Formalisation of UML use cases

António Nestor Ribeiro

Departamento de Informática, Campus de Gualtar,
Universidade do Minho, 4710-057 Braga, Portugal.
e-mail: anr@di.uminho.pt

Abstract

UML has become a standard de facto for modeling object-oriented software systems. Among the several reasons for that standardization one can point that it is relatively easy to understand and learn, it addresses several views of software systems and it also provides a good overview of the entire software architecture. Despite its intrinsic value it still lacks some important aspects of software development that it does not address properly, namely the questions arising when modeling large and complex applications.

Although UML was meant to be a widely used modeling notation, it did not address from scratch the possibility of introducing formal methods or techniques in order to provide verification of the models. Some issues regarding the distance from the diagramatical UML notation to text-based formal notation were presented as possible justifications. However the lack of some kind of formal specification techniques is particularly sensitive when discussing the behavioural aspects of UML.

The advantages of having a formal specification notation to assure formalization of UML would be: to have precise rules for manipulating the components of UML diagrams, to allow for the possibility of proving properties about the components, and to make possible to check the model consistency. This is particularly true when modeling complex systems like multi-agent systems, distributed applications, concurrent and real-time environments, making the modeling and specification of such systems not as clear, unambiguous and precise as desirable.

Several initiatives have been taken to add support for those aspects enforcing the need for the introduction of formal support to the modeling tasks. However it is not enough to ensure a precise visual syntax, it is also needed to give efficient tools that allpw the reasonig about a model.

Some of the key features that formal languages provide can be obtained within UML models by adding Object Constraint Language (OCL) constraints. The overall quality of the model is enhanced with the usage of OCL expressions since they provide formal specification precision and the ability of reasoning on top of them.

Use cases play an important role in the overall analysis process since they are essential in capturing the requirements. However they lack a formal semantic definition, and practice shows that they can not fully express the complexity of underlying applications. In this work a way

to relate informal requirements, expressed as a set of use cases, to formal specifications using OCL is addressed.

Usually use cases description templates include pre and post-conditions, which are textual descriptions of the constraints, in order to establish diferent scenarios for a use case.

The formalisation of UML diagrams usually start at the Class Diagram level by introducing precise and formal constraints given by OCL expressions. We believe that the usage of the formal OCL pre and post-conditions of operations in the Class Diagram can be used to model the informal textual pre and post-conditions of use cases. It is our belief that a modeling task should begin with the capture of the user requirements at the use case level and that it is necessary to explicitely add contextual information to the use case description avoiding possible loss of information due to implicit assumed constraints.

In our work, state chart diagrams are used to complement use case diagrams in order to explicity decompose all possible alternative flows that were introduced by the usage of OCL constraints. The usage of state charts in order to help give formal reasoning to the use case description also provides the ability to introduce concurrency on the use case view as well as to describe the sequence of actions that are aggregated in a dialogue.

In the presentation we propose a reasoning model about the traces that can be generated from state charts, describing the flow of alternatives that can be derived from a use case decorated with OCL constraints for pre and post-conditions.