Processing Redundancy in UML Diagrams Based on Knowledge Graph

Yirui Jiang, Yucong Duan(✉), Mengxing Huang, Mingrui Chen, Jingbin Li, and Hui Zhou

Hainan University, Haikou, Hainan, China
duanyucong@hotmail.com

Abstract. UML (Unified Modeling Language) is designed for all stages of software development, but it lacks precise semantic information. MDE (Model-driven engineering) takes the model as the primary software product and its main research direction is modeling and model transformation. We propose to explore the entity abstraction scenario where no existing classes fit as the representative entity for other classes through the data, information and knowledge recreation of existing classes and relationships with the introduction of knowledge graphs. In this paper, we proposed a knowledge graph to enhance model design between models.

Keywords: UML · MDE · Redundancy · Knowledge graph · Diagram

1 Introduction

UML (Unified Modeling Language) is a general graphical modeling language for object-oriented development. UML provides a variety of graphical visualization to model stakeholders' intention into a series of models. These formed models comprise model elements. The same model elements can be displayed in a number of graphical models. People can view the modeled stakeholders' intention in the form of expressed models from multiple views. Knowledge map is closer to the natural language and is more expressive than UML models. We use Using knowledge graphs to deal with easier and can collect more information. To manually identify and resolve design inconsistencies are tedious and error prone [1]. UML is formal and expression is limited to the structure. There are some defects in the expression mechanism, We detect redundancy hidden in a diagram to achieve the purpose of accurate modeling and improve modeling quality and efficiency [2]. It is necessary for us to study the transformation between class diagrams and knowledge graph because knowledge graph can enhance the semantic expression of information.

In the paper, we firstly analyze the redundancy existing in modeling in Sect. 2. We elaborate the approach of construction of UML diagram in Sect. 3. In Sect. 4, we elaborate the approach about the transformation between diagram and knowledge graph. Then, we elaborate the method of integrating knowledge graphs in Sect. 5. After that, we analyze the experimental results in Sect. 6. Conclusion is in Sect. 7.

G. Chen et al. (Eds.): PAAP 2017, CCIS 729, pp. 418–426, 2017.
DOI: 10.1007/978-981-10-6442-5_39

2 Redundancy in UML Diagrams

UML includes some graphic elements that can be combined with each other. It laws that combine these elements. UML diagrams can be divided into five categories [3]. It includes UseCase diagrams, static diagrams, behavior diagrams, interactived diagrams and implementation diagrams. UseCase diagrams describe the system function from the user's point of view and point out the operator of each function. Static graphs include class diagrams, object graphs and package diagrams. In the UML model diagrams, class diagrams describe the static structure of the class in the system [4]. They not only define classes in the system, but also represent the relationship between classes, such as association, dependency, aggregation. Class diagrams describe a static relationship. Object diagrams are instances of class diagrams that are almost identical to class diagrams. The difference between them is that object diagrams show multiple instances of the classes. Package diagrams are used to describe hierarchical structure of the system. Behavior diagrams describe dynamic models of the system and interactions between components. State diagrams describe all possible states of classes and conditions for transition of the event. State diagrams are complements to class diagrams. Interaction diagrams describe interactions between objects. Sequence diagrams show dynamic cooperative relationship among objects in order to show interaction between objects.

2.1 Redundancy in Data Representation

When requirements are expressed from a different perspective, redundancy is likely to occur. A redundancy occurs when a design artifact (perhaps partial) is represented multiple times, possibly in varying views. Redundancies can occur either in design or data representation [5]. When two design units contain the same elements or information in an expression, it can produce design related redundancy. For instance, structural redundancy refers to the redundancy in a variety of structural and behavioral diagrams in modeling. This type of inconsistency can be viewed as a result of name space mismatch or structural conflict in the requirements specification. When two characteristics share an overlapping description, they will produce state conflict if they have the common usage.

A characteristic of a data representation is opposite in data objects. When R is regarded as the relationship between objects, the relationship has reversed two object positions of the reverse relationship. It can be seen as an inverse relation R^{-1} [6]. The definition of reverse correlation is very complex. So there is data redundancy. The redundancy in UML diagrams is the redundancy of data expression.

