View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by espace@Curtin

SOFTWARE ENGINEERING ONTOLOGIES AND THEIR IMPLEMENTATION

Wongthongtham, P.[1], Chang, E.[2], Dillon, T.S.[3] & Sommerville, I.[4]
[1, 2] School of Information Systems, Curtin University of Technology, Australia
pornpit.wongthongtham, elizabeth.chang@cbs.curtin.edu.au
[3]Faculty of Information Technology, University of Technology Sydney, Australia
tharam@it.uts.edu.au
[4]Department of Computer Science, Lancaster University, Lancaster, UK
is@comp.lancs.ac.uk

ABSTRACT
In this paper, we propose a new approach to software engineering. We organize software engineering concepts, ideas and knowledge along with software development methodologies, tools and techniques into ontologies and use them as a basis for classifying the concepts in communication and allowing knowledge sharing. The explanation of software engineering knowledge formed in our ontologies clarifies the software engineering concepts, thereby making them not only explicit but also aiding in the formalization of a consistent use by team developers. Furthermore, the ontology form can be understood by computers.

KEY WORDS
Software Engineering, Ontology, Ontology Development

1. Introduction

We note that software engineering training and practice vary quite significantly between cities and countries. Some universities' computer science and engineering faculties do not have a subject on software engineering or software engineering methodologies such as object oriented analysis and design in UML.

In many large IT organizations and large IT teams, some software engineer state they have never undertaken a subject called software engineering, however, they are 'software engineers'. We found it can be difficult to communicate between teams and among team members if strict software engineering principles are not understood and followed within the same project team. Also, inconsistency in presentation, documentation, design and diagrams are likely to result, which could exclude other teams or members from thorough understanding. Sometimes they are ignored because they were not understood (such as a diagram using non-standard notation) and clarification is not requested.

The software engineering body of knowledge is commonly accepted and is an easily learned subject using some of the latest technologies and methodologies such as UML which can be easily adopted. However, different teams within the same project could have different books and references on software engineering and while some books are titled software engineering are actually mainly on Java. Some books use other words for 'software engineering' such as 'Code Complete' or 'Object Oriented'. Different members may use books titled IT project management as software engineering books as an independent guide when communicating. Thus, each member of the team may have a varying understanding of terms being used. Many times the issues raised are related to inconsistency in understanding of software engineering theories and practice. Therefore, we present a foundation of software engineering knowledge based on Sommerville's book [1] together with 'Software Engineering, the body of Knowledge' (http://www.swebok.org) [2]. Ontologies are intended for knowledge representation, sharing, management, modeling, engineering and education among others. We organize software engineering concepts, ideas and knowledge, software development methodologies, tools and techniques into ontologies and use them as a basis for classifying the concepts in communication and enabling knowledge sharing.

In this paper, we describe the features of software engineering ontologies and illustrate how software engineering ontologies can be developed, used and customized to assist communication among teams within the same project and allowing knowledge sharing. Section 2 provides details on proposed software engineering ontologies. Section 3 gives an idea of ontology modeling and design. Section 4 describes some ontology representation languages. Section 5 focuses on the ontology development tool – Protégé. Section 6 describes ontology implementation. Section 7 shows our software engineering platform and the final section offers a conclusion and points to ongoing and future work.

2. Software Engineering Ontologies

We propose two ontologies of software engineering: (1) generic ontology and (2) application-specific ontology.
Generic ontology is a set of software engineering terms including the vocabulary, the semantic interconnections,

and some simple rules of inference and logic for software development. We link an object to its semantic description, ontology. The generic ontology provides the vocabulary for the terms in software engineering. The contents of software engineering are annotated with a concept or relationship from the generic ontology. It will enable software engineering content to be machine-readable and man-machine-interoperable.

Application-specific ontology is an explicit specification of software engineering for a particular software development project. This ontology can be used for communication in project agreement providing consistent understanding among project members. Software agents, for instance, can be utilized to extract and or subtract information within the project team.

