Abstract

In the past few years, model-driven software engineering (MDSE) and domain-specific modeling languages (DSMLs) have received a lot of attention from both research and industry. The main goal of MDSE is generating software from models that describe systems on a high level of abstraction. DSMLs are languages specifically designed to create such models. High-level models are refined into models on lower levels of abstraction by means of model transformations.

The ability to model systems on a high level of abstraction using graphical diagrams partially explains the popularity of the informal modeling language UML. However, even designing simple software systems using such graphical diagrams can lead to large models that are cumbersome to create. To deal with this problem, we investigated the integration of textual and graphical modeling languages by comparing two approaches, and designed a DSML with a concrete syntax consisting of both graphical and textual elements. The DSML, called the Simple Language of Communicating Objects (SLCO), is aimed at modeling the structure and behavior of concurrent, communicating objects, and is used as a case study throughout this thesis. During the design of this language, we also designed and implemented a number of transformations to various other modeling languages, leading to an iterative evolution of the DSML, which was influenced by the problem domain, the target platforms, model quality, and model transformation quality.

Traditionally, the state-space explosion problem in model checking is handled by applying abstractions and simplifications to the model that needs to be verified. As an alternative, we demonstrate a model-driven engineering approach that works the other way around using SLCO. Instead of making a concrete model more abstract, we refine abstract models by transformation to make them more concrete, aiming at the verification of models that are as close to the implementation as possible. The results show that it is possible to validate more concrete models when fine-grained transformations are applied instead of coarse-grained transformations.

Semantics are a crucial part of the definition of a language, and to verify the correctness of model transformations, the semantics of both the input and the output language must be formalized. For these reasons, we implemented an executable prototype of the semantics of SLCO that can be used to transform SLCO models to labeled transition systems (LTSs), allowing us to apply existing tools for visualization and verification of LTSs to SLCO models. For given input models, we can use the prototype in combination with these tools to show, for each transformation that refines SLCO models, that the input and output models exhibit the same observable behavior.

This, however, does not prove the correctness of these transformations in general. To prove this, we first formalized the semantics of SLCO in the form of Structural Operational Semantics (SOS), based on the aforementioned prototype. Then, equivalence relations between LTSs were defined based on each transformation, and finally, these relations were shown to be either strong bisimulations or branching bisimulations. In addition to this approach, we studied property preservation of model transformations without restricting ourselves to a fixed set of transformations. Our technique takes a property and a transformation, and checks whether the transformation preserves the property. If a property holds for the initial model, which is often small and easy to analyze, and the property is preserved, then the refined model does not need to be analyzed too.

Combining the MDSE techniques discussed in this thesis enables generating reliable and correct software by means of refining model transformations from concise, formal models specified on a high level of abstraction using DSMLs.