3 Transformation to Knowledge Graph

UML is a kind of general visual modeling language and used to describe sofewares, visualize processes and build documents of software system [7]. It supports a whole process of software development from requirements. UML can construct system models according to kinds of diagrams. There are static models and dynamic models in

diagrams. Static model diagrams are UseCase diagrams, class diagrams, object diagrams, component diagrams and deployment diagrams. Dynamic model diagrams are sequence diagrams, collaboration diagrams, state chart diagrams and activity diagrams. These diagrams can be used to visualize the system from different angles. UML is a set of graphical expressions used to describe OOAD process. It provides a comprehensive representation of requirements, behaviors and architectures of an object-oriented design [8]. The main function of UML is to help users describe and model the software system. It can describe a whole project that software has been implemented and tested from the requirements. A class diagram shows static views of the system, consisting of classes, relationships between them. Classes are represented by rectangles showing the name of the classes and the name of the operations and attributes. A UseCase diagram can describe user requirements and analyze the function of systems a user's point of view. In a UseCase diagram,an ellipse represents a use case and a human symbol represents a role.

3.1 Transformation from a Class Diagram to a Knowledge Graph

The quality of information transmission is low in a class diagram and the diagram may lose information. It may add invalid or wrong classes. Knowledge graph can make up the limitation of class diagram and enhance semantic expression. A class diagram can be mapped to a knowledge graph. A class diagram can be mapped to a knowledge graph. A class can be mapped to a concept in a knowledge graph. An attribute can be mapped to an entity, and the type of an attribute can be mapped to a concept. An operation can be mapped to a concept. A concept or an entity in a knowledge graph is presented as a node. We give rules for mapping classes and relationship as follows. There are rules for mapping classes.

Rule 1: the node of mapping by a ClassName is named C_ClassName.
Rule 2: the node of mapping by a AttributeName is named A_AttributeName.
Rule 3: the node of mapping by an AttibuteTypeName is named A_AttributeTypeName.
Rule 4: the node of mapping by a OperationName is named O_OperationName.

There are rules for mapping relationship.

Rule 1: the representative relationship of "Has a" means that the class ClassName has an attribute named AttributeName.
Rule 2: the representative relationship of "Has o" means that the class ClassName has an operation named OperationName.
Rule 3: the representative relationship of "Is a" means that attribute AttributeName has an attribute named AttributeTypeName.
Rule 4: Relationship types like generalization, aggregation, composition, dependency and association need to be described respectively.

3.2 Transformation from a UseCase Diagram to a Knowledge Graph

A UseCase diagram describes the function of the system from a user's view and abstracts the implementation of each function. A UseCase diagram can not express the requirements clearly. Many details can not describe clearly. We can use a knowledge graph to express completely. We elaborate rules for transforming a use case diagram to a knowledge graph as follows. They are rules for mapping actors.

Rule 1: the node of mapping by an ActorName is named A_ActorName.
Rule 2: the node of mapping by a UsecaseName is named U_UsecaseName.
Rule 3: the node of mapping by a UsecaseTypeName is named U_UsecaseTypeName.

There are rules for mapping relationships.

Rule 1: the representative relationship of "Operate" means that the actor operates a UseCase named UseCaseName.
Rule 2: relationship types like generalization, aggregation, composition, dependency and association need to be described respectively.

3.3 Transformation from a Knowledge Graph to a UML Diagram

A class diagram is a static view that describes classes in a system and relationships between classes. Class Diagrams help us to have a comprehensive understanding about the system before coding. In order to generate codes conveniently, it is necessary for us to transform a knowledge graph to a class diagram. If a knowledge graph satisfies the inverse of rules given previously, it can be transformed to a class diagram as well.

4 Deduplication in UML Diagram

4.1 Deduplication in Class Diagram

If there is no inheritance relationship between the two classes in a class diagram. But they have the same attributes or operations, there are redundant. A solution is to duplicate the attributes or operations. We can construct a new class. It makes a parent class have same attributes or operations, and inherit a parent class.

As shown in Fig. 1, there are two classes with redundant attributes in a class diagram. Librarian and reader have the same attributes like name, ID and Tel. The same attributes are redundant.

We map a class diagram in Fig. 1 to a knowledge diagram as shown in Fig. 2.

Each node can be traversed in a knowledge graph. We define the array Count[] represents traversal times of different nodes.