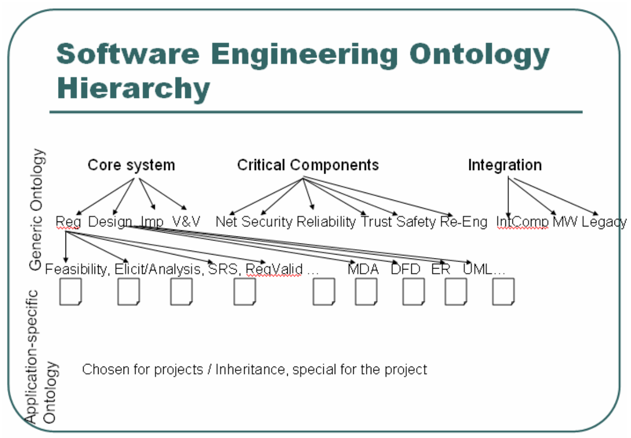

Software Engineering Ontology Hierarchy
Generic Ontology
Core system
Critical Components
Integration
Req Design Imp V&V
Net Security Reliability Trust Safety Re-Eng
IntComp MW Legacy
Feasibility, Elicit/Analysis, SRS, ReqValid ...
MDA DFD ER UML...
Application-specific Ontology
Chosen for projects / Inheritance, special for the project

Figure 1: An overview of generic ontology and application-specific ontology of software engineering

The software engineering discipline covers many aspects of software development, such as core components (business functions and logic), critical components (security, fault tolerance etc.) as well as legacy systems. Since each project is different, they may only need a subset of the software engineering ontology. Therefore, this information resource allows one to generate a subset ontological knowledge about software engineering, known as application-specific ontology. Instances represented by the project specific knowledge, which specifically meet a particular project need, will be put into the application-specific ontology. Note that application-specific ontology and its instances may vary based on its use for a particular project. Figure 1 shows an overview of the generic software engineering ontology and the application-specific ontology of software engineering. The ontological representation of software engineering not only represents the commonly agreed knowledge but also provides detailed relationships (descriptions) between the concepts and associated diagrams, notations systems and templates, such as documents or tools. The application-specific ontology of software engineering can also be customized. Once created, it is available to be shared among the team. All team members are encouraged to obtain knowledge from it through software agents, studying, obtaining answers, classifying knowledge and using it as a basis of conceptual discussion and raising questions including specification, design, implementation and documentation

3. Ontology Modeling and Design

Before development, a designer has to have a model of the conceptual structure of the domain i.e. the ontology as well as an understanding of the structure of information describing instances of these concepts and their relationships [3]. A critical aspect of modeling and designing ontology is lack of graphical notation [4]. We use UML to model ontology. UML object diagrams can be interpreted as declarative representations of knowledge. Instance information can be conveyed as a UML object diagram that shows the values of object attributes and the link i.e. instances of associations that exist between objects. There are benefits for using the same paradigm for modeling ontologies and knowledge. Even standard UML cannot express advanced ontology features such as restrictions, cannot easily conclude whether the same property was attached to more than one class, and cannot create a hierarchy of properties [5]. However, it is a kind of agile modeling method for ontology design. The main aim of this use of UML notation is simply to create a graphical representation of ontologies to make them easier to understand. This use of UML notation to model the underlying ontology should be distinguished from its use in software development to model the application domain model.

4. Ontology Representation Languages

In this section, we provide examples of ontology modeling and designing. It is a portion of application-specific ontology. We model and design concepts of an object class diagram in object-oriented design which is shown in Figure 2. Figure 3 illustrates our ontology modeling and design for the 'use case' representing concept of use case diagram in object-oriented design. For the purpose of brevity of paper, we did not show modeling and design of the following ontologies: 'activity', 'state chart', 'package', 'sequence' and 'collaborative' defining concepts of activity diagram, state chart diagram, package diagram, sequence diagram, and collaborative diagram respectively. There are many ontology representation languages for creating ontology including Knowledge Interchange Format (KIF) [6], Simple HTML Ontology Extension (SHOE) [7], ISO standard for describing knowledge structures (Topic Maps) [8], Ontology Exchange Language (XOL) [9], Ontology Markup Language (OML) [10], Ontology Inference Layer (OIL [11], DAML+OIL [12]) and Web Ontology Language (OWL)[13]. We have chosen OWL because as it has now become the official W3C standard since February 2004 released by the World Wide Web consortium [14].

