Modeling and Validation: AUTOFOCUS and Quest

Oscar Slotosch*

Abstract

We describe the graphical description techniques, their underlaying semantic and the validation techniques that have been applied to the FM99 example; furthermore we sketch the model and discuss the validation results we have obtained. AUTOFOCUS is the modeling tool, the Quest environment connects the model to a broad spectrum of validation tools, including model checkers (SMV, SATO), theorem provers (VSE), and test tools (CTE). The original contribution, further publications, and AUTOFOCUS can be downloaded from `http://autofocus.in.tum.de`.

1 Modeling Technique

AUTOFOCUS offers a comprehensive set of graphical notations to specify distributed systems from different points of view. All description techniques are hierarchic and refer to a common system model. **System Structure Diagrams (SSDs)** describe the components of the system, their interfaces, and their connections. Components can be refined by assigning a substructure (again given as an SSD) to them; they can also refer to a behaviour description in the form of a state transition diagram (see below). Functional **Data Type Definitions (DTDs)** describe the types, values and functions the model is based on. Pattern matching can be used to define functions. The behaviour of components is described using **State Transition Diagrams (STDs)** that refer to the interface of components and operate on the data specified in the DTDs. Transitions consist of input patterns, preconditions, output patterns, and actions (to update local variables). Interactions between components of the system are described using **Extended Event Traces (EETs)**. EETs show interaction sequences of components. EETs are used to capture requirements, to document simulation runs, to represent test cases, and to visualize counter-examples obtained from model checking.

The formal semantic basis of the descriptions is a mathematical model consisting of time-synchronously operating components with a single element buffers for communication. Thus neither micro steps, nor instantaneous feedback are needed. This simple semantics is well suited for the verification task and forms a clear basis for the software engineering process.

*We thank TU Munich, and German BSI for financial support, and the whole development team, especially Peter Braun, Franz Huber, Heiko Lötzbeyer, and Bernhard Schätz. For comments on this paper we thank Ingolf Krüger, Robert Sandner, and Bernhard Schätz.

2 Model of the Banking System

Our model for the banking system comprises two tills that are connected to a central banking system via insecure communication channels. The central system is modeled with two drivers for the channels and a database, consisting of two parallel processes to read data from the clients' accounts. The DTDs contain descriptions of the data structure for credit cards. Cards store the pin code, the account number, the date of the last withdrawal, and the amount that has been withdrawn at this day. Furthermore, types for the interactions between users and tills, and messages to describe transactions are defined. The tills gather the required information, and send it in the form of a composite transaction, to the central. If the connections are down sending is repeated until an acknowledge signal is received. The decision whether the client can withdraw money or not is modelled nondeterministically and can be refined later. The STDs of the till, the connections, and the processes are reused for both tills, connections, and processes, to show that the model can easily be extended with additional tills.

3 Validation Techniques

Several validation techniques can be applied to the model. **Consistency checks** allow us to express static consistency of the different views. This enables detection of modeling errors like type errors, or inconsistencies between different abstraction levels and different views. The graphical **simulation** allows us to validate the systems behaviour. Consistency checks and simulation are the main features to improve the adequacy of the (formal) model. A simple temporal logic is used to describe safety critical system properties. These properties can be **model checked** using SMV and **bounded model checked** using SATO. For larger or infinite systems **abstraction techniques** can be applied to reduce complexity. For each property a proof obligation is generated ensuring that the property is true in the concrete system, provided it holds in the abstract system. These proof obligations and other properties can be verified using the **interactive theorem prover** VSE (Verification Support Environment). Several **specification based test case generation** methods can be applied to test implementations of the system or a hardware realization. Some of those methods are: classification of variables using the classification tree editor CTE, a transition tour that covers all transitions of an STD, and systematic approaches to testing the combination of several units.

4 Validation Results

We corrected several formalization and modelling errors that have been found using consistency checks and simulation. We model checked the property that the driver drives the connection correctly. Applying bounded model checking, we proved that invalid cards are rejected. The complete model was checked abstracting from the data types. Counter-examples to negated properties were used as test cases, for example to show how money can be withdrawn.