The initial value of Count[] equals to 0. If a node appears once, then the current value of Count[] = Count[] + 1.

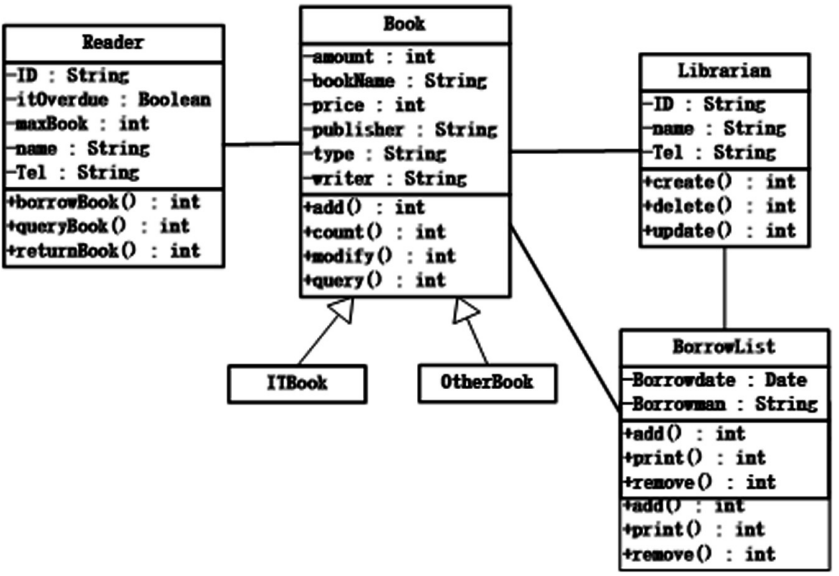

Fig. 1. A class diagram of a book management system

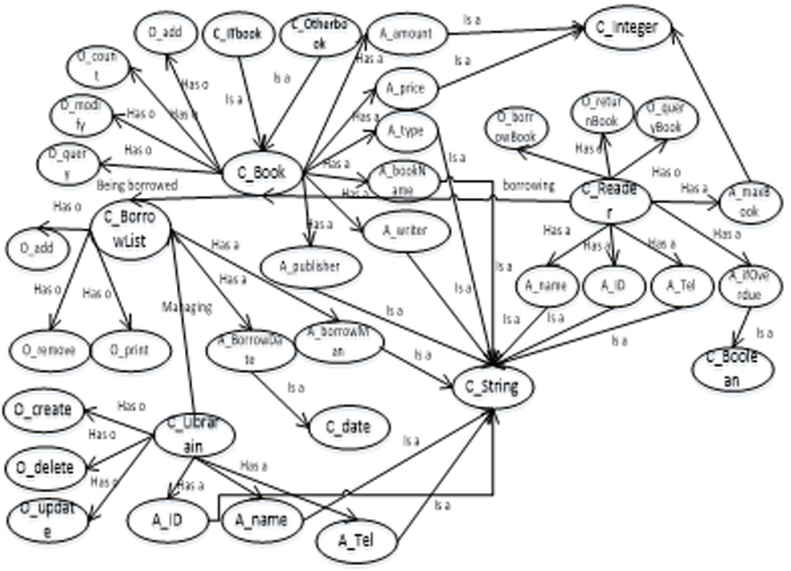

Fig. 2. A knowledge graph of a book management system

Algorithm Dealing with redundancy in knowledge graphs

Input: A knowledge graph array KG[ ]

Output: A refined knowledge graph

For : (each node KG{<KGn>,<R>} in array KG[ ])

Compute Count[];

If (Count[]≥*2*)

1. create a new node C_ClassName with the overlapping information
2. delete the same node A_AttributeName/O_OperationName
3.create<A_AttributeName,generalization,C_ClassName>and<O_OperationName,generalization, C_ClassName>

End If

End For

4.2 Deduplication in UseCase Diagram

In a UseCase diagram, if different actors have the same use cases, there will be redundant.

We give a UseCase diagram about a tutor website management system designed by a developer in Fig. 3. The actors are teachers, students and administrators.

