Ontology Model of Software Engineering for Multi-Site Software Development

Kalyana Chakravarthy Dunuku1
Software Engineer1

Komal2
Assistant Professor2

V. Saritha3
Assoc. Professor3, @ SKEC

ABSTRACT: Ontology is an important concept for software engineering to formally represent knowledge in a way software can process the knowledge and reason about it. The software engineering ontology assists in defining information for the exchange of semantic project information framework. This paper gives an analysis of what software engineering ontology is, what it consists of and what it is used for in the form of usage example scenarios. The usage scenarios presented in the paper highlight characteristics of the software engineering ontology. The software engineering ontology assists in defining information for the exchange of semantic project information and is used as a communication framework. Its end users are software engineers sharing domain knowledge as well as instance knowledge of software engineering.

Keywords:- Software Engineering, Ontology Development, Multi-site Software Development Knowledge Sharing and Knowledge Engineering.

1. INTRODUCTIONS

Having realized the advantages of multi-site software development, major corporations have moved their software development to countries where employees are on comparatively lower wages. It is this imperative of financial gain that drives people and businesses to multi-site development and the Internet which facilitates it. Software development has increasingly focused on the Internet which enables a multi-site environment that allows multiple teams residing across cities, regions, or countries to work together in a networked distributed fashion to develop the software.

The development of the software in various fields, have become more cushy and comfortable. Realizing the pros of multisite software development, major MNC’s and the corporate sectors have moved their business to the countries where the employees work for curtail and pare salaries. Software development has increasingly focused on the Internet, which enables a multisite environment that allows multiple teams residing across cities, regions, or to countries to work together in network distributed fashion to develop the software. The effective communication and coordination across multiple sites is highly usefull for the global software development. Team members, leaders and the managers who control, manage different tasks and activities respectively. That may not be located at the same site in a multisite environment.

Consider a scenario of a software development process where the team members work in a particular site and the person who manage them, their team leader is at different site who controls them and collects, integrates the completed modules for further enhancement of the software project.

The team completes their respective module and send to the team leader. They draft their own form for conclusions on the completed module with respect to their culture, customs and traditions which followed in day to day life. It is obvious that they might have not come face to face and never met as they work online. So, strict software engineering principles should be followed, to have a better communication among the teams and the team members.

The incongruity in analysis, design, documentation, presentation, and diagrams could prevent proper access by other stakeholders in a particular software project. Seldom issues of this kind are kept enigmatic. In reference to the above discussed problems, the software engineering has a commonly understood body of knowledge and is an easily learnt subject that includes some of the latest technology and methodology that is easily adopted. As the teams at different sites refer to various texts in the same software engineering domain, each individual have a personal guide and when they communicate with each other their terminology could be quite startling and unusual. This leads to inconsistency and equivocation among the teams. Communication is the real challenge that everyone
face in their daily life and the affective communication is an important part of a successful business. Ontology is an important part of developing a shared understanding across a project to lessen the problems.

2. ONTOLOGY IN SOFTWARE ENGINEERING

The term ‘Ontology’ is derived from its usage in philosophy where it means the study of being or existence as well as the basic categories [1]. Therefore, in this field, it is used to refer to what exists in a system model. An ontology, in the area of computer science, represents the effort to formulate an exhaustive and rigorous conceptual schema within a given domain, typically a hierarchical data structure containing all the relevant elements and their relationships and rules (regulations) within the domain [2].

An ontology, in the artificial intelligence field, is an explicit specification of a conceptualisation [3, 4]. In such an ontology, definitions associate the names of concepts in the universe of discourse (e.g. classes, relations, functions) with a description of what the concepts mean, and formal axioms that constrain the interpretation and well-formed use of these terms [5]. For example, by default, all computer programmes have a fundamental ontology consisting of a standard library in a programming language, or files in accessible file systems or some other list of ‘what exists’.

The whole set of software engineering concepts representing software engineering domain knowledge is captured in ontology. Based on a particular problem domain, a project or a particular software development probably uses only part of the whole set of software engineering concepts.

The actual content and the domain are represented in the fig 1 with Semantic and the Pragmatic representations respectively. The content in the semantics (Actual meanings) area can be Stuff, Things, and Relationships. The Domains in the pragmatic (Dealing or concerned with facts or actual occurrences) area can be Knowledge domain, Applications domain, and Functional domain. Combining both the Content and the Domain knowledge forms the basis for the Ontology. A simple and very regular ontological representation can be a standard library in a programming language environment which has all the methods, attributes, classes and packages that gives the answer for the preliminary question of “What Exists” in a programming language. However, some Representations may be poor due lack quality in design, implementation.

