Software Engineering Ontology – the Instance Knowledge (Part I)

Wongthongtham P[†], Chang E[†], Dillon T[†] and Sommerville I[††],

Digital Ecosystems and Business Intelligence Institute, Curtin University, Australia

School of Computer Science, St Andrews University, UK

Summary
The software engineering ontology defines common sharable software engineering knowledge including particular project information. Reaching a consensus of understanding is of benefit in a distributed multi-site software development environment. Software engineering knowledge is represented in the software engineering ontology whose instantiations are undergoing evolution. Software engineering ontology instantiations signify project information which is shared and has evolved to reflect project development, changes in software requirements or in the design process, to incorporate additional functionality to systems or to allow incremental improvement, etc. This evolution of instances provides many new challenges to an ability to design and deliver project information. In this paper, we present development of application tools to facilitate software engineering ontology instantiations management.

Key words:
Ontology, Software Engineering, Multi-site Software Development, Knowledge Engineering.

1. Introduction

Recently, there has been an explosion of interest in ontologies as artefacts to represent human knowledge and as a critical component in several applications. One unique area of research is the software engineering ontology [1]. The world's first Software Engineering Ontology is available online at www.seontology.org. The software engineering ontology defines common sharable software engineering knowledge including particular project information [2]. Software engineering ontology typically provides software engineering concepts – what they are, how they are related, and can be related to one another – for representing and communicating over software engineering knowledge and project information through the internet [1]. These concepts facilitate common understanding of software engineering project information to all the distributed members of a development team in a multi-site development environment. This should not be confused with the distributed systems, such as CORBA where the development is centralised but deployment is distributed. The ontology enables effective ways of sharing and reusing the knowledge and the project information for remote software engineers and software developers. Reaching a consensus of understanding is of benefit in a distributed multi-site software development environment. Software engineering knowledge is represented in the software engineering ontology whose instantiations are undergoing evolution. Software engineering ontology instantiations signify project information which is shared and has evolved to reflect project development, changes in software requirements or in the design process, to incorporate additional functionality to systems or to allow incremental improvement, etc. This evolution of instances provides many new challenges to an ability to design and deliver project information.

In this paper, we present development of application tools 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.

2. Software Engineering Ontology Instantiations

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 created, 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

2.1 Instantiations Analysis

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 utilised 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 summary, 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. Figure 1 shows the abstraction is fixed but only the instantiations are always changing.

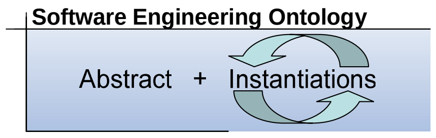

Fig. 1 Instantiations of the software engineering ontology is the only component continuously confronted to evolution problem

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.

When there are changes made to the instantiations of the ontology, they are all recorded by a logger object. Basically, instantiations can be updated by three basic operations: add, delete and modify. The add operation extends the existing instantiations of the ontology with new instantiations. The delete operation removes some instantiations from the ontology. The modify operation modifies some instantiations of the ontology but it still keeps its original construct. Generally, any update to the instantiations of ontology can be described by a sequence of the three operations. For example, a delete operation followed by an add operation can be considered as a replacement operation. Notice that the replacement operation loses its original construct while the modify operation still maintains its construct.

2.2 Instantiations Transformation

In this section, we particularly report on how software engineering project data are transformed or mapped into concepts formed in the software engineering ontology as instance knowledge. Conversely, the instance knowledge can be transformed back to more presentable and semantic project data e.g. diagram-like project data. Once transformed, instance knowledge is available for sharing among multi-site teams. Manipulation of semantics such as instance knowledge can be carried out by users or remote members.

An example of transformation is given here. Figure 3 shows an example UML class diagram that will be transformed into the class diagram ontology model as instance knowledge.

