Reprinted from:

2005 3rd IEEE International Conference on Industrial Informatics (INDIN) Perth, Australia 10-12 August 2005

IEEE Catalog Number ISBN 05EX1057
ISBN 0-7803-9094-6

Information Engineering of a Software Engineering Ontology

Wongthongtham, P [1], Chang, E [1] and Dillon, T.S.[2]

[1] School of Information Systems, Curtin University of Technology, WA, Australia,
e-mail : (Pornpit.Wongthongtham, Elizabeth.Chang)@cbs.curtin.edu.au
[2] Faculty of Information Technology, University of Technology Sydney, Australia, e-mail: tharam@it.uts.edu.au@ieee.org

Abstract— **In this paper, we describe the preliminary result of the development and implementation of a Java-based system for information gathering, knowledge extraction and maintenance of software engineering ontology. The system is capable of manipulative ontology instances from the information repositories and information sources. Design of Software Engineering Ontology through the use of the body of software engineering knowledge together with Prof Ian Sommerville's book, as well as project management experiences, has not been a difficult task. However, the maintenance of the software development ontology and security of the Ontology are issues.**

Index Terms—**Software Engineering Ontology, Ontology Development.**

I. Introduction

Multi-site software development is where distributed software development teams reside across geographically sites [1-4]. Coordination and synchronization of the software development and willingness to answer ontological and technical as well as questions to remote software engineers should be tailored to multi-site distributed development teams.

In recent years, the notion of the 'ontology' has been gaining prominence in which according to Gruber's and Borst's definition been merged and explained by Studer and colleagues [5] ontology is a formal, explicit specification of a shared conceptualization. A conceptualization abstract way of understanding of some phenomenon of the world for which we agree to accept its consensual knowledge.

Ontologies play an important role in many disciplines e.g. in medical, pharmaceutical, law, etc. as well as they will do in software engineering field. In this paper software engineering ontology (1) provides a source of explicitly defined software engineering terms that can be used in communication between team members and organizations across geographical sites and applications e.g. intelligent agents; (2) offers a consensual shared agreement within teams; (3) supplies information retrieval concerning the instances pertaining to a certain domain of knowledge.

II. Issues in Software Development

In recent times, as more companies are going global, they are face with a multitude of problems when it comes to managing projects that are being developed cross countries. One of the major hurdles to overcome is the problem of miscommunication. Miscommunication occurs when developers have differing ideas of what a single term may mean or just have differing ideas of what other developers meant by their email.

One of the areas where miscommunications are predominant is software engineering. This is because there are a lot of software engineering textbooks around and most of them offer differing view on a particular terms or concepts that are used. This creates lots of confusion for developers when they are faced with those terms.

III. System Key Features

Our ultimate goal is to (*i*) build software engineering ontology through the use of the body of software engineering knowledge [8] together with Professor Ian Sommerville's software engineering book [9], (*ii*) implement Java-based system for information gathering, knowledge extraction and maintenance of software engineering ontology. We have developed a system that embodies this ontology.

Key features of the system include:

a) Fully modular design allows for easy implementation and maintenance. It also allows us to deploy it on different server easily and also allows us to use different programming languages to design the interface.
b) Users are able to search through ontology via a simple and easy to use web interface.
c) Modifications can be done to the ontology via a simple and easy to use web interface.
d) Full permissions system allow for administrator to control user access easily.
e) Modifications pending system allow for project leaders to easily identify any changes that are done to the ontology and to easily revert back or reject any changes that are made.
f) Allow project leaders to start new project and to upload new ontology file related to the project via the web interface.

IV. System Architecture

The architecture shown in Fig. 1 is design to be as modular as possible allowing us to easily modify and maintain the system without any big headaches. We have split the system into 2 parts i.e. the Ontology Query Server and the User Interface Server.

