Validation of System Behavior Utilizing an Integrated Semantics of Use Case and Design Models

Duc-Hanh Dang

Department of Computer Science, University of Bremen
hanhdd@informatik.uni-bremen.de

Abstract. This paper summarizes an approach how to specify use cases and how to solve the problem of validating the conformance between the use case model and the design model. An integrated semantics of the two models is proposed. We employ UML- and OCL-based techniques as well as ideas from graph transformation. This research contributes to model transformation within the area of Model Driven Development (MDD).

1 Introduction

Specifying and validating the relationships between models are still challenges within MDD. A special case is the relationship between a use case model and a design model. The major difficulty in this relationship lies in the informality of use cases, and the loose relationship between the use case model and the design model. This paper proposes an approach to treat the problem. An integrated semantics of the use case model and the design model is established as a bridge between them. We use the precise semantics of the design model to complement the semantics of the use case model. The integrated semantics meets two aims: (i) to precisely specify the use cases, and (ii) to validate the conformance between the use case model and the design model.

A use case is a description of the system's behavior as the system responds to a request from actors, i.e., the types of users. Often, the description is represented as a loosely structured text or is visualized in a UML use case diagram. Many researches intend to make precise their use cases by specifying them with textual descriptions. Many proposals have been made to specify a hierarchical use case model [1]. However, the structure often lacks precise semantics in order to establish the relationship between the use case model and the design model. In other approaches, the behavior of use cases is specified using contracts, dynamic diagrams (e.g., statechart or activity diagrams). Some dynamic mechanisms of formal methods (e.g., B or ASM) are also used in this situation. However, these approaches also have difficulties to establish the relationship between the use case model and the design model in a precise way.

The relationship between the use case model and the design model is the relationship between a (functional) requirement model and a design model. In practice, e.g. Rational Unified Process, use cases are detailed in order to bring out the analysis model. Then the design model refines the analysis model. By doing so, the design model refines the use case model in an informal way. The problem is how we validate the conformance between the use case model and the design model. This problem is directly related to: (1) specifying use cases, (2) validating the design model, and (3) building refinements of models. An advanced approach to the problem (2) is that the design model is specified as an executable model. Then test cases can be used to validate the executable model. However, the approach does not consider the relationship between the use case model and the design model. For problem (3), many methods (e.g., B or Z) can be used to formalize the refinement. Some proposals have been made to formalize the refinement between the analysis model and the design model [2, 3]. Some researches consider the refinement of UML models [4] by using patterns. However, most of them solve the problem with class diagrams. Moreover, the refinement between the use case model and the design model are not considered.

In our approach, not only the textual descriptions of use cases but also the analysis of them are used to precisely specify them. We view the use case model not only from its textual description, but we do view it also from the design model. In our approach, a use case (i.e., a set of scenarios) is represented as the sequences of system snapshots. Therefore, the conformance between the use case model and the design model will be equivalent to the consistency between the snapshots at the two models. And we use UML and OCL as well as graph transformation [5] in order to validate the consistency between the snapshots.

Some challenges arise within the approach. The first one is the question how to use UML and OCL as well as graph transformation to specify the sequences of snapshots. The second one is the question how to show the consistency between the snapshots. Our approach to the problem will be discussed in this paper.

Our approach proposes a new way to validate the design model, and a new view of use cases allows us to precisely specify them. The approach opens the capability of proving the refinement between models as well as between transformation systems. The rest of the paper is structured as follows. Section 2 presents a description of the problem. Section 3 gives the details of our approach including tasks for the research. Section 4 summarizes our work.

2 Research Questions

This section discusses the problem how to validate the conformance between the use case model and the design model. The results from [5, 6] and our approach to use cases are integrated in order to get closer to a solution. The overall approach is presented in Fig. 1 and will be explained below.

