Formalization of UML Composition in OCL

Hector M. Chavez, Department of Computer Science, Western Michigan University, Kalamazoo, MI, USA

Wuwei Shen, Department of Computer Science, Western Michigan University, Kalamazoo, MI, USA

ABSTRACT

With the increasing use of design models during the development process, developers now face a daunting task to maintain consistency between the design models and their implementation. Consistency maintenance is particularly challenging when a design model language introduces constructs that have no direct counterpart in programming languages. For example, the UML composition, an important relationship in software development, reflects some important principles in software engineering such as encapsulation. Unfortunately, influenced by the ownership model, existing approaches supporting UML composition require the non-accessibility property. However, composition in the UML specification does not support the ownership model. In this paper we present a UML composition formalization using the Object Constraint Language (OCL) that strictly adheres to the UML specification.

Keywords: Composition, Model Driven Architecture (MDA), Modeling, Object Constraint Language (OCL), Ownership, Unified Modeling Language (UML)

INTRODUCTION

The Model Driven Architecture (MDA) (Miller & Mukerji, 2003) provides an approach to system development where models are used to describe the different parts (architectures) of a system and their possible interactions. At the core of the Model Driven Architecture, the Unified Modeling Language (UML) (OMG, Unified Modeling Language: Superstructure, version 2.1.2, 2007), a general purpose modeling language, can be used to fully specify, construct, and document the artifacts of a system. Among the different types of diagrams, class diagrams are used to describe the static structure of a system and are among the most widely used in software development.

The increasing use of design models during the development process has introduced the challenge of maintaining consistency between the design models and their implementation. The inconsistencies introduced by developers that do not strictly follow the semantics of the design model can cause undesired behavior on a system, thus affecting the system’s quality and reliability as well as the traceability between the design and implementation. The consistency problem is further intensified by the fact that some of the abstractions introduced in the models have no
direct counterpart in the syntax and semantics for programming languages, such as composition, a subtype of the whole-part relationship in the Unified Modeling Language.

Composition is recognized as one of the most important relationships in software development because it reflects important principles in software engineering such as encapsulation. However, composition is ambiguously defined by the UML specification. This makes it a challenging task to check whether a program correctly implements UML composition, i.e., whether a program implements UML composition constraints, since the UML composition can be interpreted distinctly by different developers.

To solve this problem, existing approaches have defined the whole-part relationship in terms of properties such as the shareability and lifetime properties that distinguish subtypes of the whole-part, such as composition. The shareability property, which includes the local exclusivity property and global exclusivity property, requires that a part object be owned by at most one object at a given time. The lifetime property for an owner object demands that the owner object cannot be outlived by its part objects. In other words, whenever an owner object does not exist, neither do its part objects. The non-existence of an object means the non-accessibility to the object from live third-party objects.

Unfortunately, most existing approaches supporting composition are greatly influenced by the object ownership model (Clarke, Potter, & Noble, 1998; Boyapati, Liskov, & Shrira, 2003; Clarke & Drossopoulou, 2002), object ownership ensures that no reference to an owned object can be exported outside its owner object, i.e., only the owner object can directly access the owned object. We call this restriction the non-accessibility property. In this model, the owner object is the sole responsible for maintaining its owned objects' state and any information needed from the owned objects must be accessed through the owner object. As an example, Figure 1 shows two possible object ownership implementations, where an instance of class A acts as the owner of an instance of class B. The implementation in Figure 1 (a) satisfies the non-accessibility property: no object except for an owner object can directly access a part object. In this context, the lifetime and shareability properties are implied by the non-accessibility property. Thus, it is essential for these approaches to ensure that a reference to a part object is not exported by its owner object to a third-party object.

However, UML does not impose the non-accessibility property on a part object. The UML specification not only allows third-party objects to access an owned object directly but it also requires this behavior to correctly define basic class diagram elements such as the association between two classes. More details about this requirement will be introduced in the following section.

Therefore, in this paper we present a UML composition formalization that strictly adheres to the UML specification. This formalization paves the way for future work to check the consistency between a design model and its implementation in terms of the UML composition. The rest of the paper is organized as follows. The motivation section explains some of the challenges when supporting UML composition and why formalization is needed. We then present the OCL formalization of the UML composition followed by the related and future work.

**MOTIVATION**

In the object ownership model (Clarke, Potter, & Noble, 1998; Boyapati, Liskov, & Shrira, 2003; Clarke & Drossopoulou, 2002), object ownership ensures that no reference to an owned object can be exported outside its owner object, i.e., only the owner object can directly access the owned object. We call this restriction the non-accessibility property. In this model, the owner object is the sole responsible for maintaining its owned objects’ state and any information needed from the owned objects must be accessed through the owner object. As an example, Figure 1 shows two possible object ownership implementations, where an instance of class A acts as the owner of an instance of class B. The implementation in Figure 1 (a) satisfies the non-accessibility property: no object except for an owner object can directly access a part object. In this context, the lifetime and shareability properties are implied by the non-accessibility property. Thus, it is essential for these approaches to ensure that a reference to a part object is not exported by its owner object to a third-party object.
Process Evolution in a Distributed Process Execution Environment
www.igi-global.com/article/process-evolution-distributed-process-execution/80245?camid=4v1a

Increasing the Accuracy of Software Fault Prediction using Majority Ranking Fuzzy Clustering
www.igi-global.com/article/increasing-the-accuracy-of-software-fault-prediction-using-majority-ranking-fuzzy-clustering/120519?camid=4v1a