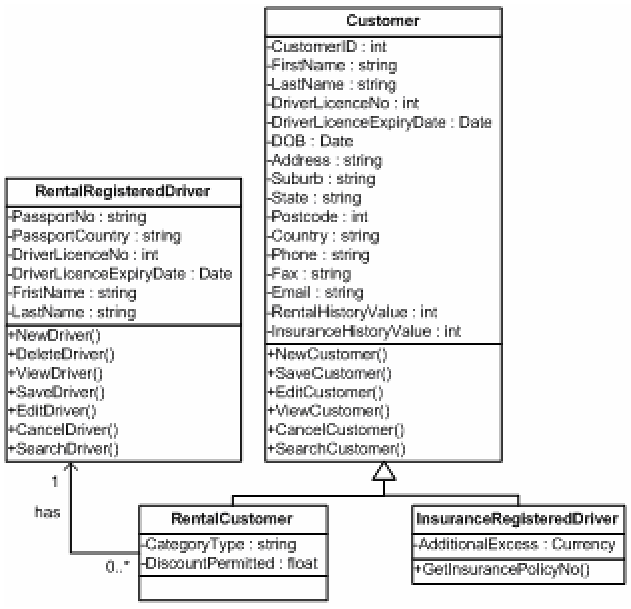

Fig. 3 An example of an UML class diagram

As from figure 3, UML classes Customer, RentalCustomer, InsuranceRegisteredDriver, and RentalRegisterredDriver apply as instances of the ontology concept Class in class diagrams ontology. Explicit domain knowledge from concept Class elicit that class consists of its properties, its operations and its relationships. This is by referring respectively, in the class diagrams ontology, to relations Class_Attribute, Class_Operation, and association ontology class ClassRelationship. The concept Class instance Customer has relation has_Attribute with concept ClassAttribute instances CustomerID, FirstName, LastName, DriverLicenceNo, etc. For example, the concept instance DriverLicenceNo has relations Class_Attribute_Datatype with xsd:string of ‘Integer’ and has relations Class_Attribute_Visibility with xsd:string of ‘Private’. These are shown in figure 4.

For a particular UML class Customer, operation NewCustomer() applies as an instance of concept ClassOperation in the class diagrams ontology model. The concept Class instance Customer has relation Class_Operation with concept ClassOperation instance NewCustomer. The concept instance NewCustomer has relations Class_Operation_Visibility with xsd:string of ‘Public.’ These are shown in figure 5.

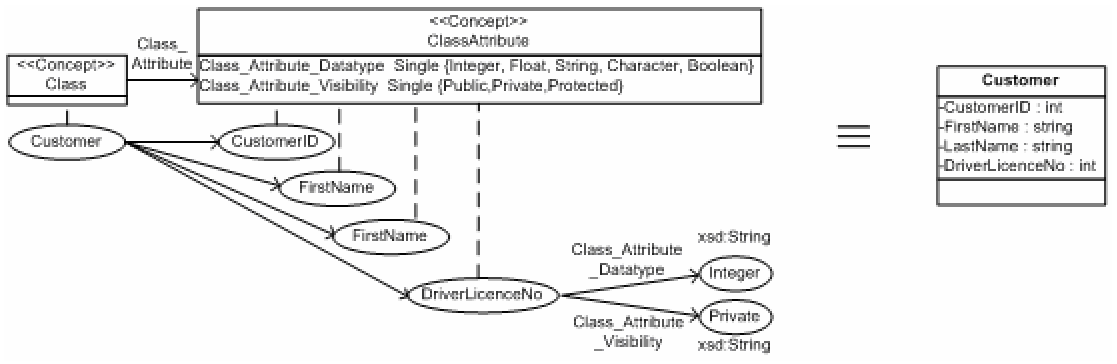

Fig. 4 Transformation of UML class Customer and its attributes to class diagrams ontology

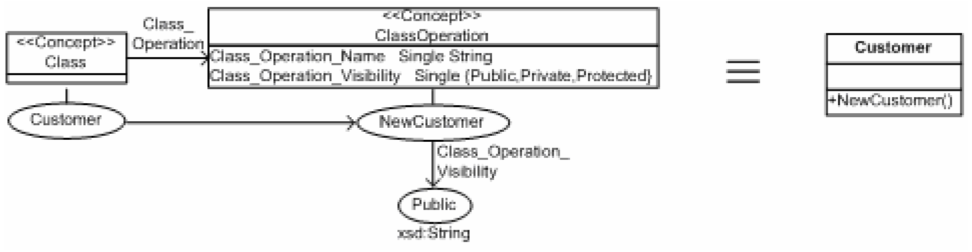

Fig. 5 Transformation of UML class Customer and its operation to class diagrams ontology