First, in the part "System at the Use Case Level", we consider the requirements of use cases. Each scenario of a use case is a sequence of system snapshots. A system operation as proposed in [6] makes one snapshot change to another one. The sequence of the snapshots can be formalized using graph transformation. Each snapshot may be attached to use case conditions: We have the pre-

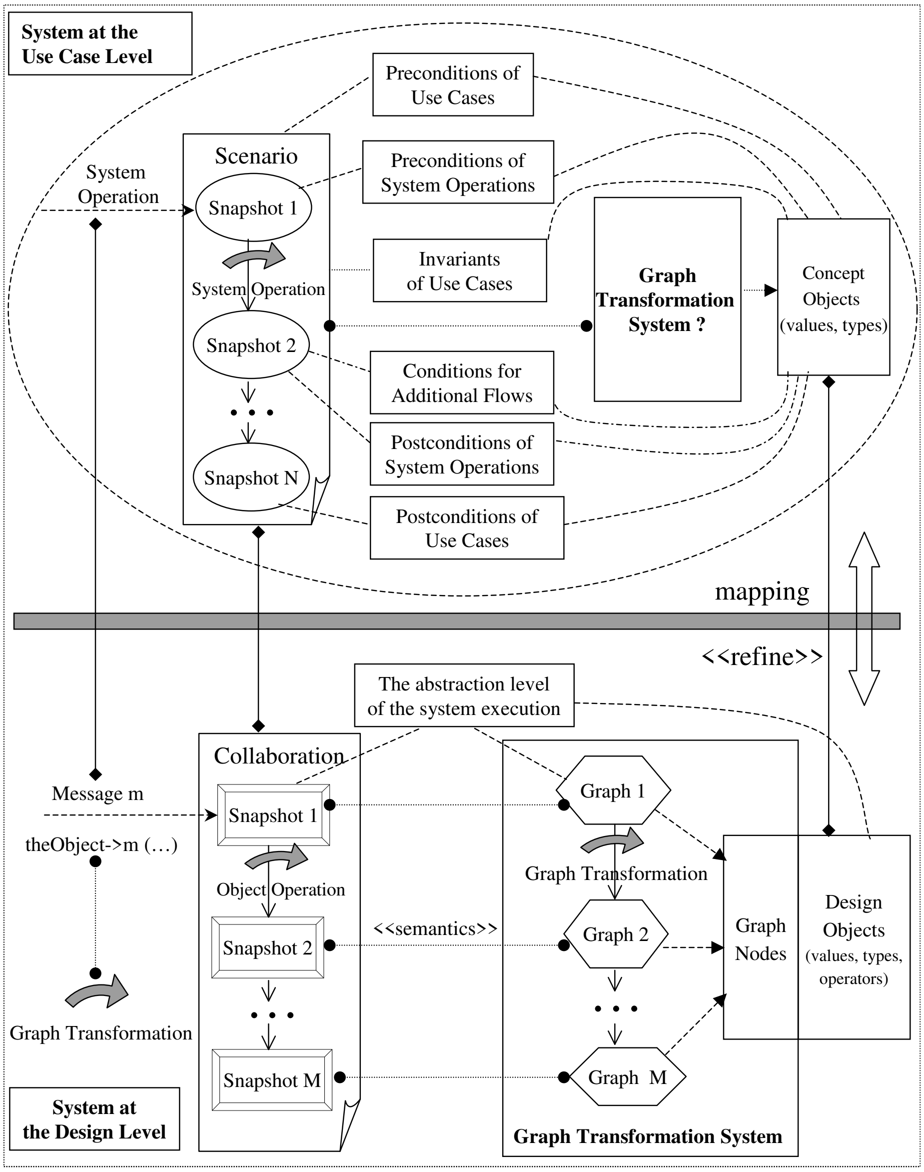

Fig. 1. The Execution of the System at the Use Case Level and the Design Level

and postconditions of use cases and system operations, the invariants (i.e. constraints on all snapshots), and the conditions indicating the additional flows. The concept objects are used to specify the conditions.

Second, in the part "System at the Design Level", we consider how the design model is executed. Each scenario of a use case is started by a message from an actor. The scenario is implemented as a collaboration of the actor and the design objects. The collaborations correspond to a sequence of snapshots. A graph transformation system is used to automatically execute the collaborations [5]. The results are snapshots (i.e., object diagrams).

