Enhancing Software Engineering Project Information through Software Engineering Ontology Instantiations

P. Wongthongtham[1], E. Chang[1], T.S. Dillon[2]
[1]*School of Information Systems, Curtin University of Technology, Australia*
{pornpit.wongthongtham, elizabeth.chang}@cbs.curtin.edu.au
[2]*Faculty of Information Technology, University of Technology Sydney, Australia*
tharam@it.uts.edu.au

Abstract

Software engineering project information is frequently evolving and queried 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 and the like. Therefore, the project information needs enhancement to ease up-to-date ontological information and to ease communication. Ontologies are widely used for capturing and organising knowledge of a particular domain of interest. We propose the use of software engineering ontology instantiations and enrichment to capture the software engineering project information.

1. Introduction

There is no doubt that currently the internet is the richest repository of information. However, semantics of the information on the internet are oriented to humans rather than to machines. We propose to enhance information with semantics. Moreover, we propose to enhance software engineering project information by associating the project information with specific entities within the domain of interest. We aim to facilitate a semantic-based interpretation of content by restricting their models of interpretation through software engineering ontology. Software engineering ontology captures software engineering knowledge conceptualisation. The software engineering ontology organises and centralises software engineering knowledge in a formal, machine and human understandable way. Machine in form of a software application or a software agent can use the knowledge regarding project information as instances in the ontology to carry out knowledge maintenance.

In the next section, we provide information concerning the software engineering ontology. Then in section 3, we describe software engineering project information enhancement. We present software engineering instantiations and enrichment in section 4. In section 5, we illustrate instantiations transformation and, finally, we conclude this paper in section 6.

2. Software engineering ontology

We have merged Gruber's [1], Borst's [2] and Studer's [3] definitions of ontology as a basis to define software engineering ontology. Hence, the software engineering ontology is a formal, explicit specification of a shared conceptualisation in the domain of software engineering. '*Formal'* implies that the software engineering ontology should be machine-understandable. Software engineering ontology enables a better communication over software engineering domain knowledge between humans and machines. '*Explicit'* implies that the type of software engineering concepts used and their constraints are explicitly defined. Software engineering ontology standardises and formalises the meaning of terms in software engineering through its concepts. '*Shared'* shows that the ontology specifies consensual knowledge of software engineering which means it is public and accepted by a group of software engineers. '*Conceptualisation'* implies an abstract model of having identified the involved software engineering concepts.

It is not necessary that ontology has instances but software engineering ontology has the instances representing project information which includes project data, project understanding and project agreement. Figure 1 shows a schematic view of the software engineering ontology.

Proceedings of the 2006 IEEE/WIC/ACM International Conference
on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)
0-7695-2747-7/06 $20.00 © 2006 IEEE