0-7803-9094-6/05/$20.00 ©2005 IEEE

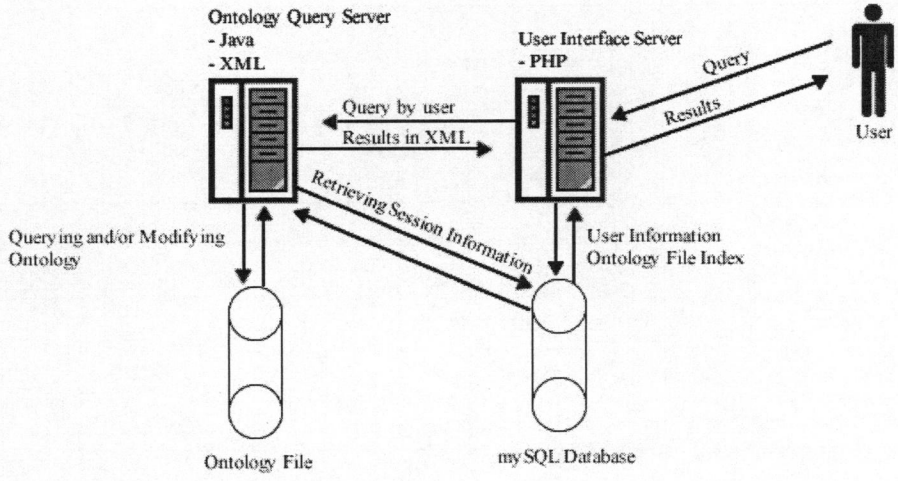

Fig.1 System architecture

The Ontology Query Server is responsible for the functions that are related to Ontology e.g. searching through the Ontology and modifying the Ontology. This is the only portion of the system that interacts with the Ontology. The Ontology Query Server is written in Java using the JENA classes and is deployed as a Java servlet allowing multitude of programs to access it via the web. The result of any queries is in XML format making it easy for programmers to understand and code for it.

In the current system, we are restricting the access to Ontology Query Server such that only users with certain privilege can access the modification or delete functions of the system but this can be easily change such that all of the available functions can be access by any person through any programs if necessarily.

The User Interface Server is basically a web server with scripts written in PHP to interact with the users. This is the interface that the users interact with and it interacts with database to perform account administration and other important functions that are needed for the maintenance of the system.

V. Software Engineering Ontology

In this section we briefly describe the conceptualized software engineering ontology.

Concepts in an ontology are normally established in taxonomies which capture inheritance and aggregation. For instance, in the software engineering domain, concepts are: software process, requirement, design, development, and verification and validation. Taxonomy of design entity can be represented by where object oriented design, real-time software design, user interface design, and so on are all subclass of design. The object-oriented design model is an association of object-oriented design. Class diagram, use case, etc are components classes of object-oriented design model.

Relationships represent a type of relationships between concepts of the domain. For example *has model* links *software engineering* to *model*. Each relationship may have an inverse relationship that links the concepts in the opposite direction and/or a symmetric relationship that links the concepts in the same direction and/or transitive relation that links the concepts in the transitive direction. For example the relationship *is tool of* is the inverse of *has tool*,

Instances describe objects or individuals in an ontology. An example of instance of the concept *class* in UML is *customer* or *driver*.

Attributes depict properties of instances and of concepts. There are two types of attributes which are instance and class attributes. Instance attributes represent properties of instances of concepts and take their values in the concept instances. For example *class name* is an instance attribute of the concept *class* in UML. Class attributes represent instances and take their values in the concept from within which they are defined. An example is the attribute *operation visibility* of the concept *operation* in UML class diagram that can be used to determine the visibility of operation of a *public*, *private*, and *protected* operation.

Formal axioms are typically used to specify constraints in an ontology. They are logical expressions which are always true. For instance a *relationship* in a UML class diagram cannot be *dependency* and *association* in the same diagram.

Rules are usually used to infer some knowledge in the ontology. For example object-oriented design is utilised in project#1 design, which is implemented using Java language.

