# Article 2

## Integration of Ontology and UML Class-Based Modelling for Knowledge Representation

Rozita Kadar
School of Computer Sciences,
Universiti Sains Malaysia, Pulau Pinang, Malaysia
Faculty of Computer and Mathematical Sciences,
Universiti Teknologi MARA Pulau Pinang Branch, Malaysia

Sharifah Mashita Syed-Mohamad, Putra Sumari
School of Computer Sciences,
UniversitiSains Malaysia, Pulau Pinang Branch, Malaysia

Nur 'Aini Abdul Rashid
Department of Computer Sciences,
College of Computer & Information Sciences,
Princess Nourahbint Abdulrahman University, KSA.

***Abstract***
*Program comprehension is an important process carried out involving much effort in software maintenance process. A key challenge to developers in program comprehension process is to comprehend a source code. Nowadays, software systems have grown in size causing increase in developers' tasks to explore and understand millions of lines of source code. Meanwhile, source code is a crucial resource for developers to become familiar with a software system since some system documentations are often unavailable or outdated. However, there are problems exist in understanding source codes, which are tricky with different programming styles, and insufficient comments. Although many researchers have discussed different strategies and techniques to overcome program comprehension problem, only a shallow knowledge is obtained about the challenges in trying to understand a software system through reading source code. Therefore, this study attempts to overcome the problems in source code comprehension by suggesting a suitable comprehension technique. The proposed technique is based on using ontology approach for knowledge representation. This approach is able to easily explain the concept and relationship of program domain. Thus, the proposed work will create a better way for improving program comprehension.*

***Keywords:*** *Program Comprehension, Knowledge Based, Information Extraction, Visualization, Ontology.*

### Introduction

Nowadays, software is developed iteratively and incrementally, which results in rapid evolution of software system. Besides, with today's rapid growth in system size and complexity, software maintainers are facing tremendous comprehension challenges driven by the need to maintain software system (Tiarks&Röhm, 2013; Carvalho, 2013; Yazdanshenas and Moonen, 2012). One of the key challenges faced by novice software maintainer is to comprehend the software system being maintained. Program comprehension is one of the major activities in software maintenance that mainly takes place prior maintaining process (Rajlich and Gosavi, 2004). Furthermore, software maintainers assigned with changing a large software system spend much effort on program comprehension to gain the knowledge on the system that needs to employ the changes (Corley et al., 2012).

Program comprehension can be very time-consuming due to the lack of proper documentation where some estimate that up to 50% of software maintenance effort is spent on understanding the software system maintained (Roongruangsuwan and Daengdej, 2010; SWEBOK, 2004; Guzzi et al., 2011; Xu, 2005), whereas 41.8% of the total effort is in reading and understanding program (Normantas and Vasilecas, 2013).

Hence, this work proposed the source code metadata extraction process and represented it in a form of ontology. The findings of this work are demonstrated as part of knowledge representation. This work focuses on Object-Oriented Programming. Meanwhile, the research questions are on how to design rules used to extract metadata from source code. The goal of this study is to facilitate novice developers to comprehend a program while performing maintenance tasks. This goal can be specifically achieved through the following objectives: to propose an extraction rules used to extract source code metadata. The expected contributions of this work are: the rules to extract source code metadata as the knowledge representation.

The paper is organised as follows. The next section reviews the previous studies by comparing the techniques in this area. The following section discusses the propose work. The conclusion of this study is presented in the final section.

**Related Work**

Ontology fragment is an approach used to improve program comprehension (Wilson, 2010). In populating ontology, the source code should be analysed through a process of information extraction, which is a process to obtain the information in a source code and display it in a different view. The retrieved information from the source code should be stored in a standard form of information. It will go through a process using an ontological approach. At present, the ontology is used in many fields to represent knowledge and to provide a formal way to define a concept. Use of ontology includes to support program understanding (Wilson, 2010). Bohnet et al. (2008) proposed a technique of retrieving source code information and then visualising various characteristics of the execution information to gain insight on how features are built on the code. Developers provide a scenario as an input that triggers a feature of interest in the system, whereas the output is presented to the developers using advanced visualisation views. This approach has been compared to a tool called grep with the result showing that developers are able to locate the concept of interest in less than half an hour without having prior knowledge about the system.

Similarly, Abebe & Tonella (2010) introduced an approach that extracts concepts from source codes by applying Natural Language Process (NLP) techniques where the identifiers of program elements are extracted and candidate sentences that use those identifiers are formed. Some of the sentences that do not follow certain rules were eliminated, while the remaining sentences were used as an input for creating ontology that captures the concepts and relations of the source code. A preliminary evaluation revealed that using information from ontology concept can improve the accuracy of query, allow developers to formulate queries that are more precise and can reduce the search space.

Petrenko et al. (2008) developed a feature location technique based on grep and ontology fragments. The ontology fragments stored partial domain knowledge about a feature. The hypothesis of this approach is that ontology fragments help developers to formulate queries and guide the investigation of their results, which would increase the effectiveness of the feature location.