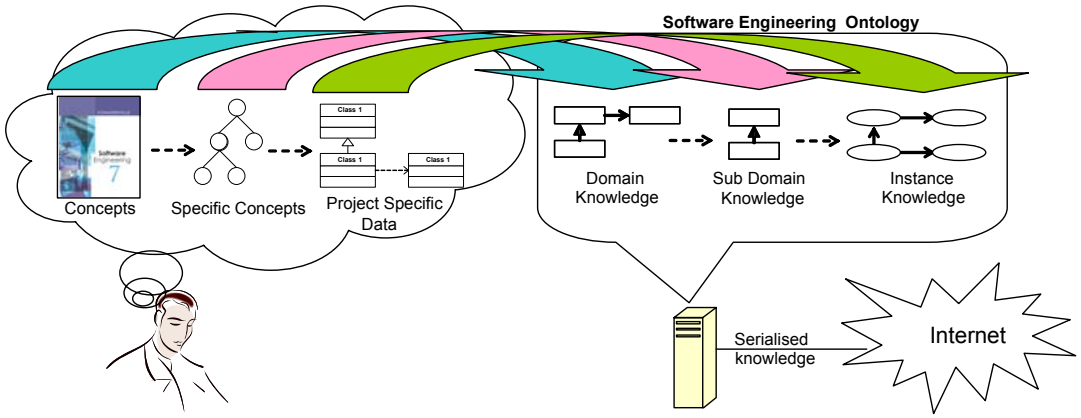

Figure 1. Schematic overview of software engineering ontology

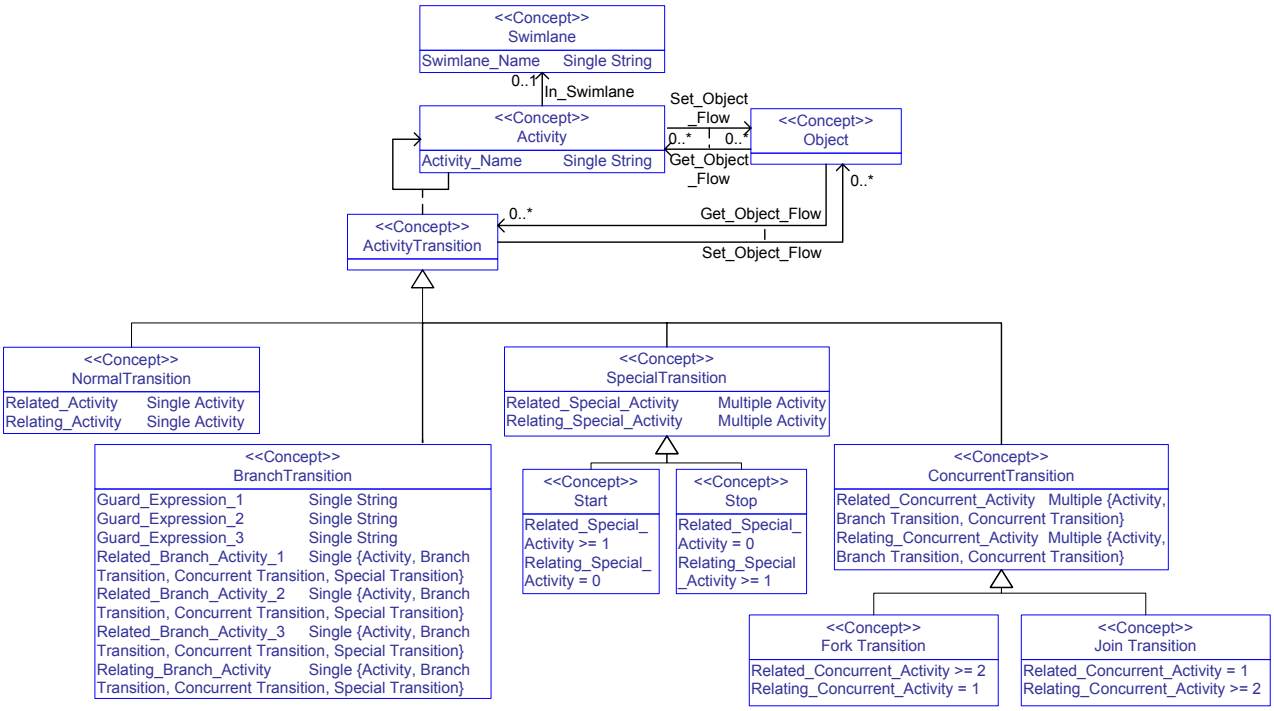

Figure 2. Schematic overview of software engineering ontology

The whole set of software engineering concepts are captured into generic software engineering ontology as domain knowledge. A particular project or a particular software development probably uses only part of the whole set of software engineering concepts. For example, if a project uses purely object oriented methodology then the concept of a data flow diagram may not be necessarily be included but instead it includes concepts like class diagrams, activity diagrams and so on. The specific software engineering concepts used for the particular software development project are captured in the specific software engineering ontology as sub domain knowledge. The generic software engineering ontology represents all software engineering concepts while specific software engineering ontology represents some concepts of software engineering that the particular project needs. Then in each project there exists project information or actual data including project understanding and project agreement. The project information specially meets a particular project need and is required for the software engineering ontology to define instance knowledge. Note that domain knowledge is separated from instance knowledge. The instance knowledge varies depending on its use for a particular project. Domain knowledge is quite certain while the instance knowledge is vague as per the project. Once all the domain knowledge, sub domain knowledge and instance knowledge are created, it is available to be

shared among software engineers through the internet. All team members, regardless of where they are, can use the semantic linked project information. .

The software engineering ontology that was used in case study for this paper, concerns only in the domain of software process design. The full version of software engineering ontology can be found in Wongthongtham's thesis [4]. The software engineering ontology was constructed using the Protégé OWL [5, 6]. Ontology models that appeared in this paper use notations developed in Wongthongtham's thesis [4].

In order to capture knowledge, we need to firstly define the main **concepts** of interest. Secondly, **relations** for each concept which links it to other concepts, known as object property, or to an XML schema data type value (boolean, float, integer, or string), known as datatype property are defined. Thirdly, **constraints** on the range and then **instances** of the concepts are defined.

Figure 2 shows an ontology model of activity diagrams. For example, in domain of activity diagrams in software process design, the main concepts are activity, its transition, swimlane and object. Every activity can be in a swimlane, however, transition may occur between lanes. This refers to a maximum cardinality restriction in relation *in_Swimlane*. Objects may be involved in the flow of control associated with an activity diagram. This refers relations *set_Object_Flow* and its inverse, *get_Object_Flow*.