Fig.1 Schematic overview of software engineering knowledge representation.
2.1 Ontologies in the Software Engineering process

From the first day’s Modularisation, distribution, reuse and integration of software components and systems are central SE issues. Mostly tasks are extended and automated, the more important is ontologies as conceptual basis of such components. Recent software development methodologies such as model-driven development (MDD) [12] is favourable models as the central knowledge based from which different implementations may be derived. If systems or components are to exchange knowledge, this will happen more on the model than on the implementation level. In the era of coalescing application landscapes a big potential for defining, implementing, disseminating and using ontologies is evolving.

2.2 Ontologies for the Software Engineering domain.

SE is an own scientific and professional domain with an own structure and terminology. Since it is a rather new domain the task of formulating one ontology/ies for this field is a necessary and challenging task. We mention two approaches: the SWEBoK project in the English-spoken community [3] and the German terminology network (“Begriffsnetz”, [8]). In the rest of this article I will focus on the first point: Ontologies in the Software Engineering process. Software engineering domain knowledge constructs should be sought in ontology, a well-founded model of reality. Ontology is used to analyse the common conceptual modelling constructs [6] which accurately reflect the world. The notion of a concrete thing applies to what software engineers perceive on the basis of software engineering domain knowledge. In this, the notion of ontology is a solution for software engineering knowledge representation. When the knowledge of the software engineering domain is represented in a declarative formalism, the set of software engineering concepts, their relations and their constraints are reflected in the representation which represents knowledge [4]. Thus, the software engineering ontology can be defined by using a set of software engineering representational terms. Then a conclusion from the knowledge of what is can be determined.

The main purpose of the software engineering ontology is to enable communication between computer systems or software engineers[17] in order to understand common software engineering knowledge and to perform certain types of computations. The key ingredients that make up the software engineering ontology are a vocabulary of basic software engineering terms and a precise specification of what those terms mean. For software engineers or computer systems, different interpretations in different contexts can make the meaning of terms confusing and ambiguous, but a coherent terminology adds clarity and facilitates a better understanding. Software engineering ontology has specific instances for the corresponding software engineering concepts[10]. These instances contain the actual data being queried in the knowledge-based applications.

3. Ontologies and Knowledge Sharing

Knowledge-based systems and services are expensive to build, test, and maintain. A software engineering methodology based on formal specifications of shared resources, reusable components, and standard services is needed. We believe that specifications of shared vocabulary can play an important role in such a methodology Several technical problems stand in the way of shared, reusable knowledge-based software. Like conventional applications, knowledge-based systems are based on heterogeneous hardware platforms, programming languages, and network protocols. However, knowledge-based systems pose special requirements for interoperability. Such systems operate on and communicate using statements in a formal knowledge representation. They ask queries and give answers. They take “background knowledge” as an input. And as agents in a distributed AI environment, they negotiate and exchange knowledge[7]. For such knowledge level[5] communication, we need conventions at three levels: representation language format, agent communication protocol, and specification of the content of shared knowledge[9].

Consider the problem of reusing a knowledge-based planning program. Such a program takes descriptions of objects, events, resources, and constraints, and produces plans that assign resources and times to objects and events. Although it may use general planning algorithms, like all knowledge-based systems the planner depends on a custom knowledge base (sometimes called a “domain theory” or “background knowledge[17]”) to get the job done. The knowledge base may contain some knowledge generic to the planning task, and some that describes the domain situations in which the planner is to run. If one wished to use the planning
system, one would need to adapt an existing knowledge base to a new application domain, or build one from scratch. This requires, at a minimum, a formalism that enables a human user to represent the knowledge so that the program can apply it. Furthermore, the developer needs to know the kinds of information given in inputs and returned as outputs, and the kinds of domain knowledge that is needed by the planner to perform its task. If the planning program were offered as a service that could be invoked over the network, or if a large planning problem were subcontracted out to several cooperating planning agents, then one would need an agreement about the topics of conversation that agents are expected to understand[11].

Underlying these content-specific agreements are ontological commitments: agreements about the objects and relations being talked about among agents, at software module interfaces, or in knowledge bases. For instance, developers and users of the planning system might agree on the definitions of objects that “provide,” “require,” and “produce” resources; time that comes in “points” and “durations”; resources that are “allocated,” “deallocated,” and “produced” in “events”; and plans consisting of “state descriptions”, “preferences” over states, and “effects” prescribed by “causal rules”, events, or “operators.”

3.1 Software Engineering Ontology Deployment.

We describe here how software engineering ontology allows knowledge sharing and facilitates the communication framework.

Knowledge Sharing

Knowledge sharing through the software engineering ontology eliminates misunderstandings, miscommunications, and misinterpretations. Software engineering ontology presents explicit assumptions concerning the objects referring to the domain knowledge of software development. A set of objects and interrelations and their constraints renders their agreed meanings and properties. For example, one would like to share design of a use case diagram shown in Figure 2.

