Formalizing Inheritance Detection of Class Diagram in UML

Alireza Souri [1], Mohammad esmaeel Akbari [2], Mohammad ali Sharifloo [1]

[1.] Department of Computer Engineering, Ahar Branch, Islamic Azad University, Ahar, Iran
[2] Department of Electrical Engineering, Ahar Branch, Islamic Azad University, Ahar, Iran
[3] Department of computer Engineering, University College of Nabi Akram, Tabriz, Iran
a-souri@iau-Ahar.ac.ir, m-Akbari@iau-Ahar.ac.ir, m.shariffloo@gmail.com

Abstract: One of the important relationships between classes in UML is inheritance relationships. Since there are few techniques for modeling and analyzing UML, by formalizing class diagram – as important section of UML – in this paper, after converting one system to class structure in UML, we present a solution for Establishment of inheritance relationships in UML. Also we can reduce complexity of each class in the system. Then by modeling this structure by this solution it will be prepare for some goals such as verifying and validating. By using an example we show procedure of formalizing the system. After all, we come to a conclusion that formal method improves ability of analyzing and implementing a system. By improving these factors, we can get better results in other techniques such as formal verification and validation and software testing.
[Alireza Souri, Mohammad esmaeel Akbari, Mohammad ali Sharifloo. **Formalizing Inheritance Detection of Class Diagram in UML.** *Academ Arena* 2018;10(6):28-31]. ISSN 1553-992X (print); ISSN 2158-771X (online). http://www.sciencepub.net/academia. 5. doi:10.7537/marsaaj100618.05.

Keywords: formal method, UML, class diagram, modeling, inheritance, relationships

1. Introduction

Modeling is discussed in many sciences. For example one part of designing software and hardware of systems can be modeled, verified and even evaluated by mathematic analyzes and logic commands [1].

UML is a modeling language used in software processes (e.g. Rational Unified Process (RUP)) to generate software models. Because of its simplicity, this language has been widely applied in the software industry. One of the disadvantages of UML models is that these artifacts can be interpreted differently by two members of a software team [4]. Papers [5], [6], [7], [8], [9] demonstrate that Alloy, which is the modeling language used in this paper, has been successfully used to formally specify UML diagrams.

A model is simplified the whole existence. Originally a model provides a map of a system. Models maybe include whole details of a system. So in general we can say that a good model is a model that can determine all the elements involved in the plan, relationships between them and how effective they can be. Each system explained by several models and there is a semantic map which describes the system in each model.

Now, by these fragments, why do we want to modeling? We want to model a system in order to have a good understanding of the system which we want to implement better. Modeling is central part of all activities that guide the software developers to good product.

In this paper, we explain class diagram and formalizing class structure briefly. Then by using an example, we present an algorithm in order to designing our system easily and exactly.

2. Class Diagram

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). The Unified Modeling Language (UML) is the standard formalism for object-oriented modeling [2]. In this paper, a class defines the methods and variables in an object, which is a abstraction of entity in a program or the unit of code representing that entity. Class diagrams are useful in all forms of object-oriented programming (OOP). The concept is several years old but has been refined as OOP modeling paradigms have evolved.

A more detailed class diagram can include the features of the entities as well as their responsibilities. There are two types of features: structural and dynamic [10]. Structural features can be subdivided in attributes and associations. Attributes correspond to variables in programming languages. Due to the fact that the associations between classes are represented as variables in programming languages, these are also considered to be structural features. The dynamic part of the classes are the operations, which are implemented by methods in a programming language. There are five types of relations between classes: association, aggregation, composition, generalization and dependency.

In a class diagram, the classes are arranged in groups that share common characteristics. A class diagram resembles a flowchart in which classes are

portrayed as boxes, each box having three rectangles inside. The top rectangle contains the name of the class; the middle rectangle contains the attributes of the class; the lower rectangle contains the methods, also called operations, of the class. Lines, which may have arrows at one or both ends, connect the boxes. These lines define the relationships, also called associations, between the classes [3].

Since class diagram has been developed widely, in this paper we choose class diagram in order to formalizing the systems, too.

3. Formalizing Class Structure

In our related work [1], we formalize class diagram because we can show all of the relationships of between classes in the system model easily. Now, by using a relationship in these methods [1], we want to present some solutions for class diagram.

These solutions cause:

- It influences on *normalization* design in creating database.
- Programmer finds main classes and it can program easily and exactly.
- Complexity of information has reduces in each class, so design of database, verification of system and implementation of system will be done better.

By formalizing a system using class diagram, we need to show all of the entities, specifications and behavior of the system as a class diagram.

Each system has some subsystems for converting a system to class diagram. We should create one main class and then convert subsystems to subclasses in main class.