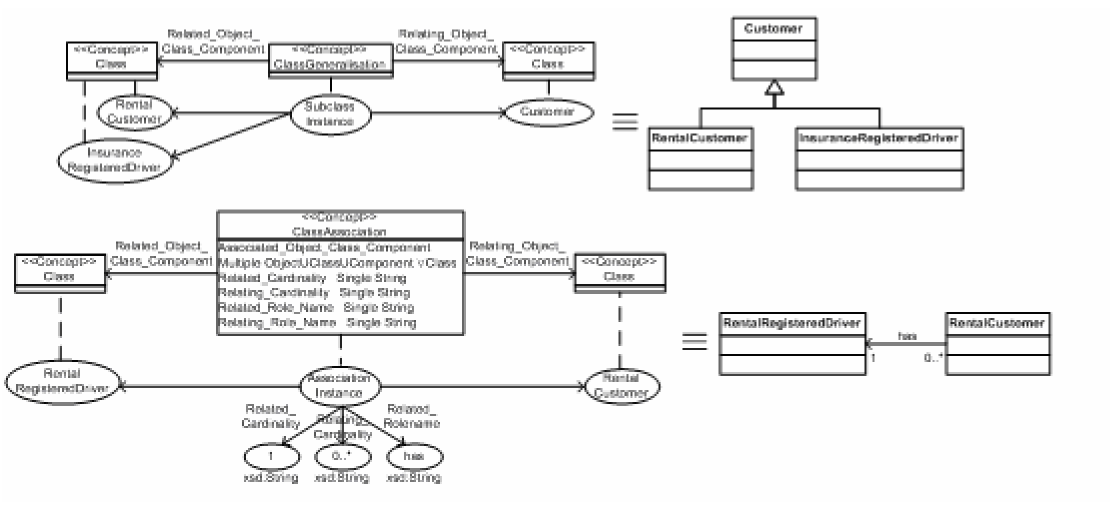
Related_Object_Class_Component
<<Concept>> Class
<<Concept>> ClassGeneralisation
Relating_Object_Class_Component
<<Concept>> Class
Customer
Rental Customer
Subclass Instance
Customer
Insurance RegisteredDriver
RentalCustomer
InsuranceRegisteredDriver
<<Concept>> ClassAssociation
Associated_Object_Class_Component
Multiple Object∪Class∪Component ∨ Class
Related_Cardinality Single String
Relating_Cardinality Single String
Related_Role_Name Single String
Relating_Role_Name Single String
Related_Object_Class_Component
<<Concept>> Class
Relating_Object_Class_Component
<<Concept>> Class
Rental RegisteredDriver
Association Instance
Rental Customer
Related_Cardinality
Relating_Cardinality
Related_Rolename
1
0..*
has
xsd:String
xsd:String
xsd:String
RentalRegisteredDriver
has
1
0..*
RentalCustomer

Fig. 6 Transformation of generalisation and association relationships to class diagrams ontology

For the particular class diagram shown in figure 3, the concepts of generalisation relationship and association relationship are applied. An instance of concept ClassGeneralisation has relations Related_Object_Class_Component with concept Class instances RentalCustomer and InsuranceRegisteredDriver and has relations Relating_ Object_Class_Component with concept Class instance Customer. Instance of concept ClassAssociation has relations Related_Object_Class_Component with concept Class instance RentalRegisteredDriver, has relations Relating_ Object_Class_Component with concept Class instance RentalCustomer, has relations Related_Cardinality with xsd:String of ‘1’, has relations Relating_Cardinality with xsd:String of ‘0..*’, and has relations Related_Role_Name with xsd:String of ‘has’. These are shown below in figure 6.

All project data/agreements which are instantiations of the software engineering ontology need management to promote the use of semantic project data for multi-site distributed software development.

3. Software Engineering Ontology Instantiations Management

Management tasks for software engineering ontology are assigned to the systems containing a number of sub systems. There is a set of systems to facilitate management of software engineering ontology named safeguard system, ontology system, and decision maker system. The architecture of the systems in the multi-site environment is shown in figure 7.