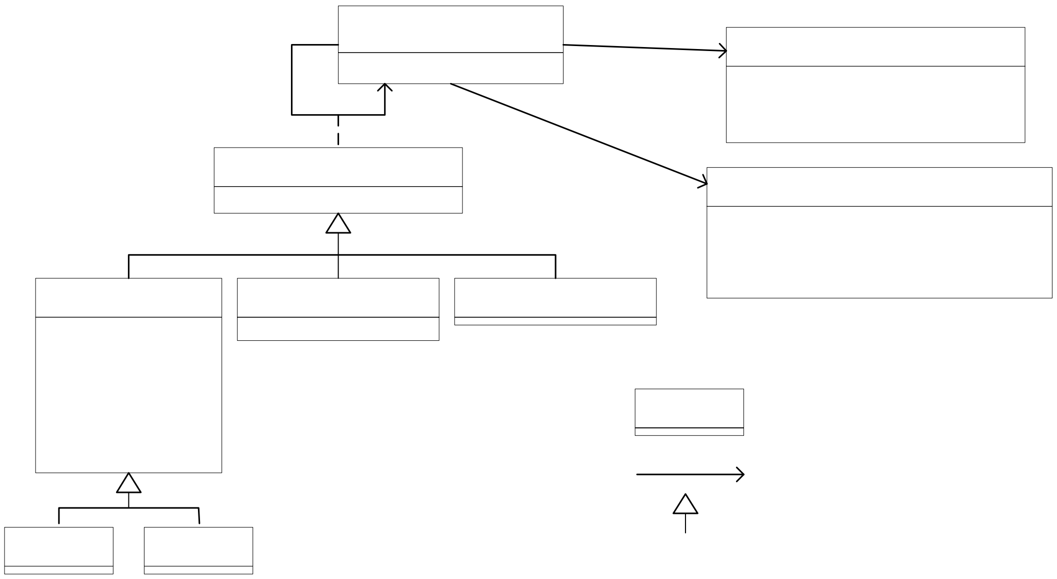


Figure 2: An example of ontology modeling and design describing concept of 'class' diagram in object-oriented design

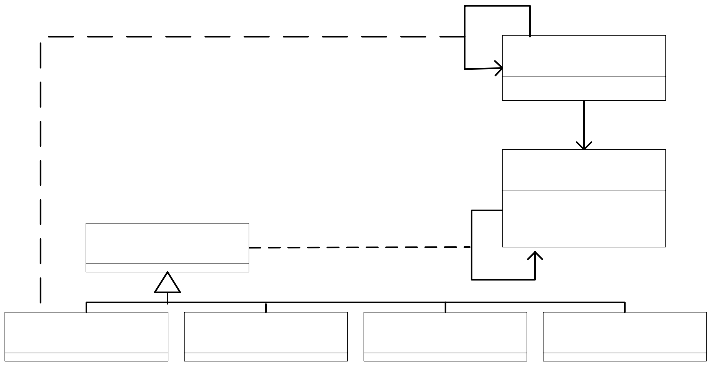


Figure 3: An example of ontology modeling and design describing concept of 'use case' diagram in object-oriented design

Ontologies are used to capture knowledge in some domains of interest. Ontology describes the concepts in the domain and also the relationships that hold among those concepts. Different ontology languages provide different facilities [15]. The most current development in standard ontology languages is OWL. Likewise Protégé OWL makes it possible to describe concepts but it also provides different facilities. It is based on a different logic model which makes it possible for concepts to be defined and described [15].

OWL ontology consists of Individuals, Properties, and Classes.

- **Individuals** represent objects in the domain of interest. Individuals are also known as instances. It can be referred to as being instances of classes or concepts. For example, from an object class diagram in Figure 4 instances or individuals of operation concept of customer class are *NewCustomer*, *SaveCustomer*, *EditCustomer*, *ViewCustomer*, *CancelCustomer*, and *SearchCustomer*.