For classifying one class, we should define some variables and values and we can show one system as followed set:

System: {Type, Classes, Attributes, Variables};

- Type: determines type of system. Type of system can be Basic, Real time, distribute, embedded, fuzzy and etc.
- Class: determines name of classes that have been classified and are connected together based on subsystems in the system.
- Attribute: determines attributes of system which are used for connecting to the classes in the system.
- Variables: determines Inputs, Outputs and the amount of Data of system which are placed as variables in the classes.

Now, when all states of the system converted to the class, we demonstrate all the relationships between elements of each class by each other.

Each class can have communications to its elements:

(Active, Module, finite) $\in$ C

- Boolean type:= active: shows activity of class
- Boolean type:= Module: shows the main class
- Boolean type:= finite: shows that the class is finite

Variables can communicate to their elements:

(Public, private, protected, package, initial) $\in$ V

Also, Attributes can communicate to their elements:

(Type, enable) $\in$ At

Ways of relationships between elements of system are:

C & At: V or At: C $\rightarrow$ V

Now, after explaining formula and variables, we create relationships between classes. We define three variables which are in structure of each class for creating relationships between classes:

{Extend, Attribute, Operation};

When two same attributes from two different classes give us a same result, we should place name of the second class in Extend section of the first class.

For example, in converting class relationships to the formulas like follow:

$\{c1, c2\} \in C$

$\{v1, v2\} \in V$

$\{at\_1, at\_2\} \in At$

at_1: c1 $\rightarrow$ v1

at_1: c2 $\rightarrow$ v1

We should place name of c2 class in Extend section of c1 class at the above formulas.

Whenever protected element enables in set of V, we should place the name of V in the Attribute section of resulted class.

For example:

$\{c1, c2, c_i\} \in C$

$\{v1, v2, v_i\} \in V$

$\{at\_1, at\_2\} \in At$

at_1: $c_i \rightarrow v_i \Rightarrow v_i \in$ (public, private, protected, package, initial) = (0, 0,1,0,0)

Therefore we should place v_i in Attribute section of c_i class.

Whenever Public element enables in set of V, we should place the name of V in the Operation section of resulted class.

For example:

$\{c1, c2, c_i\} \in C$

$\{v1, v2, v_i\} \in V$

$\{at\_1, at\_2\} \in At$

at_1: $c_i \rightarrow v_i \Rightarrow v_i \in$ (public, private, protected, package, initial) = (1, 0,1,0,0)

Therefore we should place v_i in Operation section of c_i class.

According to above explanations, we convert the relationships between classes as class diagram [1]:

The Roles of classes:

Extend: Ω

Inheritance

$c_i \in C$ { The name of all Classes }

$i \in C$ { The name of all Classes} – { c_i }

If $c_i \in i$. Ω

According to above formulas, whenever there was a class like c_i in Extend section of i class, i class inherits from c_i class.

In this relationship, we can define subclasses that create in property of each class:

Property: = {field name, type, data}

Each property can contain some fields. The information of each field includes information of subclasses. We purpose class i and class j that each class has a property. Property of each class includes some fields that these fields put content data in each class.

In this follow, we have two class_student and class_teacher:

For the Student:

Class_student_1 =: Property {(name of student, char, Alex), (family of student, char, Huffman), (ID number, int, 12345) };

Class_student_2 =: Property {(name of student, char, James), (family of student, char, Stanson), (ID number, int, 14145) };

Class_student_3 =: Property {(name of student, char, Kate), (family of student, char, Anderson), (ID number, int, 13345) };

For the teacher:

Class_teacher_1 =: Property {(name of teacher, char, Robert), (family of teacher, char, Macpean), (ID number, int, 6789) };

Class_teacher_2 =: Property {(name of teacher, char, Tomas), (family of teacher, char, Kameron), (ID number, int, 6779) };

Now, we want to find joint data in two classes by using function_Ω. First, we should implement search algorithm for finding common data in each field of class. See the figure-1.

The function_ Ω gets first field of Class_student_1 and compare first field of Class_teacher_1 and first field of Class_teacher_2. If there are Subscriptions between these fields,{field name, type, data} of same data put in the main class_parent. Also, this compare will execute for the other field.