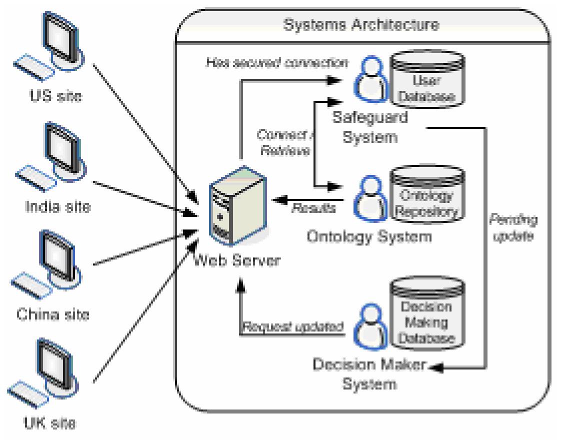

Fig. 7 Model of Management Systems

Team members, regardless of where they are, connect to the web server via a web browser. This will enable team members to directly use the system without having to download any software or install any application. Each team member is served by the intelligent systems tool as the communication media. This allows direct communication between different team members using a messaging system and allows monitoring and recording of the activities of the team members as well. Each team member is provided with a particular set of access privileges that are dependent of the role of that team member in the project. The set of sub systems within the intelligent support systems architecture is comprised of: safeguard system, ontology system and decision maker system. The safeguard system represents system authentication for user authorisation and determination of the access level. The safeguard system communicates with the ontology system if the user / team member wants to query or update the software engineering ontology. The ontology system manipulates and maintains the software engineering ontology repository. The decision maker system operates tasks if an operation needs to be certified. The decision maker system is responsible for decision making on the matter of updating the software engineering ontology including acknowledgement of the decision made to all involved team members. As can be seen from the model of management systems, only the safeguard system has any connection with the user database. This means that the safeguard system performs all recording of user activities as well. All other systems call the safeguard system and pass the information to log all the events that the system carried out. Thus, tracking can be accomplished by the safeguard system if needed. Not only does it allow tracing; the safeguard system can determine bottlenecks, if there are any occurrences, with the use of the timestamp. The ontology system is the only one manipulating the software engineering ontology. Thus, it is the only one that has access to the ontology repository. All other systems contact the ontology system in the cases of wanting to view, query, or update the ontology. The decision maker system has its own database to store data for decision making occurring in the systems.

In detail, the functionalities of each system can be observed in figure 8. 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 (figure 8, number 3). A team member can log into systems. Once logged in, system authentication (figure 8, 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 (figure 8, 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 (figure 8, number 3-5). Manipulation of instantiations is functioned by the ontology system and is recorded into user database by the safeguard system (figure 8, number 7). Solution proposal is managed by safeguard system (figure 8, number 6). Any decision is made by decision maker system (figure 8, number 8).

4. Safeguard of Software Engineering Ontology

To implement security features into the systems, it has been decided to appoint the safeguard system. The safeguard system implements and enforces the systems' authentication, the access control policies and member activity log. All these operations have different logic involved whose details are given in the next following sections.

4.1 Software Engineering Ontology Access Authentication

Once users log into the system, the user identification will be verified with the user database handled by relational database. Once authorised, the user can access, modify or update the ontology depending on the access privileges held by the user whose details are given in the next section.

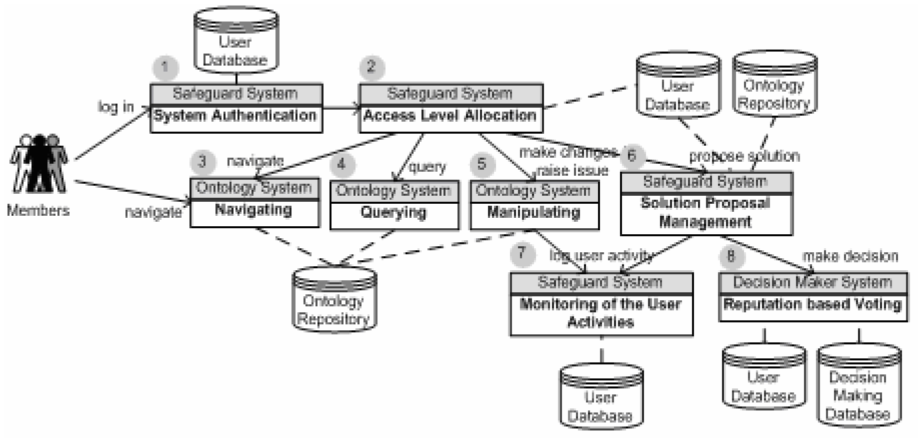

Fig. 8 Functionalities provided in each system

There are a few security levels, mainly (i) Software engineers – they can be software engineers, who have no access to updating, but can only look up, or software engineers who have an access to update project data, and (ii) team leader / project manager – they have unlimited access to all functions such as updates, backups, ontology maintenance and database access. It should be noted that only the team leader / project manager has access to the database server. Team leaders, who do not have any of the access levels stated, will not have access to the specific section of the ontology.