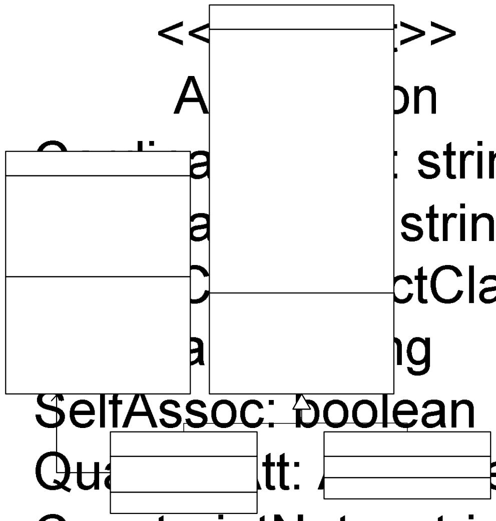


Figure 4: An example of class diagram put into the ontology as instances

- **Properties** are relationships between two things i.e. a concept/individual links to a concept/individual known as *object property* or a concept/individual link to an XML schema datatype value or an rdf literal known as *datatype property*. For example, the object property *hasAttribute* links the concept *ObjectClass* to the concept *Attribute* and the datatype property *ObjectClassName* links the concept *ObjectClass* to string, XML schema datatype. Properties can have inverses. For example, the inverse of *hasRelationship* is *isRelatedTo*. Properties can be limited to having a single value; to being functional or multiple values i.e. to being non-functional. Also, they can be either transitive or symmetric. These property characteristics are explained in greater detail in the next section. Properties are also known as roles in description logics, slots in Protégé, and attributes in UML and other object-oriented notions.

- **Classes** are a concrete representation of concepts interpreted as sets that contain individual(s). Individuals may belong to more than one class. Classes may be constructed in a superclass-subclass hierarchy, which is also known as taxonomy. Subclasses are subsumed by their superclasses. For example, in object-oriented design, association dependency and generalization are all a relationship between object classes. Association, dependency and generalization are subclass of Relationship shown in Figure 2.

5. Ontology Development Tool – Protégé

Protégé is an open-source ontology-development tool developed at Stanford Medical Informatics. It can represent ontologies consisting of classes, properties, property characteristics and restriction and instances. Apart from Protégé there are more leading ontology editors including OntoEdit, OilEd, Chimaera. Protégé has a number of different plug-ins including OWL Plug-in. The OWL Plug-in is a complex Protégé extension which can be used to edit and create OWL files and databases. In this paper, we are using Protégé and OWL Plug-in for developing an ontology. In the next section we will describe creating an ontology of software engineering using Protégé – OWL.

6. Ontology Implementation

In this section, we describe how to create ontology of software engineering.

6.1 Ontology Classes

As can be seen in Figure 2, there are nine classes or concepts to be named and constructed in the hierarchy. The class hierarchy of its nine classes is shown in Figure 5. The class owl:Thing is the class that represents the set containing all individuals. Thereby, all classes are subclasses of owl:Thing. OWL classes are assumed to overlap. Therefore, one cannot assume that an individual is *only* a member of a particular class; it can be a member of more than one class. In order to separate a group of classes, we must make them disjoint from each other. This assures that an individual who has been asserted to be a member of one of the classes in the group cannot be a member of any other class in that group. For example, *Association*, *Dependency*, and *Generalization* have been disjointed from one another. This means that there is no chance for an individual to be an association and dependency and generalization relationship. Likewise, *Attribute*, *ObjectClass*, *Operation*, and *Relationship* have been disjointed also, because individual such as an *Attribute* cannot be individual of either *ObjectClass*, *Operation*, or *Relationship* in the group of *ObjectClassDiagramEntity*.