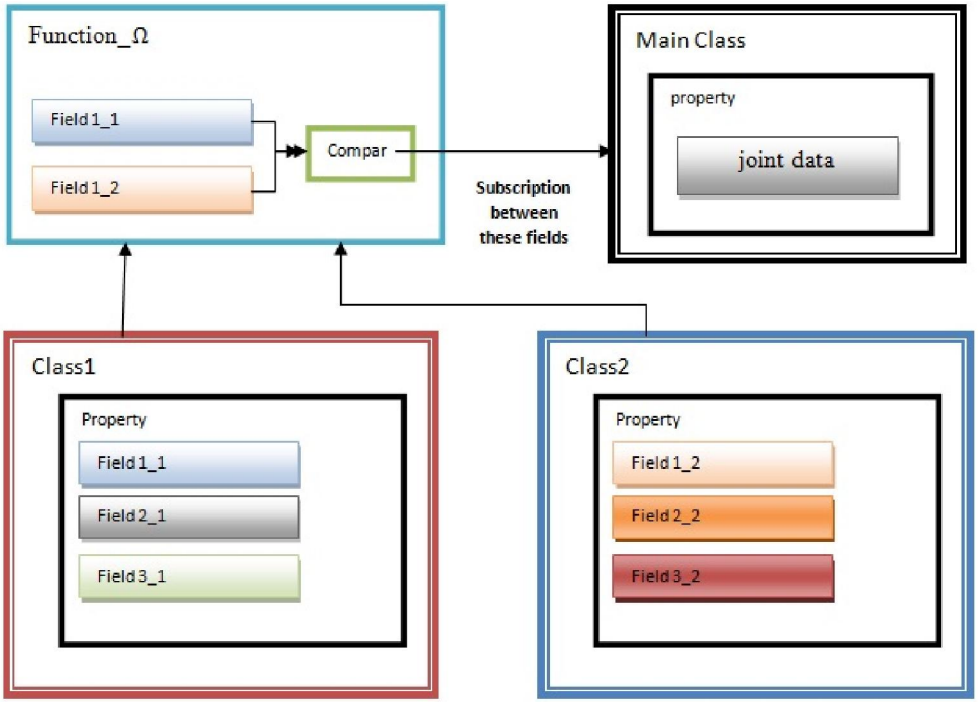

Fig.1: Class structure

Now, the function_ Ω gets second field of Class_student_1 and compare second field of Class_teacher_1 and second field of Class_teacher_2. If there are Subscriptions between these fields, field family, type, data} of same data put in the main class_parent. Then this compare will continue for class_student_2 and class_student_3. So, there are a main class_parent in the system model that this class maintenance fields such as name field and family field. The other properties will inheritance from these class properties.

After completing this main class_parent, the modeling of system by using the relationships will be easy. Understanding of inheritance relationships between classes will be better. Complexity of classes is reduced and system model is intelligible. By this model, we can implement our system easily and exactly. When subscription of classes is found, we have inheritance relationship between subclasses and main class. In normalization time, density of information and complexity of system are reduced. So, our database design will be better and softly.

4. Conclusion

By formalizing the structure of class in UML, we could model a system as a class diagram in order to determine all the relationships in the system clearly.

Also, we have shown the procedure of formalizing the system and by using a solution we can extend inheritance relationships between classes in system. This work has cause information inspection and search in system has been done. Also, when we want to model a big system, all of the relationships between classes and properties are modeled and we use these properties easily. Also, by using inheritance detection we can normalization database of system completely and easily.

We come to these results that formal method is one of the important and efficient methods for analyzing systems, by using this method we can model one system easily.

In the future we will try to analyze the other structures of relationships of UML and also we will attempt to present one Debugger algorithm for modeling UML.

Acknowledgements:

Foundation item: The National Project of India (No.: xxx-xxxx). Authors are grateful to the Department of Science and Technology, Government of India for financial support to carry out this work.

6/25/2018

Corresponding Author:
Dr. Geeta Kharkwal
Department of Botany
DSB Campus, Kumaun University
Nainital, Uttarakhand 263002, India
E-mail: geetakh@gmail.com

References

1. Alireza souri, Mohammad ali sharifloo and Monireh norouzi. " formalizing class diagram in UML". Proceeding of International conference on software engineering and service science 2011. pp 524-527.
2. Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. "Reasoning on UML Class Diagrams using Description Logic Based Systems". Proc. of the KI'2001 Workshop on Applications of Description Logics. Volume 44 of CEUR Electronic Workshop Proceedings,2001.
3. http://searchsoa.techtarget.com/definition/class-diagram.
4. Fernando Valles-Barajas, "Using Lightweight Formal Methods to Model Class and Object Diagrams", Com SIS Vol. 9, No. 1, January 2012.
5. Valles-Barajas, F. "A formal model for a requirements engineering tool". In: First Alloy Workshop, co-located with the 14*th* ACM/SIGSOFT Symposium on Foundations of Software Engineering (FSE'06). ACM, Portland, Oregon (2006).
6. Valles-Barajas, F.: A formal model for a state machine modeling tool: A lightweight approach. In: 3rd IEEE Systems and Software Week (SASW 2007). IEEE/NASA, Baltimore, Maryland (2007).
7. Valles-Barajas, F.: A metamodel for the requirements diagrams of Sys ML. IEEE Latin America Transactions 8(3), 259–268 (2009).
8. Valles-Barajas, F.: Use of a lightweight formal method to model the static aspects of state machines. Journal of Innovations in Systems and Software Engineering (ISSE): A NASA Journal 5(4), 255–264 (2009).
9. Valles-Barajas, F.: A precise specification for the modeling of collaboratios. Malaysian Journal of Computer Science 23(1), 18–36 (2010).
10. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, 3*rd* edn. (2003).