4.2 Determination of the Software Engineering Ontology Access Level

A user logged into the systems after authentication will be provided with different services according to different access levels. All team members are provided with the service to query the software engineering ontology. The different access levels are (i) querying level – only querying service that allows no changing, (ii) add and modifying level – restricted access to add and modify service of the software engineering ontology instances (project data). At this level, some operations may be required to be hold through making decision system e.g. request for revision of project design model. Simple updates like the status of the project or documentation update, would immediately be updated to the software engineering ontology, and (iii) full access level – unrestricted access to all services provided.

These access levels are given according to the different status of the team members. The hierarchy of the software engineering sub-ontology is used to categorise hierarchy to in order to determine the access to the status of team members. The team leader is the one who assigns proper access privileges to each member in the team. For example, the sub-ontology ‘software design’ would require a designer team or the sub-ontology ‘software requirement’ would demand an analyst team to access, add, and modify their project data. Nevertheless, the team designer can look up project requirements through the sub-ontology ‘software requirement’ but no changes have been allowed, which means they are on the querying level. The team leader will have full access including monitoring team member activities. The process of viewing, querying and manipulating software engineering ontology is done by the ontology system. The safeguard system will only verify the authorisation, the access level and log activities and then pass the request to the ontology system.

4.3 Monitoring of the User Activities

Every single completed activity will be recorded by the logger application resided in the safeguard system. This allows monitoring of all the user activities. The safeguard system is requested by any other systems i.e. ontology system and decision maker system to record team member activities. Because the safeguard system is the only system that connects to the user database; thus, if there is any action to log into the database, it will be done by the safeguard system.

4.4 Solution Proposal Management

In a software engineering project, once issues arrive, one can raise these issues with the team and the suggested solutions can be proposed by any one in the project team. All team members then can vote for either the selected solution, or they can support the original. This is like a communication tool that allows a project team member to voice opinions or suggestions on the particular issue that has arisen. Multiple solutions can be proposed for a single issue. After voting, the decision maker system whose details are in the later section, operates decision making showing the proposed solution that has been chosen and acknowledged by the team. Functionalities of a solution proposal management therefore include raising an issue, proposing possible solutions, voting for a final solution and retrieving all parts of an issue and its pending proposed solutions. These functionalities are assigned to the three representing platforms which will be explained in the later sections.

5. Software Engineering Ontology Repository 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.

There are three different services here in the ontology system: navigating, querying and manipulating services which are given in the next sections.

5.1 Navigation of Software Engineering Ontology

In this section, we deal with the accessing of information held in the software engineering model. 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.

5.2 Query of Software Engineering Ontology

The previous section dealt with the case of navigating the software engineering model. This section deals with searching the software engineering model. As stated in the earlier section, 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.

5.3 Manipulation of Software Engineering Ontology

This section deals with manipulating the software engineering model. 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.

5.4 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.

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.

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.

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.

As mentioned in Chapter 5, 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.

6. Decision Support

As the name itself suggests, the job of the decision maker system is to make a decision on an issue such as a major update request. In this study, we have developed a combination of two techniques to implement the decision maker system. The decision making is based on members in the teams agreeing to vote, along with the reputation of each individual member involved in the software engineering project. In the following sub sections, details of the voting and reputation techniques are presented. We illustrate the combination of both techniques.

6.1 Voting System

The voting system provides a means for making changes to the reflected project data or instance knowledge in the software engineering ontology based on votes from each member of the project teams. Every member in the teams involved in a given software engineering project has a right to vote for proposed solutions. Everybody's vote is worth points. Below is a list of requirements for the voting system.

- A member can work on a project or multiple projects at the same time
- A member can work in a team or in multiple teams
- A member can work in different teams in different projects
- A project involves multiple teams and multiple members