Reasoning Mechanisms

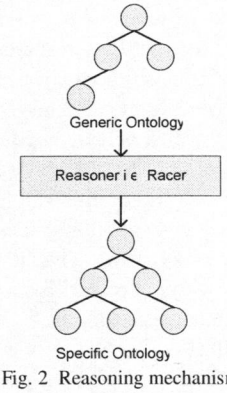

Fig. 2 Reasoning mechanism

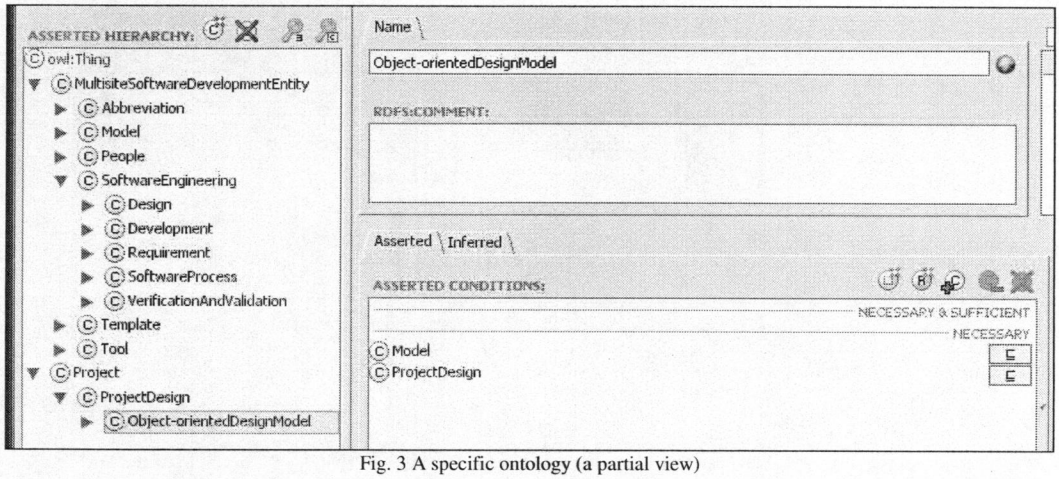

Fig. 3 A specific ontology (a partial view)

There are a few inference engines available for reasoning e.g. FaCT, RACER, TRIPLE, etc. but for reasoning with OWL there are not many inference engines available. We have adopted RACER for reasoning in our software engineering ontology. Reasoner will permit performing creation of the automatic classified specific ontology from a generic ontology based on agreement in that particular project as shown in Fig. 2. Fig. 3 shows a specific ontology obtained from reasoning with a generic ontology for that particular project which utilizes object-oriented design for the project design. As the OWL ontology allows for multiple inheritances, object-oriented design can be subclass of model and project design in the specific ontology shown in Fig. 3. Monotonic reasoning is assumed in OWL. Having said that, facts obtained by inference engines can only be extended never reduced and that new information cannot controvert previous information. In a specific ontology sense, more recent information concerning a particular project will be added on the generic ontology.

To be more precise with multi-site software development environments, every remote team infers a shared generic ontology. Output of reasoning with a shared generic ontology is a specific ontology which will be used by particular project team members to help clarify information within their teams etc.

VI. Software Engineering Ontology Maintenance

A scenario where using our system is applicative is when users are requesting for modifications to the ontology instances. Project leaders need to approve each and every modification done to the ontology instances, even those that are considered minute. This places a lot of work on the project leaders as they need to identify and go through each of the modifications before the changes can be committed or rejected. By using the system, the amount of workload on the project leaders can be dramatically reduced. This is done by using system as a safeguard. It is especially useful as we can leave minor changes (changes that don't affect the project in a huge way) approval by system and also use system to suggest changes that might be helpful to the project. The system can also identify whether the changes can be made and immediately notify the users if a particular change is considered to be violating certain aspects of the project and thus reduces waiting time.

By using the system, we can significantly reduce the amount of workload on the project leaders but we will still need human intervention for certain changes that are too big and might have serious repercussions on the projects' development if the changes are made.

Ontology Self-maintenance

The ontology itself will evolve over time as more and more projects are added into it and corrections are made. With this in mind, the maintenance of the ontology presents a challenge to the project leader and team leader.

With a modifications pending system in place, we can make sure that no changes are made without approval from the project leader or team leader. By using the pending system, we reduce the chances of committing changes that might be harmful to the project development. Although this is not a very efficient system, it helps to reduce the amount of errors that might have otherwise occurred if users have free reign to modify the ontology instances.

VII. The Prototype

Jena [10] provides utilities to allow communication with the Ontology. The Jena program is the application which retrieves relevant information for users with the use of a web GUI, parsing the input data to the Jena program. The project provides users with 2 kinds of search, a generic search which displays semantics of software engineering concepts and a specific search for searching and doing maintenance functions like add, modify and delete for specific ontology.

10.1 Generic Ontology Implementation (General Search Program)

Generic Search is mainly used for displaying search results only. The main function is to allow users to access retrieve and view the knowledge base and acquire the information that they are seeking. First the program gets the user input from the web interface and then finds the class in the Ontology. If it is not found, an error message will be given back to the user, which can be customizable using the JSP. If it is found, the program will display all the properties of

that particular class. Another feature of this program is the ability for it to display the parent class and all the subclasses of that particular class in a tree format. All the results are stored in a vector which can be easily accessible for JSP to manipulate the formatting in web pages. Fig. 4 shows generic search of computer-aid software engineering concept.

