# Ontology Instantiations for Software Engineering Knowledge Management

Pornpit Wongthongtham, Elizabeth Chang
School of Information Systems
Curtin University of Technology
Perth, WA, Australia
Email: {Pornpit.Wongthongtham, Elizabeth.Chang}@cbs.curtin.edu.au

***Abstract*—In this paper, we explore the development of systems for software engineering knowledge management. Software engineering knowledge is represented in the software engineering ontology whose instantiations, which are undergoing evolution, need a good management system. Software engineering ontology instantiations signify project information which is shared and has evolved to reflect project development, changes in the software requirements or in the design process, to incorporate additional functionality to systems or to allow incremental improvement, etc. In a large volume of updated project information, systematic management is of importance.**

## I. INTRODUCTION

Software engineering ontology instantiations are derived as a result of populating software engineering project information and are referred to as ontology instances of ontology classes. Instantiations are also known as instance knowledge of the software engineering ontology. In other words, once the software engineering ontology is designed and implemented (see papers [1, 2]), it needs to be populated with data relating to the project. This process is usually accomplished by mapping various project data and project agreement to the concepts defined in the software engineering ontology. Once mappings have been created, project information, including project data, project agreement, and project understanding, is in a semantically rich form and management is needed to maintain the instantiations.

The software engineering ontology contains abstractions of the software engineering domain concepts and instantiations. There are two types of the abstraction which are the generic software engineering and the specific software engineering. The abstraction of the generic one represents the concepts that are common to a whole set of software engineering concepts, while the abstraction of the specific one represents the set of software engineering concepts that are specifically used for some categories of particular projects. The instantiations, also known as population, are simply the project data. The abstraction of the specific software engineering ontology has its instantiations ultilised for storing data instances of the projects. Each abstraction can have multiple instantiations in different circumstances of projects. The corresponding concrete data instances are stored as instantiations. In this study, the software engineering ontology integrates abstractions and instantiations together, rather than separating them by storing instances in a traditional relational database style linked to the knowledge base. The latter, SQL queries, can help with the large volume of concept and instance management and maintenance. Nevertheless, in the software engineering ontology, the data volume is not very large and coherent integration of abstraction and instantiations are important in the software engineering projects. Putting them together instead of separately would be more suitable for this study. For example, each project contains a different narrow domain (specific software engineering ontology) and limited numbers of data instances. The domain specific ontologies are locally defined; that is, they are derived from the generic software engineering ontology so they are not created with respect to some global declarations. Obviously, this example scenario strongly indicates that abstractions and instantiations are better stored together instead of separately because the latter one asks for a unified global declaration of abstractions. In conclusion, ontology instantiations for software engineering knowledge management actually means management of the instantiations.

In reality, in software engineering projects, the project data over a period of time needs to be modified to reflect project development, changes in the software requirements or in the design process, in order to incorporate additional functionality to systems or to allow incremental improvement. Since changes are inevitable during software engineering project development, the instantiations of the software engineering ontology is continuously confronted with the evolution problem. If such changes are not properly traced or maintained, this would impede the use of the software engineering ontology. Due to the complexity of the changes to be made, at least a semi-automatic process becomes increasingly necessary to facilitate updating tasks and to ensure reliability. Note that this is not ontology evolution because it does not change the original concepts and relations in the ontology, rather instantiations of the ontology change or that conform to the ontology change. Fig. 1 shows the abstraction is fixed but only the instantiations are always changing.

1-4244-0755-9/07/$20.00 ©2007 IEEE

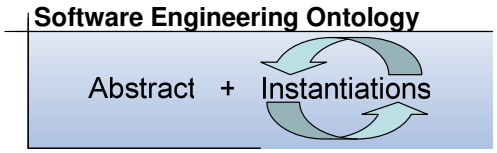


Fig. 1. Instantiations of the software engineering ontology is the only component continuously confronted to evolution problem.