The tool used to support the management of the ontology fragment is called Protégé. Wilson (2010) extended Petrenko et al. (2008) approach by introducing a systematic approach for formulating queries based on ontology fragments, which represents partial knowledge about the system. This approach has allowed the developers to formulate a query based on terms presented in the ontology fragment. A preliminary evaluation involving four developers to perform concept location on the Mozilla and Eclipse systems revealed that a small and partial knowledge about the system is sufficient for successfully locating a concept in the code. This approach is relevant to Petrenko et al. (2008) approach, but the main difference is that the former approach automatically generates the ontology, whereas the latter approach is manually generated by developers.

Meng et al. (2006) proposed a technique to program comprehension using ontology and description logic. As a part of the technique, they adopted a new interactive story metaphor to represent the interactions between users and comprehension process. The comprehension process can be viewed as authoring an interactive narrative between users and systems towards completing a specific goal. Developers have mapped between program comprehension process model and story model. The advantage of using story metaphor is that it provides an interactive context to guide comprehension process.

## The Integration Rules of Ontology and UML Class-based Modelling

In general, ontology development is divided into two main phases: specification and conceptualisation. The goal of specification phase is to acquire informal knowledge on the domain while the goal of conceptualisation phase is to organise and structure the obtained knowledge. The processes taken in developing the proposed program ontology that consists of three phases are illustrated in Fig. 1. Nonetheless, this article only discusses the first phase, which is the integration process of ontology and UML class-based modelling.

Ontology is used to define sets of concept describing the domain knowledge and allow for specifying classes by rich and precise logical definitions. The basic idea of developing ontology for software system is to provide an artefact consisting both code knowledge and domain knowledge, with which software maintainers can understand the features of source code. Ontology includes the concepts, relationships and instances to describe the **specific** domain of concern.

Concepts are referred to a category that is also known as a class. A series of concepts represents the topics or characters in the domain ontology; relations show the connection between concepts and used to describe the association between concepts when considering a specific concept, which is also called an attribute; while instances describe a series of concepts and relationships with specific knowledge. Instances in ontology are the values of attribute of the class that describe necessary properties. Instances will also inherit all attributes or relationships of their class.

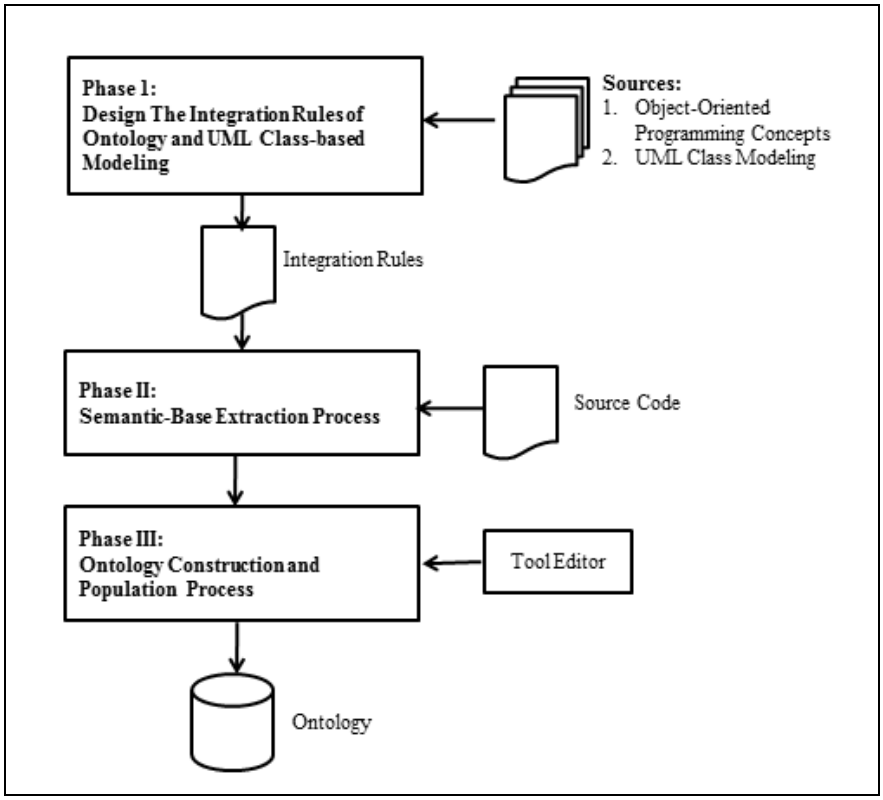


Figure 1: A Framework of Ontology Development

The proposed approach was used to extract an ontological point of view for software system by integration utilising the ontology and UML class-based modelling. Due to the similar features shared by both UML class diagram and ontology, class diagram was used to aid in populating code ontology (Table 1).

UML class has the following features that make it possible to be transformed in the form of ontology.

**Definition 1** A UML class diagram is a tuple, $C = \{C|\ C = (C_N, C_A, C_M, C_R)\}$ where $C_N$ represents the name of class in UML class diagram, $C, C_A$ is the list of attributes associated with this particular class, $C, C_M$ is a set of methods for defined class, $C$, whereas $C_R$ describes the different types of relation that can exist between any pair of classes in the UML class diagram, $C$.