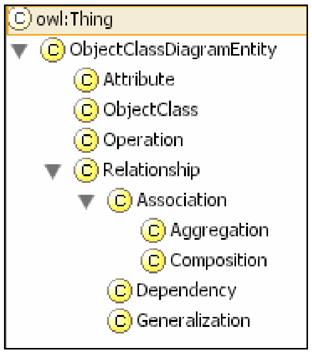
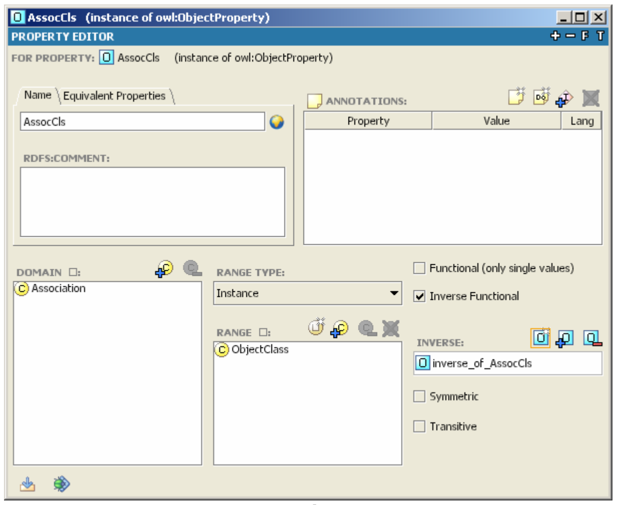


Figure 5: Class hierarchy shown concept of object class diagram in the object-oriented design

6.2 Ontology Properties


AssocCls (instance of owl:ObjectProperty)

PROPERTY EDITOR

FOR PROPERTY: AssocCls (instance of owl:ObjectProperty)

Name | Equivalent Properties

AssocCls

RDFS:COMMENT:

ANNOTATIONS:

Property	Value	Lang

DOMAIN: Association

RANGE TYPE: Instance

RANGE: ObjectClass

[ ] Functional (only single values)

[✓] Inverse Functional

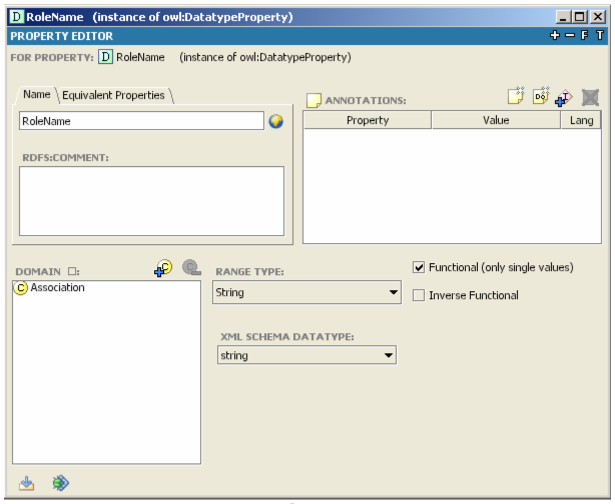
INVERSE: inverse_of_AssocCls

[ ] Symmetric

[ ] Transitive

Figure 6: An object property

There are three types of properties: Object properties, Datatype properties, and Annotation properties. (*i*) Object properties link one class or individual to another; (*ii*) Datatype properties link a class or an individual to an XML schema datatype value or an rdf literal; (*iii*) Annotation properties are used to add information to classes, individuals and object and datatype properties. Figure 6 is screenshot from Protégé depicted that the object property named *AssocCls* linking from class *Association* to class *ObjectClass*. Figure 7 shows that the property named *RoleName* is Datatype property linking from class *Association* to XML's datatype valued string.


RoleName (instance of owl:DatatypeProperty)

PROPERTY EDITOR

FOR PROPERTY: RoleName (instance of owl:DatatypeProperty)

Name: RoleName

RDFS:COMMENT:

ANNOTATIONS: Property | Value | Lang

DOMAIN: Association

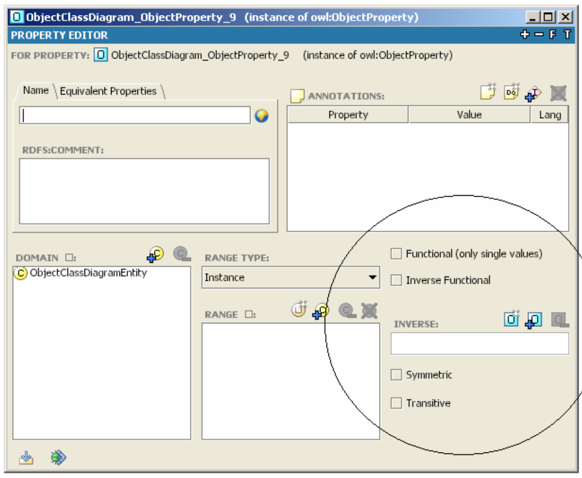
RANGE TYPE: String

[✓] Functional (only single values)

[ ] Inverse Functional

XML SCHEMA DATATYPE: string

Figure 7: A datatype property


ObjectClassDiagram_ObjectProperty_9 (instance of owl:ObjectProperty)

PROPERTY EDITOR

FOR PROPERTY: ObjectClassDiagram_ObjectProperty_9 (instance of owl:ObjectProperty)

Name:

RDFS:COMMENT:

ANNOTATIONS: Property | Value | Lang

DOMAIN: ObjectClassDiagramEntity

RANGE TYPE: Instance

RANGE:

[ ] Functional (only single values)

[ ] Inverse Functional

INVERSE:

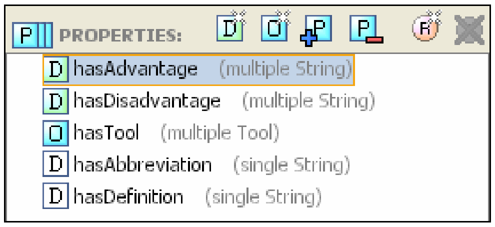
[ ] Symmetric

[ ] Transitive

Figure 8: Various characteristics set in Protégé

The meaning of properties is enriched through the use of property characteristic. The various characteristics that properties have are functional, inverse functional, transitive, and symmetric. In Protégé these characteristics can be set as shown in the circle in Figure 8.

6.2.1 Functional Properties


PROPERTIES:

hasAdvantage (multiple String)

hasDisadvantage (multiple String)

hasTool (multiple Tool)

hasAbbreviation (single String)

hasDefinition (single String)

Figure 9: Functional and non-functional properties

If a property is functional, there will be at most one individual that is linked to the individual through the property. Figure 7 shows that a property named *RoleName* is a functional property and meant only having a single string. Figure 9 is a screenshot from Protégé of our generic ontology showing *SoftwareEngineering* class properties. This says that *hasDefinition* and *hasAbbreviation* are functional property; and *hasAdvantage*, *hasDisadvantage*, and *hasTool* are non-functional property.

6.2.2 Inverse Properties

If property links individual *x* to individual *y* then its inverse property will link individual *y* to individual *x*. Figure 6 also shows the property *AssocCls* and its inverse property *inverse_of_AssocCls*. If the individual *Association* links with property *AssocCls* to the individual O*bjectClass* then because of the inverse property it can infer that the individual O*bjectClass* also links with property *inverse_of_AssocCls* to the individual *Association*. If a property is inverse functional then it implies that inverse property is functional also.