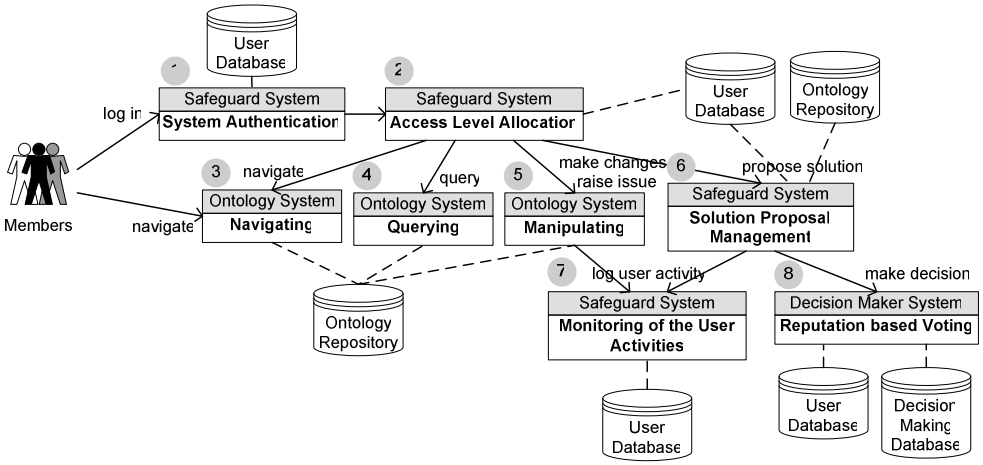


Fig. 2. Functionalities provided in each system

Thus, software engineering domain changes that are produced by new concepts, and change in the conceptualisation as the semantics of existing terms are modified with time, are all outside the scope of this study.

In this paper we present development of applications in systems to facilitate software engineering ontology instantiations management. The software engineering ontology is made available to any application to deploy. The ability to make use of the software engineering knowledge, described in the software engineering ontology, enables applications in the systems to have capabilities in managing instance knowledge in multi-site distributed software development.

In the next section, we give an overview of systems development for software engineering ontology instantiations management. Then in section 3, we describe software engineering ontology management. We present software engineering ontology system model packages in section 4. In section 5, we finally conclude paper and suggest for future work.

## II. Development of Systems for Software Engineering Ontology Instantiations Management

Management tasks for software engineering ontology are assigned to the systems. In detail, the functionalities of each system can be observed in Fig. 2. The safeguard system functionalities include system authentication, access level allocation, solution proposal management and monitoring of the user activities. Functionalities in the ontology system include navigating, querying and manipulating software engineering ontology. For the decision maker system, a method of reputation based voting is used in the system.

Every team member can navigate the software engineering ontology but no changes allow (number 3). A team member can log into systems. Once logged in, system authentication (number 1) in the safeguard system verifies user access from user database. Once authorised, the member will be provided the access privileges from the safeguard system (number 2). The member can now navigate, query, make changes, raise issue, or propose solution. Navigation, query, and manipulation of instantiations in software engineering ontology are functioned by the ontology system (number 3-5). Manipulation of instantiations is functioned by the ontology system and is recorded into user database by the safeguard system (number 7). Solution proposal is managed by safeguard system (number 6). Any decision is made by decision maker system (number 8).

## III. Software Engineering Ontology Management

The purpose of having an ontology system is to manage connections with the software engineering ontology. The ontology system is built on top of Jena [3] which we would like to gratefully acknowledge. Jena developed by the Hewlett-Packard Company is a Java framework having capacity of manipulating ontologies [4]. The version of Jena used is Jena 2.1. The ontology system provides navigating, querying, and manipulating software engineering ontology. The design philosophy of the ontology system is to use the in-memory storage model and serialise it into a physical document stored in the ontology repository. It is an attempt to minimise the query response time. Note that this is not like a knowledge base system that uses the data based model to query the ontology and instance data.

Software engineering concept structures are formulated so that it can easily be navigated. A team member can navigate in the software engineering ontology for clarification or classification certain concepts. The information provided is in hierarchical form so upper level concepts or lower level concepts or adjacent concepts can easily be navigated.

Technically for this function, the ontology system focuses on the software engineering ontology model, the set of statements that comprises the abstraction and instantiations. To navigate the software engineering ontology, the ontology system reads OWL software engineering ontology into a model and then accesses the individual elements.

As stated earlier, the in-memory storage model is used hence a query primitive supports. The query facilities of RDQL [5] which is a data based model held in a persistent store, is not within the scope of this study.