Transitions of activities are classified into four main areas. Firstly, normal transition shows the path from one activity to the next activity. This refers ontology class *NormalTransition* that has cardinality restriction restricted the only one activity in the relations *Related_Activity* and *Relating_Activity*. Secondly, special transition is further divided into an initial and a stop transition. The initial transition is where the activity diagrams start. This refers ontology class *Start* that has cardinality restriction restricted at least one activity in relation *Related_Special_Activity* but no activity in relation *Relating_Special_Activity*. The stop transition is where the activity diagram stops. This refers ontology class *Stop* that has cardinality restriction restricted at least one activity in relation *Relating_Special_Activity* but no activity in relation *Related_Special_Activity*. Thirdly, branch transition which specifies alternate paths taken based on some guard expression refers to ontology class *BranchTransition*. Lastly, concurrent transition is further divided into a fork and a join transition. The fork transition represents the splitting of a single flow of control into two or more flows of control. This refers to ontology class *ForkTransition* that has cardinality restriction restricted at least two activities in relation *Related_Concurrent_Activity* and only one activity in relation *Relating_Concurrent_Activity*. The join transition represents the joining of two or more incoming transitions and one outgoing transition. This refers to ontology class *JoinTransition* that has cardinality restriction restricted at least two activities in relation *Relating_Concurrent_Activity* and only one activity in relation *Related_Concurrent_Activity*.

Once project members are committed to the domain knowledge of, for example, activity diagrams and recognise constitutes of activities, transitions, constraint of activities and transitions and so forth then the commitment enables project members to talk or discuss in the same language. Consequently project members can better coordinate their activities.

3. Software engineering project information enhancement

In this paper, we focus on the semantic increase of software engineering project information concerning the instances that exist in a domain of interest. We hope it eliminates misunderstandings, miscommunications and misinterpretations and provides semantic consistency. Software engineering ontology presents explicit assumptions concerning the objects referring to domain knowledge of software development. A set of objects and interrelations and their constraints renders their agreed meanings and properties. For example, the confusing terms of 'classes', 'objects' and 'components' in object oriented software development can be simplified to terms which the software engineers agree to recognise their constitutes, their interrelations and their constraints. Conclusively determining what concept of project information is captured or where that project information resides it is assumed that it is determined by members who specify of what the project information really means in that given context.

4. Software engineering ontology instantiations

As stated previously, software engineering ontology contains abstraction of software engineering domain concepts and instantiations. There are two types of the abstraction which are generic software engineering and specific software engineering. The abstract of a generic one represents the whole software engineering concepts while the abstract of a specific one represents the software engineering concepts used for some particular projects. The instantiations, also known as population, represent the project information. The abstraction of the specific software engineering ontology is having its instantiations being used to capture data instances of the projects. Each abstraction can have multiple instantiations in different circumstances of projects. The corresponding concrete data instances are stored as

Proceedings of the 2006 IEEE/WIC/ACM International Conference
on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)
0-7695-2747-7/06 $20.00 © 2006 IEEE

instantiations. In this study, the software engineering ontology integrates abstractions and instantiations together, rather than separating them by storing instances in the traditional, relational, database style inside the knowledge base. The latter SQL queries can help in the large volume concept and data management and maintenance. Nevertheless, in the software engineering ontology the data volume is not very large and coherent integration between abstraction and instantiations are important in the software engineering projects. Combining them rather than having them separate 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. As you can see, this example scenario strongly favours the combined abstractions and instantiations for storing because it asks for a unified global declaration of abstractions.

5. Instantiations transformation

In this section, we present how to populate the software engineering ontology with instances. Populating refers to the process of creating instances of corresponding concepts in software engineering ontology.

Particularly, software engineering project information is transformed or mapped into corresponding concepts formed in the software engineering ontology as instantiations. Once transformed, instantiations are available to be shared among project members. Manipulation of semantic linked instantiations can be carried by project members.