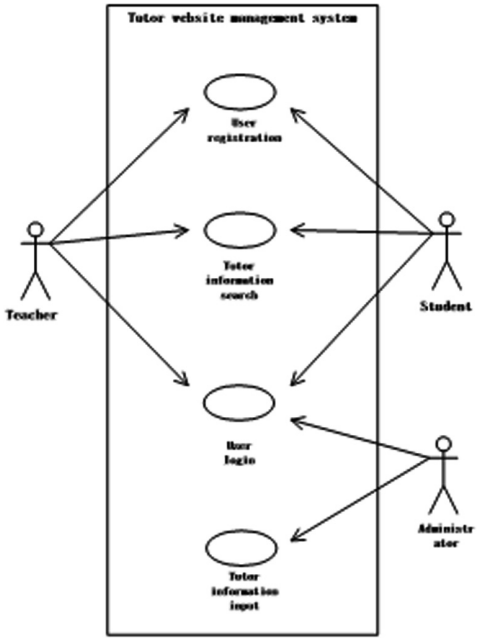

Fig. 3. A use case diagram of tutor website system

We map the use case diagram in Fig. 3 to a knowledge graph in Fig. 4.

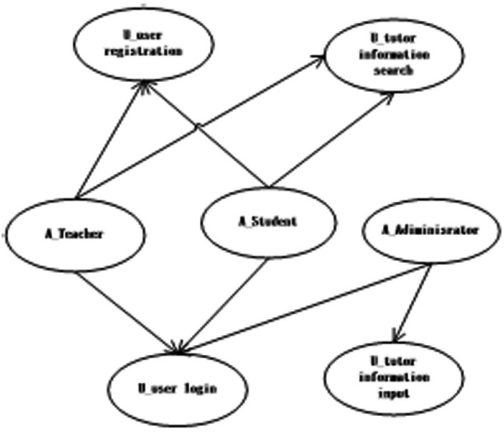

Fig. 4. A knowledge graph of tutor website system

Each node can be traversed in a knowledge graph. We define the array Count[] represents traversal times of different nodes.

The initial value of Count[] equals to 0. If a node appears once, then the current value of Count[] = Count[] + 1.

Algorithm Dealing with redundancy in knowledge graphs

Input: A knowledge graph array KG[ ]

Output: A refined knowledge graph

For : (each node KG{<KGn>,<R>} in array KG[ ])

Compute Count[];

If (Count[]≥2)

1. create a new node A_ActorName with the overlapping information
2. delete the same node U_UsecaseName
3.create<U_UsecaseName,generalization,A_ActorName>

End If

End For

5 Experimental Analysis

We measure an efficiency of identifying redundancy existing in diagrams through mapping them to knowledge graphs. We can eliminate redundancy between diagrams designed through a UML parser and examine redundancy between diagrams through a UML parser. Our work is comparing the time of deduplication between the use of diagrams and the transformation of knowledge graphs. Experimental tools can

eliminate redundancy according to rules we set before in seconds. We measured the median time of performing 1000 times. Our experiment uses for between 1 and 10 diagrams. The time spent in the two groups was measured. In our assessment, the time of eliminating redundancy in the transformation of knowledge graphs is shorter than the time of the use of diagrams (Fig. 5).