It serves as a searching tool to help narrow down the vast number of concepts in the ontology. Through the use of the ontology search function, the team member can re-classify concepts to match their project needs. This leads to the specific ontology. Note that the information provided by this function is all in XML format, which means that it can be easily managed to display only a certain part of the information retrieved or be able to provide a different display interface with the same set of information retrieved.

In the specific software engineering ontology which contains project data, a team member can add, delete, and update the project data. However, the ontology system will only allow direct updates for the minor changes/updates. The changes will be recorded and team members will be advised of the changes. An example of minor changes is enumerated types where the changes allowed are already fixed and team members cannot put in other values. Another example of minor changes is a changing of status of a document with the option of, for example, 'verified' or 'processing'. By default, any updating apart from the minor changes will be done by the decision maker system and be recorded. Even the ontology system considers whether they are minor changes or major changes, though there is an option for a team member to select whether or not these changes will go through the decision maker system. In the decision maker system, the changes will not be updated immediately to the specific ontology. They need to be voted by members of the community and therefore need to be stored in the decision making database. The process of decision making is handled by the decision maker system whose details are given in the next section. The ontology system simply checks whether the update request had been authorised before being updated. Basically, for major changes, the ontology system will pass the request of changes to the decision maker system to proceed further with processes of, for example, gathering information, consulting the ontologies in ontology repository etc. Once it has passed through the decision maker system, the updating can be done by the ontology system. Every activity will be recorded and the results of the processes are sent to the user that made the enquiry and to every team member involved.

## IV. Software Engineering Ontology System Model Packages

The architecture of the ontology system consists of three packages: 'generic', 'specific' and 'ontology'. The 'generic' package defines the interfaces of the data structures of generic software engineering ontology and generic software engineering ontology objects. Likewise, the 'specific' package defines the data structures of specific software engineering ontology interfaces and specific software engineering ontology objects, such as class and its instances. Both 'generic' and 'specific' packages provide an in-memory implementation of the data models of generic and specific ontologies respectively. The 'ontology' package provides the utilities for the ontologies defined in 'generic' and 'specific' packages.

### *A. Generic Package*

Generic software engineering ontology can be accessed by anyone without system authentication. It is used for a search of concepts relating to the software engineering domain. Unlike specific software engineering ontology, it is meant to be used for the projects, and therefore system authentication is required. Generic search allows searching of any concept within the software engineering ontology. Search results display the contents of the concept the user specifies including its subclasses, its properties and restrictions. The output is in XML format in order for it to be displayed easily on the web browser. Basically, the display of the hierarchy of subclasses can be accomplished using a recursive function. The function will find out the entire sub concepts of a concept by recursively calling the function itself over and over again until no more sub concepts can be found.

### *B. Specific Package*

A specific package provides a set of functionalities that helps the project team to have a mutual understanding through the use of specific software engineering ontology. Not only can the project members update project data, but also by withdrawing partial relevant knowledge from the software engineering ontology, issues can be discussed and solutions proposed. The set of functionalities includes: view, query, add, delete and modify instances or simply project data and properties.

To retrieve instances of a concept or ontology class, a function retrieves all direct instances related to the class or the concept. From here, users can browse this instance information. All information associated with the instance is like its definition, its properties (both object properties and data type properties) or its relationships, value inside those properties and its restriction. The main purpose of retrieving its relationships are firstly to help the team members understand its underlying concept; secondly, to help discussion on the issues or solutions; and lastly, to help update project

data to be completed according to its domain concept. It is easy to become confused if discussion takes place with words only, especially when there are many ambiguous words in the software engineering domain. Therefore, by retrieving the relationships associated with instances, it can help team members illustrate what they truly mean. This is done even better with the Java drawing toolkit which can be used to draw a relationship diagram.

Manipulating specific software engineering ontology is an essential tool to help maintain a project because all project data is stored as instance. In reality, project data are always updated from time to time. When project data need to be updated or added, a function even helps to check essential parameters needed in order to retrieve from its associated relationships, its restrictions etc. Updating is divided into two types of update: minor and major. Any significant changes such as those that annihilate certain information or add an entirely new instance to a project, are considered as a major update. Additionally, all object properties are considered as a major update because they reflect the changes of relationships. Requirements that satisfy the condition of being a minor update are firstly, any changes made by members in their field of expertise or simply in their team. For example, a designer making changes in the domain of project design will have the right to do so, therefore they are considered as minor updates. However, the designer making changes to the domain of project implementation will then not have valid rights and the changes will be considered as major updates instead. All data type properties are also classified as minor updates.