6.2.3 Transitive Properties

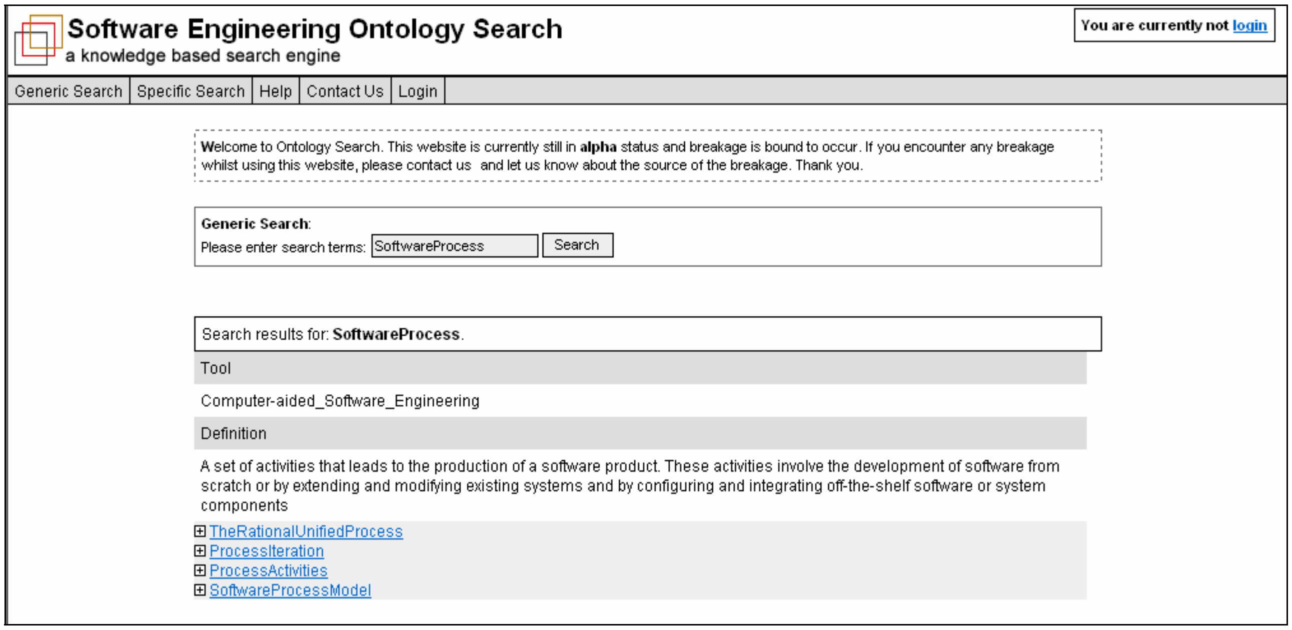
If property *x* is transitive and the property *x* relates individual *a* to individual *b* and also individual *b* to individual *c*, then it can be inferred that individual *a* is related to individual *c* via property *x*. For example, if extreme programming is an agile method and agile method is a rapid software development then it can be inferred that extreme programming is a rapid software development.

6.2.4 Symmetric Properties

If property *x* is symmetric and the property links individual *a* to individual *b* then individual *b* is also linked to individual *a* via property *x*. For example in '*use-case*' ontology as diagram shown in Figure 3 *hasLink* can be set as a symmetric property. If the individual *Actor* is linked to the individual *UseCase* through *hasLink* then it can be inferred that the individual *UseCase* must also be linked to *Actor* through the *hasLink* property. In other words, the property has its own inverse property.

7. Software Engineering Ontology Platform

In this section, we illustrate how software engineering ontologies facilitate communication and allow knowledge sharing. Figure 9 shows software engineering knowledge base allowing knowledge sharing. Any concept related to software engineering can be fetched showing the concept's details e.g. its definition, abbreviation, principles, advantage, disadvantage, output, template, tool, involved concept, etc. If the user clicks on relevant concepts which are arranged in hierarchy, user will see all details of the concept as well. This can be done by utilizing generic ontology and software agent to go through the ontology. Furthermore, from generic ontology software agent will be able to extract information e.g. from templates stored in the ontology as instances and create a handle book for the project. Figure 10 is a typical example of global communication which does not create consistent understanding. By utilizing application-specific ontologies (e.g. diagrams shown in Figure 2 and 3) and individuals/instances of a particular project data (e.g. diagram shown in Figure), it can convert the plain text


Software Engineering Ontology Search

a knowledge based search engine

You are currently not login

Generic Search | Specific Search | Help | Contact Us | Login

Welcome to Ontology Search. This website is currently still in **alpha** status and breakage is bound to occur. If you encounter any breakage whilst using this website, please contact us and let us know about the source of the breakage. Thank you.

Generic Search:
Please enter search terms: SoftwareProcess [Search]

Search results for: **SoftwareProcess**.