**Definition 2** An attributes in UML class diagram is a tuple $C_A = \{A|A=(A_n, A_t, A_v, A_d)\}$ where $A_n$ is an attribute name, $A_t$ is an attribute type, $A_v$ is an attribute visibility (Public, Protect or Private) and $A_d$ is an attribute default value if given.

**Definition 3** A set of methods in UML class diagram is a tuple $C_M = \{M|M=(M_n, M_t, M_v)\}$ where $M_n$is the name of the method, $M_t$ is the method type and $M_v$is the visibility of the method.

**Definition 4** A relationship between classes in UML class diagram is a tuple, $C_R = \{R|R=(R_t, R_c, R_r)\}$ where $R_t$ is the type of relationship (Association, Composition, Aggregation or Generalization), $R_c$ is the cardinality specified for the source class and $R_r$ defines the target class with which the source class is connected.

A set of class diagram ontology described above are the complete structure of class diagram, which contains 4-tuples where the first concept is the class name, CN. The other element is concept of attribute, CA, where attribute name ɜ An, attribute type ɜ At, attribute visibility ɜ

Av and attribute default ɜ Ad. The concept method defines a set of methods in class diagram and describe as method name ɜ Mn, method type ɜ Mt and method visibility ɜ Mv.

The concepts relations described in the ontology were defined as relation type ɜ Rt, relation cardinality ɜ Rc and relation between classes ɜ Rc. All elements can be described as attribute type or method type = {numeric, string, NULL}, visibility of attribute and method = {public, private, protected}. Meanwhile, the relationship type = {aggregation, composition, association, generalisation}. The relationship cardinality = {0..*, 1..*, 0..1, 1} was used to describe the quantitative relationship between classes given by specifying minimum and maximum cardinalities.
Hence, a set of transformation rules was proposed to populate ontology from a UML class diagram. On the other hand, it is believed that class diagram also has semantics representation, which is somehow implicitly preserved. Thus, ontology could be used to recover the semantics for class diagram.

Table 1: Concepts Similarity of UML Class and Ontology Model

| Concept Similarity | Similarity Description |
|---|---|
| **CS1** | UML class denotes a set of objects with common features, while concept in ontology also does the same thing. |
| **CS2** | UML class has hierarchical structure, while hierarchical structure is basic structure for taxonomy, which is one of the features that ontology has. |
| **CS3** | UML class has properties, while ontology has two types of properties: object property and data type property |
| **CS4** | UML class has relations such as associations and dependencies, while these relations represented as roles or properties in ontology. |
| **CS5** | Class diagram includes class name, attributes and operation, while in ontology includes concepts, relationships and instances |
| **CS6** | Class itself will be transformed into concept in the ontology |
| **CS7** | The attributes of the class will be transformed into properties of that concept in ontology |
| **CS8** | For the generalization classes, the relationships SubClassOf will be preserved by the subclass concepts. |
| **CS9** | For the inheritance classes, the relationships SuperClassOf will be preserved by the superclass concepts. |
| **CS10** | Association transformed into ConnnectTo property, and it is a symmetric property. |
| **CS11** | Dependency transformed into DependOn property and its inverse property Depend. |
| **CS12** | Aggregation transformed into HasA property. |
| **CS13** | Composition transformed into PartOf property. |

**Conclusion**

Ontology has become popular in several fields of information technologies including software engineering. In software engineering, ontology is understood as a conceptual model representing a domain knowledge in a set of concepts within the domain, the properties of the concepts and interrelations of those concepts. It is acknowledged that ontologies are important sources of knowledge in the conceptualisation of certain domains as well as the background for software development. In future work, the proposed knowledge integration will be applied

as data source for information retrieval technique in concept location to find the relevance location in source code to implement change request. The aim is to facilitate the developers to find the location of source code prior implementing change request.

**References**

Abebe, S. L., & Tonella, P. (2010). Natural language parsing of program element names for concept extraction. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on* (pp. 156–159). IEEE.

Bohnet, J., Voigt, S., & Dollner, J. (2008). Locating and Understanding Features of Complex Software Systems by Synchronizing Time-, Collaboration- and Code-Focused Views on Execution Traces. *2008 16th IEEE International Conference on Program Comprehension*, 268–271. http://doi.org/10.1109/ICPC.2008.21

Meng, W., Rilling, J., Zhang, Y., Witte, R., Mudur, S., & Charland, P. (2006). A Context-Driven Software Comprehension Process Model. *2006 Second International IEEE Workshop on Software Evolvability (SE'06)*, 50–57. http://doi.org/10.1109/SOFTWARE-EVOLVABILITY.2006.1

Petrenko, M., Rajlich, V., & Vanciu, R. (2008). Partial domain comprehension in software evolution and maintenance. *IEEE International Conference on Program Comprehension*, 13–22. http://doi.org/10.1109/ICPC.2008.14

Wilson, L. A. (2010). Using ontology fragments in concept location. *IEEE International Conference on Software Maintenance, ICSM*. http://doi.org/10.1109/ICSM.2010.5609555