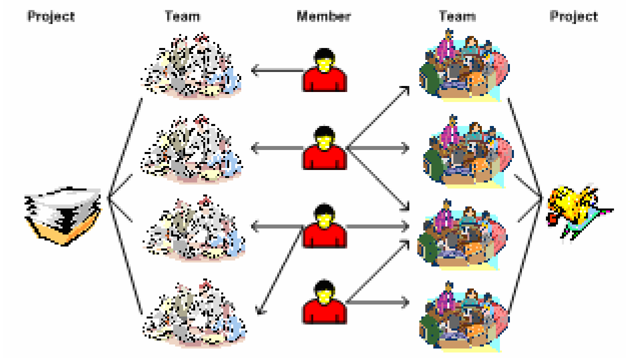

Fig. 9 An example of members working in project teams

The vote cast by each team member is mathematically weighted by the factor of which 'members who actually work on a task have the best understanding of that task'. In other words, if a member votes on an issue which arises within the area he/she is working on, presumably this falls within his/her area of expertise, then his/her vote carries more weight than that of a member who does not have expertise in the issue area, or who does not really work on it. There are four areas of expertise categorised by following four software processes in the software engineering domain. These are software requirement, software design, software construction, and software testing. Members classified in these areas of expertise are analysts, designers, programmers or implementers, and tester respectively. Figure 9 gives an example of this classification. A member, for example, may work in the design team for a particular project and may also work in the requirements team and construction team in another project. It is assumed, for example, that the designers of a project who work on the project design, have expertise in project design, or know more than others do about about this aspect of a project. Thus, if the project issue relates to project design, the votes of members in the design team carry more weight than others.

Table 1 shows an example of three possible solutions named A, B, and C in the issue of project design. Let us assume that solution A received a single vote from a designer, solution B received a single vote from an analyst, and solution C received a single vote from a programmer. The vote of each of these people is weighted by their expertise in the area of the issue. From the above description, solution A then would receive the maximum vote points since it has been voted by a member in the design team who, it is assumed, has some degree of expertise in the project design because he/she actually works on it.

Table 1: An example of the three possible solutions in the issue of project design

Solution	Requirement (y)	Design (x)	Construction (y)	Testing (y)	Design Issue Voting Points ($x>y$)
A		√			$1\times x$
B	√				$1\times y$
C			√		$1\times y$

6.2 Reputation System

The reputation based system provides a means for making the changes to the reflected project data in the software engineering ontology based on the reputation of the team members involved in the software engineering project. Below is a list of requirements for the voting system.

- A member has a reputation value for a particular area or domain in a given project
- A member can have a different reputation value for a different area or domain in a different project
- The reputation value of the team member continues to increase if the team member votes for the chosen (or correct) solution and vice versa
- The reputation value of the team member decreases if the team member did not vote for the chosen (or correct) solution and vice versa.

The reputation value of members may change with time. In other words, at a given time and in a particular area, reputation value may increase, decrease or remain the same. Figure 10 shows as example of different reputation values in the different areas of a member working on different projects. Using the Markov Model [7], the change in the reputation value of each team member in a given phase is tracked. Additionally, using the Markov Model, we consider what could be the most probable future reputation value of a given team member in the category of the issue at a time in which the decision has to be made.

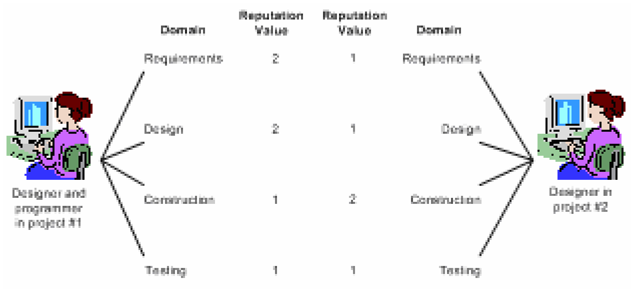

Fig. 10 An example of different reputation value in the different area of expertises of members working in different project

The calculation of a user's reputation value, which is a value of either 1 or 2, is based on the past reputation points for different domains. In order to calculate a user's reputation value, the first step is to calculate the current state value (CSV) which is the latest up-to-date reputation value; second, calculate the Markov matrix; and third, multiply CSV with the Markov matrix in order to arrive at the determined reputation value of the user [7]. Along with the explanation, we will provide an example for further clarity. If the last reputation value is a 2, the CSV is a matrix [0 1]. If the last reputation value is a 1, the CSV is a matrix [1 0]. There can be only these two possibilities. For example, a set of reputation value list is {2, 2, 1, 2, 2, 2}. As can be seen, the last value is a 2 then the CSV for that member is a matrix [0 1]. Once CSV has been found, the Markov matrix is calculated next. The transition states matrix is needed. Since there are only 1s and 2s, there are four states of transition namely: 1-1 state, 1-2 state, 2-1 state and 2-2 state. By counting the numbers of each state, we form the transition states matrix. As from the example, there is none of 1-1 state; there is one 1-2 state; there is one 2-1 state and there are three 2-2 states as shown in Eq. 1.