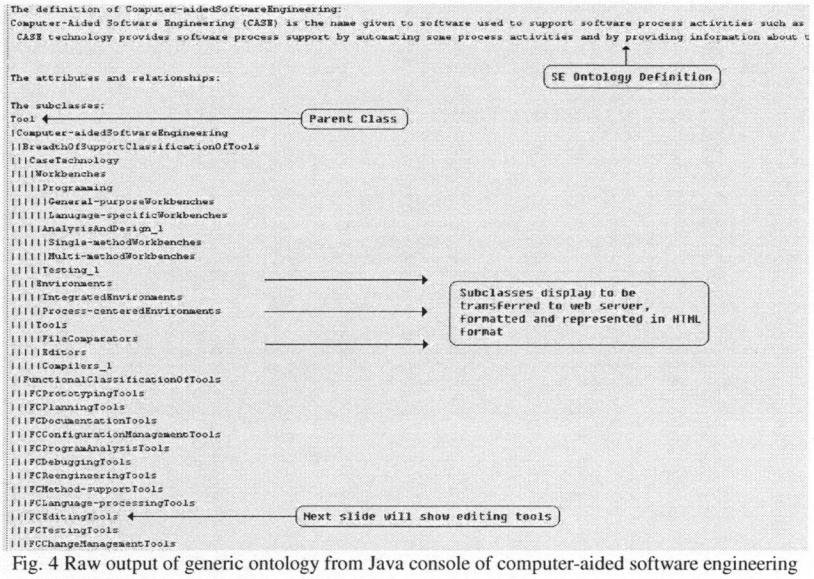

Fig. 4 Raw output of generic ontology from Java console of computer-aided software engineering

In the following example shown in Fig. 5 *FCEditing-Tools* does not have any definitions defined therefore the field is empty. The purpose of this example is to show the program's ability to display the instances (examples) of the particular searched class.

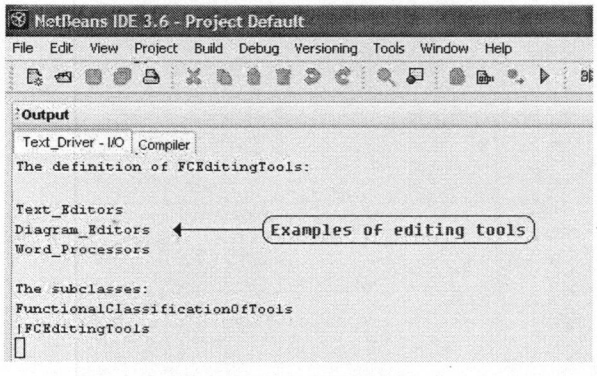

Fig. 5 Instance ontology output from Java console.

10.2 Specific Ontology Implementation (Add, modify, Delete)

Specific Ontology program allows software engineers involved in a certain project to access that project with the ability to search within the project. With authorized access, the engineers will be able to add, modify or delete instances/individuals and property values of them. All the standards like unique individual names, class names are taken into account and the program will detect such errors and prompt the users with relevant error messages.

Fig. 6 shows output example A, B, C, and D. Example A shows a failed attempt at updating the *student* instance because the value of *object class name* is invalid. The program recognizes a different value and terminates the process and then gives the user an error message and advises the user of correct methods of adding. Example B is the same as A except in this case the value of *object class name* is validated and therefore the updating process can be completed successfully. Example C shows an invalid input with new value, which is the *student age*. It is not a valid attribute instance in the ontology, therefore the validation fails and the program gives error messages. Example D contains valid new value, *DOB.Customer* which is an attribute instance in the ontology. Therefore the updating process is completed successfully.

Fig. 7 shows examples of add, modify and delete of ontology instances and properties. With a given scenario to each example, it helps to give meanings to why the instance or property is needed to be updated or deleted. Example A shows the *insurance registered driver* is not needed in the ontology and therefore a delete instance operation is carried out. Example B proposes that a new operation is needed to record customers who have been with them for more than 5 years so in the near future they can carry out loyalty rewards for these customers. In order to do these, a new operation instance has to be added called *ViewVIP.Customer* which displays a list of loyalty customers who have been with them for more than 5years. After creating a new instance, there is no property added yet, therefore new properties need to be added. Example C shows the addition of the operation name. After reviewing, the manager decides to remove "ViewCustomer.Customer" operation property from Customer instance because there is another operation which serves the same purpose as shown in Example D. These are a few examples of the operations that are available.