### C. *Ontology Package*

The 'ontology' package is a compilation of functions that provides the utilities for the ontologies defined in 'generic' and 'specific' packages. It does not belong to any category and does not have enough information to create its own category either. The functions in the package are mainly like *(i)* getting ontology name space, *(ii)* search engine for the ontology system, *(iii)* ontology class or concept restrictions and property characteristics checker to specify the range or restrict the values of input, *(iv)* converting information into XML format for output and *(v)* parsing and serialising ontologies in OWL language.

Firstly, getting ontology namespace is needed to extract the namespace of the ontology. The OWL file stores all the information of the different URL and namespace in the header of the file. When OWL file is loaded into the ontology model by calling the Jena modelfactory [6], all the URLs and namespace for the ontology can be retrieved. Since the ontology model loads all URLs first then loads the namespace of the ontology, the namespace is always the last element. By using Java's StringTokeniser [6] the namespace of the ontology can be retrieved with ease.

Secondly, a function in the package serves as a search engine for the ontology system. It especially includes finding any close match of ontology class or concept. This is useful when the search does not return any exact match to the user.

Ontology restrictions include quantifier restrictions and cardinality restrictions and ontology property characteristics include functional, inverse functional, symmetric and transitive properties. Functions of checking all ontology restrictions and property characteristics are all in the package to restrict the conditions. A function checks whether there is a minimum cardinality restriction implemented on the concept or ontology class. Minimum cardinality restriction refers to the minimum number of properties that must be input in order to satisfy the condition. For maximum cardinality restriction, a function checks for a maximum number before a new property value is added. If the maximum cardinality number is reached, adding of a new property value is disallowed, or else adding of a new property value is allowed. If there is a cardinality restriction present, it means there can be no more and no less cardinality than the cardinality specified. The quantifier restrictions of allValueFrom means that all the values of this property to whom this restriction applies, must have all values falling within its range. Likewise, someValueFrom restrictions, some values of the property whom this restriction applies to must have some values falling within its range. Therefore, the aim of a function for this is to check whether the new value which is going to be added falls into the category (if so return true; if it does not fall into the category, return false). These kinds of restrictions are only for adding new object properties.

Fourthly, a function is to convert information retrieved into XML format for output. This function is used often to display instance information including its restrictions information retrieved. All restrictions follow the same output format with XML tag as the name of the restriction and its value.

Lastly, parsing and serialising ontologies in OWL language are needed for format translation. A format or syntax translator requires the ability to parse, represent the results of the parsing into an in-memory ontology model, and then serialise. Manipulation capabilities for example would also be required, in between the parsing and serialising processes, in order to allow construction and editing of ontologies. In the implementation view, parsing is taking the OWL file and converting it to an in-memory ontology model. Conversely, analogous to the parsing, serialising produces an OWL concrete syntax in the form of a syntactic OWL file from an in-memory ontology model.

## V. CONCLUSION

Ultimately, the systems facilitate collaboration of teams in multi-site distributed software development. We have explored the development of systems for management of software engineering knowledge formed in the software engineering ontology. We have analysed instantiations in the software engineering ontology. Instantiations signify project information which is shared and evolved to reflect project development, changes in the software requirements or in the design process, to incorporate additional functionality to systems or to allow incremental improvement, etc. Accordingly, management

systems have been introduced and discussed. For future work, we shall focus on platform development as per discussed in this paper.

## REFERENCES


[1] 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.

[2] Wongthongtham, P., E. Chang, and T.S. Dillon. *Software Engineering Sub-ontology for Specific Software Development*. in *29th Annual IEEE/NASA Software Engineering*. 2005. Washington DC, USA.

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

[4] McBride, B. *Jena: Implementing the RDF Model and Syntax Specification*. in *Semantic Web Workshop, WWW2001*. 2001.

[5] Seaborne, A. *Jena Tutorial: A Programmer's Introduction to RDQL*. Updated February 2004.

[6] McCarthy, P., *Introduction to Jena: use RDF models in your Java applications with the Jena Semantic Web Framework*. 2004, SmartStream Technologies, IBM developerWorks.