$$\text{Transition States Matrix} = \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} \qquad (1)$$

By counting a frequency of transition from state 1, in the above example state 1 transition frequency is one and state 2 transition frequency is four. This is used to calculate the percentage of whether it changes state or stays at the state. For the above example, 1-1 state has 0/1 that means 0% or 0, 1-2 state has 1/1 that means 100% or 1, 2-1 state has 1/4 that means 25% or 0.25 and 2-2 state has 3/4 which means 75% or 0.75. From here the Markov matrix is shown in Eq. 2.

$$\text{Markov Matrix} = \begin{bmatrix} 0 & 1 \\ 0.25 & 0.75 \end{bmatrix} \qquad (2)$$

By multiplying the Markov matrix with the CSV, we will be able to obtain the reputation value of the user. The probability of the reputation value is given in Eq. 3.

Reputation Value Probability =

$$\begin{bmatrix} 0 & 1 \\ 0.25 & 0.75 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.75 \end{bmatrix} \qquad (3)$$

The probability of the reputation value is in the form of [a b]. If the value a is greater than b, it means the most probable reputation value will be a 1 and if value b is greater than a, it means the most probable reputation value will be a 2. Therefore, in this example, the reputation value probability of [0.25 0.75] where 0.75 is greater than 0.25, shows that the user's reputation value is worth 2.

6.3 Reputation based Voting for Making Decision

Whenever issues arise, such as a major update request of project data, the decision making system sends a message to every team member asking him/her for an opinion. Subsequently, it then gathers and stores the possible solutions for that particular issue. Of all the possible solutions, one solution is chosen by asking all team members to vote for one of the proposed solutions. The reputation based decision will then determine the total number of votes for each solution and, as mentioned earlier, each vote is weighted by the expertise of the person casting it in the area of the problem. Additionally, the reputation value of individual member who votes is weighted.

For example, assume that a weighting value for member who his/her expertise is not in the area of the issue is 0.2 and a weighting value for member who his/her expertise is in the area of the issue is 0.8. Let us follow the previous example of the three possible solutions named A, B, and C on the project design issue. Let assume that for the design area, the reputation point of a member who votes for solution A is 1. Since this member's expertise is in the area of design, which is the area where the issue is raised, (i.e. project design), this member's vote would have a value of 0.8 (1 multiplied by 0.8). If the reputation value of a member who votes for solution B is 2, then this member's vote would have of value 0.4 (2 multiplied by 0.2) because this member's expertise area is requirement (this member is an analyst) while the issue is about project design. Similarly, if the reputation value of member who votes for solution C is 1, then this member's vote would have a value of 0.2 (1 multiplied by 0.2) because this member's expertise area is construction (this member is a programmer) while the issue is about project design. Table 2 shows the calculation of the voting points.

Table 2: An example of voting point calculation

Solution	Requirement	Design	Construction	Testing	Design Issue Voting Points
A		0.8 x 1			0.8
B	0.2 x 2				0.4
C			0.2 x 1		0.2

For a particular issue, whichever solution has the highest vote value will be chosen. Therefore, from the example, solution A that has the highest vote value is chosen as a final solution. Once a solution has been chosen and finalised, the project data in the software engineering ontology will be updated along the lines of the chosen solution. The system advises all team members of the final decision and records the event. The users' reputation points are also updated for future use.

7. 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.

References

[1] Wongthongtham, P., A methodology for multi-site distributed software development, in School of Information Systems. 2006, Curtin University of Technology: Perth.

[2] Wongthongtham, P., et al., Ontology-based multi-site software development methodology and tools. Journal of Systems Architecture, 2006. 52(11): p. 640 - 53.

[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 [cited.

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

[7] Chang, E., T. Dillon, and F.K. Hussain, Trust and Reputation for Service Oriented Environment: Technologies For Building Business Intelligence And Consumer Confidence. 2006: John Wiley and Sons.

Dr. Pornpit Wongthongtham received the M.Sc. in Computer Science from Chulalongkorn University, Thailand, and PhD degrees in Information Systems from Curtin University of Technology, Australia, in 1999 and 2006, respectively. Currently, she is working as a research fellow at the Digital Ecosystems and Business Intelligence Institute (DEBII), Curtin University of Technology.