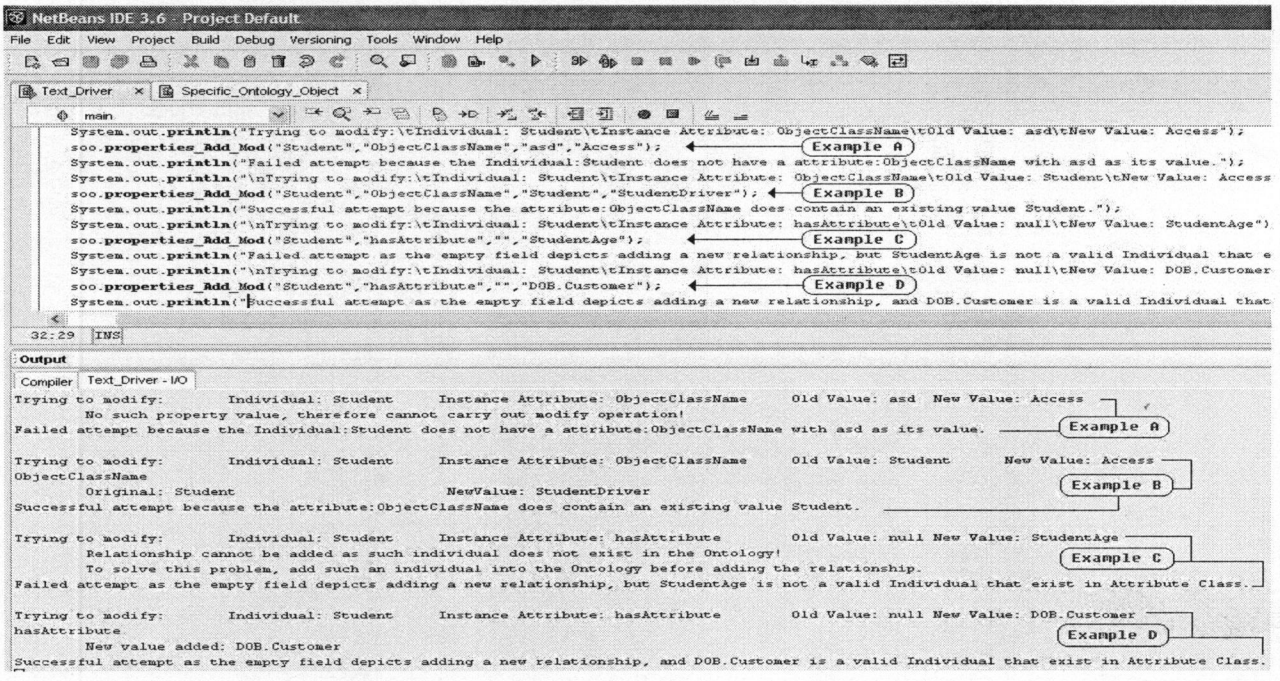

Fig. 6 Backend ontology maintenance, validations of new property of software engineering ontology instance.