At last, we check whether the results satisfy the requirements of the use cases in order to obtain the conformance between the two models.

A list of open questions follows:

– How do we obtain the correspondence between snapshots at the two levels?
– How do we map the constraints (on the snapshots) between the two levels?
– What is the relationship between the two graph transformation systems?

3 The Proposed Approach

This section presents our approach to the problem. It includes three tasks.
Task 1 - Specifying the sequence of snapshots at the use case level
Purpose: We represent use cases in order to map them to design models.
Approach: Transforming the analysis of use cases to a structure representing the sequences of snapshots. The role of system operations in the sequences is also specified. Each snapshot has to satisfy the conditions of the use cases. Using UML and OCL as well as graph transformation in order to specify the conditions of the use cases. Defining the scripts to present the structure in a validation and animation system.
Results: We have a structure to specify use cases.
Task 2 - Mapping between snapshots at the use case level and the design level
Purpose: Establishing the relationship between the snapshots at the two levels. Two snapshots reflecting the same system state must be consistent.
Approach: Building the mappings between the use case model and the design model: (i) between the concept objects and the design objects, (ii) between the system operations and the collaborations of the design objects, and (iii) between the conditions (on the snapshots) at the use case level and the design level. UML and OCL as well as graph transformations are used to specify the mappings.
Results:

- The mapping between the concept objects and the design objects and between the system operations and the collaborations of design objects.
- The relationship between the conditions at the two levels.
- The relationship between graph transformation rules at the two levels.

Task 3 - Validating the consistency between snapshots
Purpose: Obtaining the consistency between the snapshots at the two levels.
Approach: Building a mechanism to define snapshots at the design level corresponding to the considered snapshots at the use case level. The input of the mechanism is the sequences of the use case snapshots. The output is the information used to control the graph transformation system at the design level. By the execution, we obtain the snapshots. We use the results of the task 2 to transform the conditions of the use cases to the constraints on these snapshots. We check the constraints to obtain the results of the mentioned consistency.
Results:

- A mechanism to validate the consistency between snapshots at the two levels.
- A mechanism to translate the use case specification (in the task 1) to graph transformations at the design level.
- Validating the relationship between the graph transformation systems.

4 Summary

Due to the paper format, many references are not mentioned in this paper. At the time of writing, we have worked on the execution of design models and the specification of use cases with our USE system. We are ready to implement some results of the task 1 within USE. One part of the task 2 will be deployed soon. The task 3 will be started when the task 2 is finished. Due to the space limitations, a case study illustrating our approach is not presented in this paper.

Acknowledgments. I would like to thank Martin Gogolla, my supervisor, for many suggestions and support during this work. I also thank Fabian Büttner and Mirco Kuhlmann for the useful comments.

References

1. Whittle, J.: Specifying precise use cases with use case charts. In Bruel, J.M., ed.: MoDELS Satellite Events. Volume 3844 of Lecture Notes in Computer Science., Springer (2006) 290–301
2. Baar, T., Markovic, S., Fondement, F., Strohmeier, A.: Definition and correct refinement of operation specification. In Kohlas, J., Meyer, B., Schiper, A., eds.: Research Results of the DICS Program, Springer (2006)
3. Straeten, R.V.D., Jonckers, V., Mens, T.: A formal approach to model refactoring and model refinement. Software Systems Modeling (2006)
4. Pons, C.: Heuristics on the definition of uml refinement patterns. In Wiedermann, J., Tel, G., Pokorn'y, J., Bieliková, M., Stuller, J., eds.: SOFSEM. Volume 3831., Springer (2006) 461–470
5. Ziemann, P., Hölscher, K., Gogolla, M.: From uml models to graph transformation systems. Electr. Notes Theor. Comput. Sci. **127**(4) (2005) 17–33
6. Sendall, S., Strohmeier, A.: From use cases to system operation specifications. In Evans, A., Kent, S., Selic, B., eds.: UML. Volume 1939 of Lecture Notes in Computer Science., Springer (2000) 1–15