The use case diagram used as example here is derived from the book of Enterprise Java with UML [13][14]. Sharing project information drawn based on a consensus of domain knowledge of software engineering formed in the software engineering ontology, makes information explicit. Having attached domain knowledge, it makes project information more understandable, linear, predictable and controllable. Users learn about some missing pieces that make sense of the attentive interaction among users. Alarms can be activated when there are some missing pieces while sharing project information.

Fig.2. An example of use case diagram showing knowledge sharing

4. DESIGN AND EVALUATION OF ONTOLOGY

As information systems play a more active role in the management and operations of an enterprise, the demands on these systems have also increased. The goal of the Toronto Virtual Enterprise Modeling project is to create the next generation enterprise Model, Enterprise has the ability to deduce answers to queries that require relatively shallow knowledge of the domain.

With in the enterprise engineering project, we are conducting research leading to the creation of an information system to support Enterprise Design and Execution. An enterprise design environment allows for the exploration cost, quality and ability. Much of our effort has been in creating representations of organisation behaviour, Activity, State, causality and time, and the objects they manipulate resources inventory, orders and products.

The goal for any given ontology is to agree upon a shared terminology and set constraints on the objects. We must agree on the purpose and ultimate use of our ontologies. We must therefore provide a mechanism guiding the design of ontology. Such a framework allows a more precise evolution of different proposals for an ontology, by demonstrating the competency of each proposal with respect to the set of questions that arise from the applications. The methodology used in design and evaluation of integrated ontologies. Including the
proposal of new ontologies and the extension of existing ontologies show in below diagram.

Fig. 3 Procedure for ontology design and evaluation

4.1 Software Engineering Ontology Evaluation

The software engineering ontology has been implemented in the OWL and deployed [15] on a platform. It can be accessed at www.seontology.org. This section is devoted to evaluating the software engineering ontology through the deployed ontology on the platform. Software engineering knowledge, formed into software engineering ontology [16], helps communications among team members and provides consistent understanding of the domain knowledge.

Software engineering ontology, together with its instance knowledge, is used as a communication framework within a project. Thereby providing rational and shared understanding of project matters. In the platform, software engineering instance knowledge, in accordance with domain knowledge that is described in software engineering ontology, is extracted. By consulting the software engineering ontology, the platform enables references of software engineering domain knowledge and enables extraction of instance knowledge. For example, class diagrams referred to in the software engineering ontology assert how a set of classes is formed in the diagram. The specification imposing a structure on the domain of class diagrams i.e. elicitation of each class consists of class name, class attributes, class operations and relationships hold with other classes. Using software engineering domain knowledge, together with instance knowledge, the platform dynamically and automatically acts for a certain class instance that the member navigates to retrieve accordingly attribute instances, operation instances, and relationship instances together with the related class instance details. Fig. 4 shows examples of instances of class diagrams ontology that are navigated in the platform. In Fig. 4(a), Classinstance CR_Customer is navigated to consequently retrieve ClassAttribute instances and ClassOperation instances.

ClassRelationship instances can also be navigated to subsequently retrieve Classinstances that hold in the relationship and applicable properties of the relationship. For example, in accessing ClassAssociation instance, Classinstances held in the relationship and properties like role name and cardinalities are automatically retrieved as shown in Fig. 4(b). Similarly, if those Classinstances are accessed, then a list of ClassAttribute instances and a list of ClassOperation instances are retrieved to show its attributes and its operations respectively. In accessing each ClassAttribute instance, details of attribute’s name, attribute’s data type, and attribute’s visibility are shown as referred to ClassAttribute ontology in the software engineering ontology. In Fig. 4(c), navigating ClassAttribute instance CR_CustomerID, its name of ‘Customer ID’, its data type of ‘integer’, and its visibility of ‘public’ can be revealed. The same as ClassOperation ontology referred in the software engineering ontology, in accessing each ClassOperation instance, details of operation’s name, operation’s visibility, and operation’s parameters and parameters’ data type can be retrieved.
5. CONCLUSION

In this paper, we have analysed the characteristics of software engineering ontology. We have then defined graphical notations of modelling software engineering ontology as an alternative formalism. The modelling notations are used to design software engineering ontology. We have only covered some distinguished part of modelling domain knowledge of software engineering as example. The practical software engineering ontology has been implemented and deployed. Deployment has been discussed in aspects of knowledge sharing and communication framework.

The evaluation of the ontology is presented of its useful in practice. Finally, the deployed software engineering ontology applied to the realities of distributed development is given to demonstrate its real value to the software engineering ontology.

There are many improvements that can be made through future work of software engineering ontology evolution. The software engineering domain knowledge make change in the introduction of new concepts, and change in the conceptualisation as the semantics of existing terms that have been modified with time.

The software engineering domain knowledge is mature and has undergone no further changes. Instead, instantiations in the software engineering ontology change with corresponding changes to the ontology.

REFERENCES