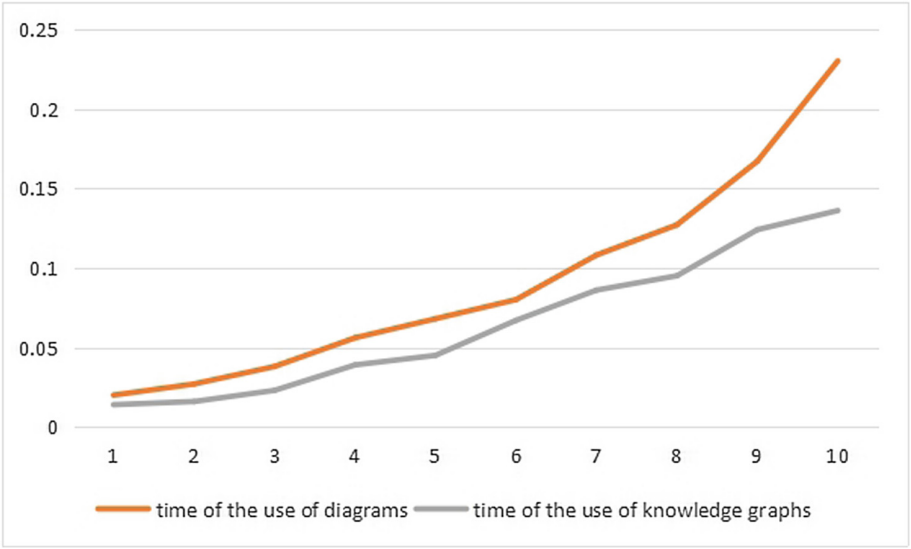

Fig. 5. Median time for eliminating redundancy

6 Related Work

Nowadays, there are two main methods to study the redundancy of UML: the first method is to use formal model. A specific approach is to make change for UML model and allow for easy detection of redundancy. OMG defines the syntax of the UML language through the use of model and the established OCL rules. The precise UML team has a prominent contribution for this. Its goal is to ensure the integrity of each element in the UML. The second method is to eliminate redundancy by establishing good limits. Some researchers have proposed an extension to OCL and claimed that OCL is not sufficient to express all kinds of limitations today. There are also some people who suggest using the rest methods like GCC (graphical consistency conditions) limited description language. The goal of the second approach is to improve the ability to limit the definition of language expression in order to formalize more semantic constraints. In the paper, we transform UML diagram to knowledge graph. Knowledge graph can enhance the expression of UML diagram and enhance the abstraction of UML diagram. This approach has a very good balance between the three languages used in the formal language, the formal language and the redundancy analysis.

7 Conclusion

Model-Driven Engineering (MDE) alleviates the cognitive complexity and effort through the refinement and abstraction of consecutive models [9]. MDE is close to human understanding and cognitive models, especially visual models. Designers can focus on business logics and related information. Changes to a model may introduce redundancies, which need to be detected and resolved. It is especially easy to produce redundancy when a product is designed from a different perspective. It is not only in design, but also in data representation. Language expression of UML is not complete and it is not effective [10]. Through a knowledge graph can enhance the expression of UML diagram and enhance the abstraction of a diagram. Compared with UML diagrams, knowledge graphs have abundant natural semantics, and their expression mechanisms are close to natural language and contain various and complete information. In this paper, we propose a method to transform a UML diagram into a knowledge graph. We detect redundancies hidden in a diagram to improve modeling quality and efficiency. Using knowledge graphs can eliminate redundancy in UML diagram.

Acknowledgement. This paper is supported by NSFC under Grant (Nos. 61363007, 61502294, 61662021, 61661019), NSF of Hainan Nos. ZDYF2017128 and 20156243, NSF of Shanghai No. 15ZR1415200, and STCSM project No. 14YF1404300.

References

1. Liu, W., Easterbrook, S., Mylopoulos, J.: Rule-based detection of inconsistency in UML models. In: Workshop on Consistency Problems in UML-based Software Development, pp. 106–123 (2002)
2. Boehm, B.W.: A spiral model of software development and enhancement. IEEE Comput. **21** (5), 61–72 (1998)
3. Duan, Y., Cheung, S.-C., Fu, X., Gu, Y.: A metamodel based model transformation approach. In: Third ACIS International Conference on Software Engineering Research, Management and Applications, pp. 184–191. IEEE (2005)
4. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst. Man Cybern. **11**(2), 109–125 (1981)
5. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artif. Intell. **168**, 70–118 (2005)
6. Van Der Straeten, R.: Inconsistency Management in Model Driven Engineering An Approach using Description Logics
7. Chein, M., Mugnier, M.L.: Graph Based Knowledge Representation. Springer, London (2009). doi:10.1007/978-1-84800-286-9
8. Spanoudakis, G., Finkelsteain, A., Till, D.: Overlaps in requirements engineering. Autom. Softw. Eng. **6**(2), 171–198 (1999)
9. Liu, J., Tong, G., Li, M., Zang, F.: Ontology-based semantics reasoning of UML class diagram. Comput. Appl. Softw. **28**(4) (2011)
10. Gogolla, M., Richters, M.: Expressing UML class diagrams properties with OCL. In: Clark, T., Warmer, J. (eds.) Object Modeling with the OCL. LNCS, vol. 2263, pp. 85–114. Springer, Heidelberg (2002). doi:10.1007/3-540-45669-4_6