Tool

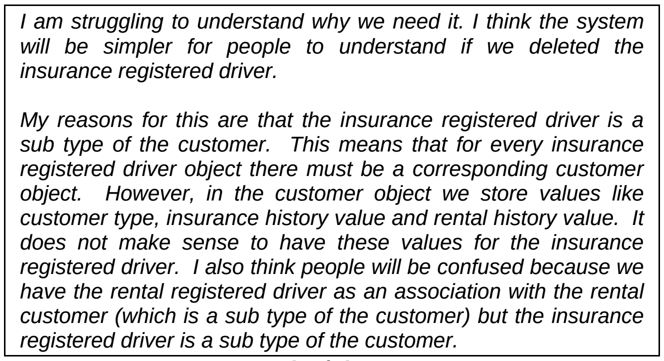
Computer-aided_Software_Engineering

Definition

A set of activities that leads to the production of a software product. These activities involve the development of software from scratch or by extending and modifying existing systems and by configuring and integrating off-the-shelf software or system components

⊞ TheRationalUnifiedProcess
⊞ ProcessIteration
⊞ ProcessActivities
⊞ SoftwareProcessModel

Figure 9: Screenshot of generic software engineering ontology search


I am struggling to understand why we need it. I think the system will be simpler for people to understand if we deleted the insurance registered driver.

My reasons for this are that the insurance registered driver is a sub type of the customer. This means that for every insurance registered driver object there must be a corresponding customer object. However, in the customer object we store values like customer type, insurance history value and rental history value. It does not make sense to have these values for the insurance registered driver. I also think people will be confused because we have the rental registered driver as an association with the rental customer (which is a sub type of the customer) but the insurance registered driver is a sub type of the customer.

Figure10: An example of plain text communication

into a UML-like diagram that helps communication among the team members within the same project and provides consistent understanding. Software agents can be utilized to extract information from ontology described in OWL. To do so, the software agent consults, for example, the 'object class' ontology. The ontology shows how class is formed in the class diagram; and each class contains a name, attributes, and operations; and relationships between the classes. Therefore, the software agent dynamically acts to retrieve involved class names, involved class attributes, involved class operations, and involved relationships to draw a class diagram.

References:

[1] I. Sommerville, *Software Engineering* (7th ed. Pearson Education Limited, 2004).

[2] P. Bourque, et al., The Guide to the Software Engineering Body of Knowledge, *IEEE Software*, *16*(6), 1999, 35-44.

[3] S. Cranefield, J. Pan, & M. Purvis. A UML ontology and derived content language for a travel booking scenario, *OAS'03 Ontologies in Agent Systems, 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia, 2003.

[4] D. Gašević, V. Devedžić, D. Djurić, MDA Standards for Ontology Development, *International Conference on Web Engineering (ICWE2004)*, Munich, Germany, 2004.

[5] S. Cranefield & M. Purvis, A UML profile and mapping for the generation of ontology-specific content languages, *Knowledge Engineering Review*, *17*(1), 2002, 21-39

[6] M.R. Genesereth, R.E. Fikes, et al., Knowledge Interchange Format Version 3 Reference Manual, Logic-92-1, Stanford University Logic Group, 1992.

[7] S. Luke & J. Heflin, *SHOE 1.01 Proposed specification,* SHOE Project, Feb. 2000.

[8] G. Librelotto, J.C. Ramalho, P.R. Henriques, *XML Topic Map Builder: Specification and Generation*. In: XATA: XML, Aplicaes e Tecnologias Associadas, 2003.

[9] R. Karp,V. Chaudhri, & J. Thomere, "XOL:An XML-Based Ontology Exchange Language, Aug. 1999.

[10] R. Kent, Conceptual Knowledge Markup Language, 1998.

[11] I. Horrocks et al., OIL in a Nutshell, *Proc.ECAI '00 Workshop on Application of Ontologies and PSMs*, Berlin, Germany, 2000.

[12] I. Horrocks & F. van Harmelen, Reference Description of the DAML+OIL Ontology Markup Language, draft report, 2001.

[13] McGuinness, D.L. & F.V. Harmelen, *OWL Web Ontology Language Overview*. 2004.

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

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