Abstract

We are surrounded by a large number of safety critical systems such as railway, cars and aircrafts. The incorrect behavior of such systems may have serious consequences, even to the extent of threatening human lives, so we need techniques supporting the design and development of correct systems. The application of *model-driven paradigms* is getting more and more important as the complexity of such systems have increased rapidly which could not be managed by traditional development methods. The main advantage of model-driven approaches is that not only do they document the components of the system, but implementation can be derived automatically using *code generation*. Several tools and languages are available supporting the design of systems with models. The internal behavior of reactive systems are usually represented by state-based models, starting from the component-level and using composition to build the system-level model. Unfortunately, many of the tools that support composition fail to define the precise semantics, making automatic code generation infeasible. Precise validation and formal verification of the design models are rarely supported for the same reason.

Proving correctness is an important requirement when designing safety critical systems. In addition to testing, *formal methods* can be applied to verify the correctness of the system design in an early phase. A common approach to state-based behavior analysis is *model checking*. Unfortunately, most of the modeling formalisms tailored for engineers are not suitable for direct analysis, therefore formal models usually have to be created manually by an expert team.

The goal of this work is to develop a framework that supports the design and analysis of *state-based behavioral* models. Based on an intermediate statechart language, a new language is defined to facilitate the *composition of statechart models* with precise semantics. The framework includes a code generator that produces the implementation of the composed system, assuming the implementation of the statechart models are given (as most tools support code generation for a single statechart) and following the semantics of the compositional language. To support the modeling process, validation rules have been defined for the intermediate statechart language to find design flaws as soon as possible. Furthermore, the automatic transformation of individual statecharts as well as their composition to formal models has been developed to support the formal analysis of the design models.

The framework currently builds on Yakindu, an open-source state-based modeling tool. Transformation from Yakindu statechart models to intermediate

formal models, as well as from intermediate formal models to UPPAAL formal automata is implemented by model transformations. The validation rules have been developed by using graph pattern matching languages and algorithms. One of the main advantages of the framework is that it is extensible with arbitrary state-based engineering and formal modeling languages, so it can be integrated with other design and analysis tools. The application and the merits of the framework are demonstrated in a project of the Fault Tolerant Research Group which includes the design and analysis of a distributed railway interlocking system.