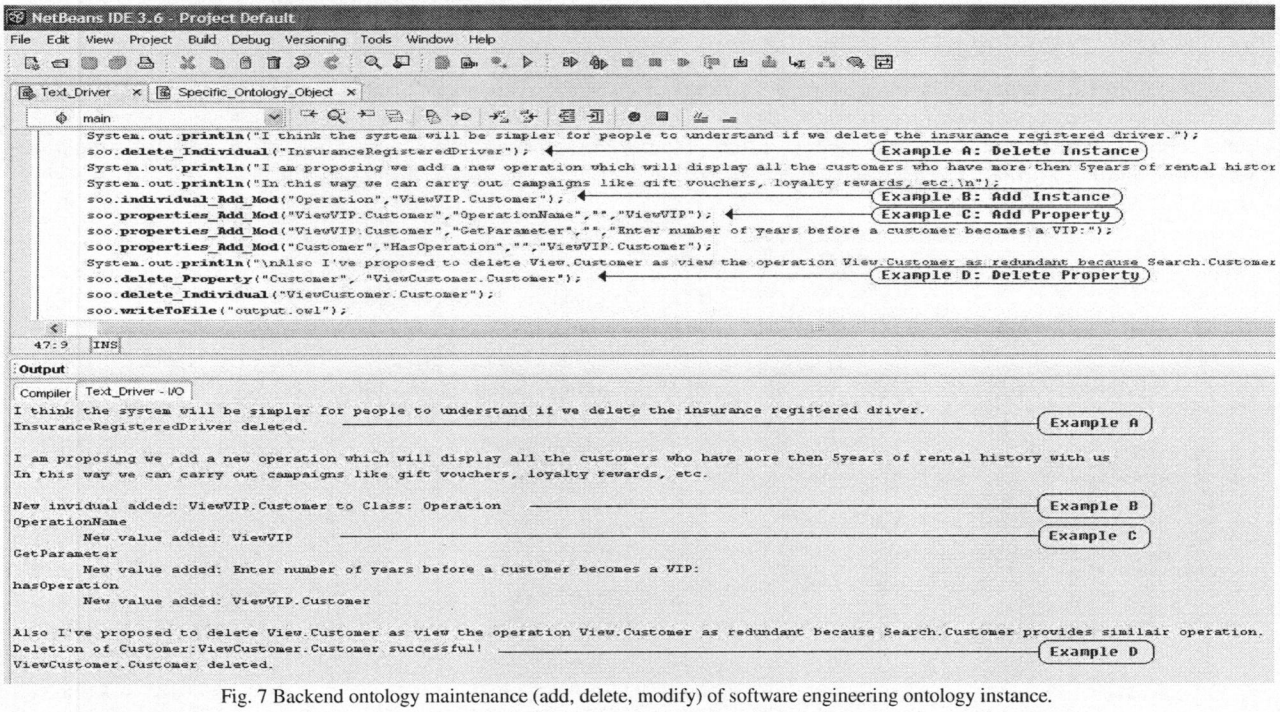

Fig. 7 Backend ontology maintenance (add, delete, modify) of software engineering ontology instance.

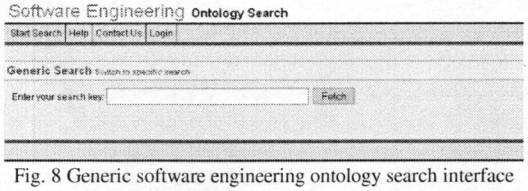

Fig. 8 Generic software engineering ontology search interface

Fig. 8 shows the generic search screen that user will see when they first visit the website. A user can key in the search terms they want to search for in the search field which will then return a list of results with definitions for the user.

Fig. 9 shows the specific software engineering search interface which allows the user to search through particular project ontology and to have more control over their search in respect of the particular class or type that they want to search for. Additionally, it allows the user to add a new ontology instance and also a property. Users are also able to modify and delete any properties that they think are not useful for the project. All of the actions are subject to the rule

that the user has the permission to modify or delete the ontology in the particular project. With the use of web interface, we hope that we can reduce the learning curve and also be able to implement this system on a wider scale.

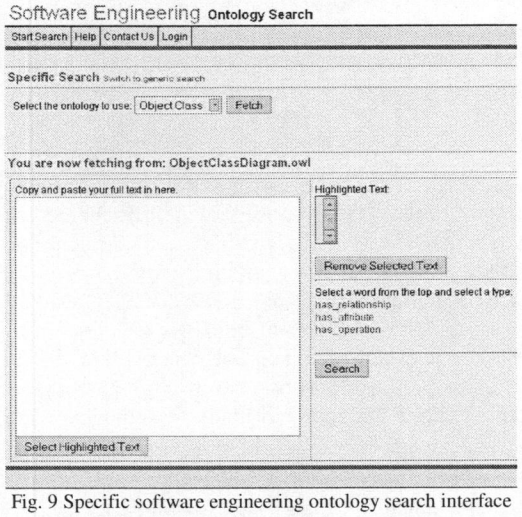

Fig. 9 Specific software engineering ontology search interface

VIII. Conclusion

This project was developed to reduce the miscommunications problem that might occur and to also allow a way for project leaders to have a better understanding of the project and to allow project members to be able to communicate much more efficiently with one another even if they are not situated in the same location. This is done through using an ontology and a web interface with which, the project members can query, retrieve and modify information from the ontology instances.

IX. References

[1] Chang, E., 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.

[2] 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.

[3] 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.

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

[5] Studer, R., B. VR, and D. Fensel. *Knowledge Engineering: Principles and Methods.* in *IEEE Transactions on Data and Knowledge Engineering*. 1998.

[6] Costello, R.L. and D.B. Jacobs, *'OWL Web Ontology Language'*, 2003, The MITRE Corporation.

[7] Lacourba, V., *'Archive of W3C News in 2004'*, 2004.

[8] Arban, A., Moore, J., Bourque, P., Dupuis, R.L., Tripp, L. *Guild to the Software Engineering Body of Knowledge - SWEBOK.* in IEEE-Computer Society Press, May 2001, URL:http://www.swebok.org.

[9] Sommerville, I. *'Software Engineering'*, 2004: Addison Wesley.

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