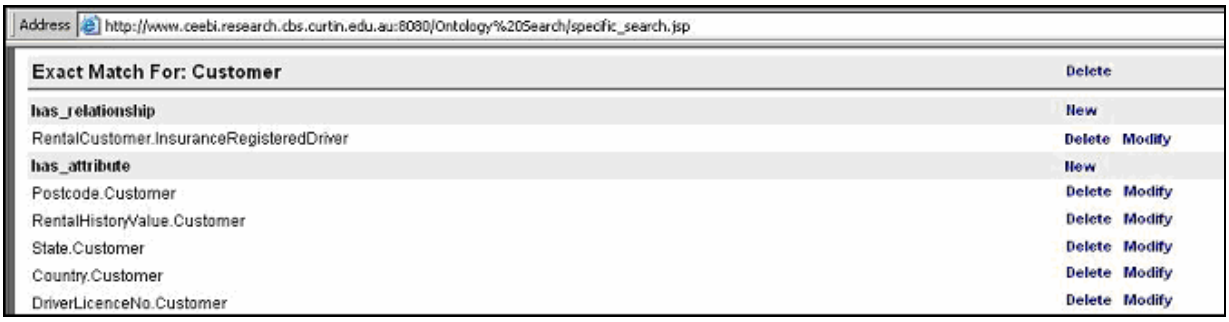

Figure7: Suggestion/Solution platform

6. Conclusions and Future Work

We have described how an application-specific ontology models an approach to transform explicit semantic knowledge (ie data instances in computers) to conceptual knowledge representations (by using UML notations) and formalise consensus agreement between project team players to approve instance knowledge as the common communication language and project knowledge across all sites. Our future work aims at implementing multiple software agents to access data from a project-ontology repository and performing reasoning etc. The research would also examine ways of how these agents mine and aggregate knowledge from other project-ontology repositories in networked systems to reconfigure a generic MDSD ontology, which can automatically grow and establish a larger scale of common communications and MDSD/domain knowledge without human being intervention. This concept can also extend to self a universally generic ontology that can self generate and maintain all the universal knowledge of the world and across different domain disciplines!

7. References

[1] Carmel, E. and R. Agarwal, *Tactical Approaches for Alleviating Distance in Global Software Development.* IEEE Software, 2001. **18**(2): p. 22-29.

[2] Van Fenema, P.C., *Coordination and Control of Polycontextual, Geographically Dispersed Temporary Systems: The Case of Global Software Projects*, in *Department of Decision and Information Sciences, School of Management: Rotterdam*. 2002, Erasmus University: The Netherlands.

[3] Kotlarsky, J.M., K. Kumar, and J.v. Hillegersberg. *Coordination and collaboration for globally distributed teams: the case of component-based/object-oriented software development*. in *Proceedings of International Workshop on Global*

Software Development (ICSE 2002). 2002. Orlando, Florida, USA.

[4] Wongthongtham, P., et al. *Ontology based solution proposal for multi-site distributed software development*. in *Proceedings of the 16th International Conference on Software and Systems Engineering and their Applications*. 2003. Paris, France.

[5] Wongthongtham, P., E. Chang, and T.S. Dillon. *Methodology for multi-site software engineering using ontology*. in *Proceedings of the International Conference on Software Engineering Research and Practice*. 2004. Las Vegas, USA.

[6] Wongthongtham, P., E. Chang, and T.S. Dillon. *Intelligent communication through software agent and ontology for multi-site software engineering*. in *Proceedings of the 3rd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*. 2004. Edinburgh, UK.

[7] Wongthongtham, P., E. Chang, and N. Jayaratna. *Ontology-based Software Engineering to Multi-site Software Development*. in *(accepted for presentation) We-b Doctoral Consortium November 25*. 2004. Fremantle, Australia.

[8] Wongthongtham, P., et al. *Software Engineering Ontologies and their Implementation*. in *(accepted for presentation) The IASTED International Conference on SOFTWARE ENGINEERING, February 15-17*. 2005. Innsbruck, Austria.

[9] Genesereth, M.R. and R.E. Fikes, *Knowledge Interchange Format Version 3 Reference Manual, Logic-92-1*. 1992, Stanford University Logic Group.

[10] Luke, S. and J. Heflin, *SHOE 1.01 Proposed specification*. 2000, SHOE Project.

[11] Librelotto, G., J.C. Ramalho, and P.R. Henriques, *XML Topic Map Builder: Specification and Generation. In: XATA: XML.* 2003, Aplicaes e Tecnologias Associadas.

[12] Karp, R., V. Chaudhri, and J. Thomere, *XOL:An XML-Based Ontology Exchange Language.* 1999.

[13] Kent, R., *Conceptual Knowledge Markup Language.* 1998.

[14] Horrocks, I., et al. *OIL in a Nutshell*. in *Proceeding of ECAI '00 Workshop on Application of Ontologies and PSMs*. 2000. Berlin, Germany.

[15] Horrocks, I. and F.v. Harmelen, *Reference Description of the DAML+OIL Ontology Markup Language*. 2001.

[16] McGuinness, D.L. and F.V. Harmelen, *OWL Web Ontology Language Overview*. 2004.

[17] Gennari, J., et al., *The Evolution of Protege: An Environment for Knowledge-Based Systems Development*. 2002, Stanford University, http://protege.stanford.edu.

[18] Bechhofer, S., et al. *OilEd: a Reason-able Ontology Editor for the Semantic Web*. in *Proceeding of KI2001, Joint German/Austrian conference on Artificial Intelligence*. 2001. Vienna: Springer-Verlag LNAI.

[19] Sure, Y., et al. *OntoEdit: Collaborative Ontology Development for the Semantic Web*. in *International Semantic Web Conference (ISWC02)*. 2002. Sardinia, Italy: LNCS 2343.

[20] Farquhar, A., R. Fikes, and J. Rice. *The Ontolingua Server: A Tool for Collaborative Ontology Construction*. in *10th Knowledge Acquisition for Knowledge-Based Systems Workshop*. 1996. Banff, Canada.

[21] Arpirez, J.C., et al. *WebODE: a scalable ontological engineering workbench*. in *First International Conference on Knowledge Capture (K-CAP 2001)*. 2001. Victoria, Canada.

[22] Seaborne, A., *Jena Tutorial: A Programmer's Introduction to RDQL*. Updated February 2004, http://jena.sourceforge.net/tutorial/RDQL/index.html.

[23] Carroll, J.J., et al., *Jena: Implementing the Semantic Web Recommendations*. 2004, Digital Media Systems Laboratory, HP Laboratories Bristol.