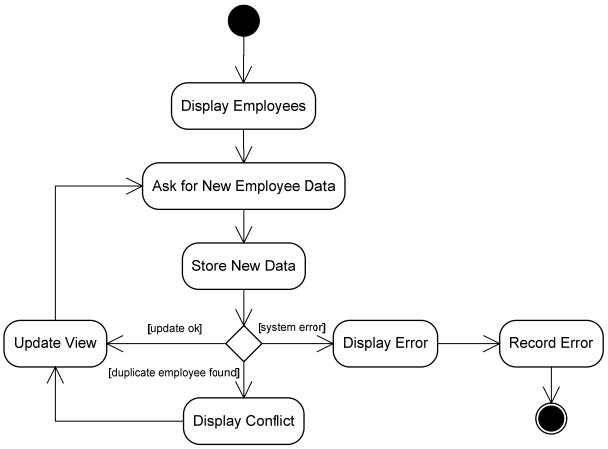

Figure 3. An example of activity diagram

We use the example of activity diagrams in domain of software process design. Figure 3 shows an example of UML activity diagram that will be transformed into activity diagram ontology (its model shows in Figure 2) as instantiations. Note that the activity diagram used as an example here is derived from the book of Enterprise Java with UML [7].

As from Figure 3, a list of updating actions is as follows:

- Adding new instances 'display employees', 'ask for new employee data', 'store new data', 'update view', 'display error', 'record error' and 'display conflict' for concept *Activity*.
- Adding new instance for concept *StartTransition* relating relation *Related_Special_Activity* with concept *Activity* instance 'display employees'.
- Adding new instance for concept *StopTransition* relating relation *Relating_ Special_Activity* with concept *Activity* instance 'record error'.
- Adding new instance for concept *NormalTransition* relating relations *Relating_Activity* with concept *Activity* instance 'display employees' and *Related_Activity* with concept *Activity* instance 'ask for new employee data'.
- Adding new instance for concept *NormalTransition* relating relations *Relating_Activity* with concept *Activity* instance 'ask for new employee data' and *Related_Activity* with concept *Activity* instance 'store new data'.
- Adding new instance for concept *NormalTransition* relating relations *Relating_Activity* with concept *Activity* instance 'display error' and *Related_Activity* with concept *Activity* instance 'record error'.
- Adding new instance for concept *NormalTransition* relating relations *Relating_Activity* with concept *Activity* instance 'display conflict' and *Related_Activity* with concept *Activity* instance 'update view'.
- Adding new instance for concept *NormalTransition* relating relations *Relating_Activity* with concept *Activity* instance 'update view' and *Related_Activity* with concept *Activity* instance 'ask for new employee data'.
- Adding new instance for concept *BranchTransition* relating relations *Relating_Branch_Activity* with concept *Activity* instance 'store new data', *Related_Branch_Activity_1* with concept *Activity* instance 'update view', *Related_Branch_Activity_2* with concept *Activity*

instance 'display conflict', *Related_Branch_Activity_3* with concept *Activity* instance 'display error', *Guard_Expression_1* with string of 'update ok', *Guard_Expression_Activity_2* with string of 'duplicate employee found' and *Guard_Expression_Activity_3* with string of 'system error'.

Project information, which is instantiations of the software engineering ontology, promotes the use of semantic project information for software development. Having attached domain knowledge, it makes project information more understandable, more linear, predictable and controllable as members identify some missing pieces that make sense of your attentive interaction among team members.

6. Conclusion

We have presented semantically software engineering project information enhancement. We have illustrated association of the project information with specific entities within the domain of interest. This aims to facilitate a semantic-based interpretation of content by restricting their models of interpretation through software engineering ontology. We have given ideas for the use of software engineering ontology instantiations and enrichment to capture the software engineering project information. An example of software engineering ontology capturing software engineering knowledge conceptualisation has been illustrated. The software engineering ontology organises and centralises software engineering knowledge in a formal way that is understandable for both machines and humans. A machine, in form of a software application or a software agent, can use the knowledge regarding project information as instances in the ontology to carry out knowledge maintenance.

However, the populating process is a time-consuming, error-prone and labour intensive task when performed manually. For future work, we will investigate systems to facilitate the ontology instantiations and enrichment process in order to obtain knowledge from data automatically.

7. References

[1] Gruber, T.R. *Toward principles for the design of ontologies used for knowledge sharing*. in *International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*. 1993. Padova, Italy: Kluwer Academic Publishers, Deventer, The Netherlands.

[2] Borst, W., *Construction of Engineering Ontologies*. 1997, Centre of Telematica and Information Technology, University of Tweenty: Enschede, The Netherlands.

[3] Studer, R., V. Benjamins, and D. Fensel. *Knowledge Engineering: Principles and Methods*. in *IEEE Transactions on Data and Knowledge Engineering*. 1998.

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

[5] Horridge, M., *A Practical Guide To Building OWL Ontologies With The Protege-OWL Plugin*, 1.0, Editor. 2004, University of Manchester.

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

[7] Arrington, C., *Enterprise Java with UML*. 2001, New York, USA: John Wiley & Sons, Inc.