A Steam-Boiler Control Specification with Statecharts and Z

Robert Büssow, Matthias Weber

Technische Universität Berlin
Forschungsgruppe Softwaretechnik (FR5-6)
Franklinstr. 28/29, D-10587 Berlin, Germany.
email: {buessow,we}@cs.tu-berlin.de

May 1996

A widely used technique in modern software engineering is to model a system by a combination of different – but semantically compatible – "views" of that system. The primary benefit of such an approach is to keep very complex systems manageable and to detect misconceptions or inconsistencies at an early stage. In the approach presented here, we divide the modeling into three views: the architectural model of the system, the reactive model of the system, and the functional model of the system.

This paper presents a specification of the steam boiler control system following this threefold approach. The architectural model decomposes the control-program into different objects, e.g. the *SteamSensor* or the *WaterSensor*. The reactive model describes how the program interacts with its environment while the functional model specifies its data transformations, e.g. the calculations necessary to estimate the water level. A systematic relationship between the reactive and the functional view entails proof obligations to guarantee semantic compatibility.

The *architectural model* of a system describes the relationships between the types of components used in the system as well as the actual configuration of the system components itself. For the description of this model, we adopt the object-oriented modeling paradigm [1, for instance]: We understand an embedded control system as a hierarchically structured collection of objects that change state and interact with each other throughout their lifetime. The relationships between object classes are described using well-known elements of class diagrams, i.e. diagrams displaying classes and their structural relationships, such as aggregation and inheritance.

The two other views are primarily concerned with the specification of the behavior of single components of the embedded control system. We make a

fundamental distinction with respect to the behavior of system components. The *functional model* of a component comprises data definitions, data invariants, and data transformation relations, in particular, for any component, its encompasses its local state and the input/output relation of its operations. Constraints, e.g. related to safety properties, about the components states can be derived based on these descriptions. The *reactive model* comprises the life-cycle of components, i.e. interactions with other components and the control of time during these interactions. Reactive behavior is modeled by specifying how, and under which timing constraints, operations from external objects are requested or supplied (or both) in the state changes of objects.

We specify reactive behavior using an appropriate variant of timed hierarchical state transition diagrams, i.e. with a variant of statecharts [2]. There are two reasons for this choice: firstly, statecharts have proven to be sufficiently expressive for modeling complex component interactions and time control, and secondly, the use of statecharts, or close variants of statecharts, is currently spreading in industry. This also enables us to use existing analysis and simulation tools for this notation.

Often, functional behavior in state-based systems is specified by textual or formal descriptions of pre- and postconditions and of data invariants. In our approach, we specify the functional behavior of objects using the state-based formal specification language Z [3]. There are two main reasons for using Z: firstly, in our view, Z has proven to be particularly useful for modeling complex functional data transformations; and secondly, both in academia and industry, Z has become one of the most widely used formal specification notations. Since we aim at a practical approach when modeling functionality, we try to stick to a constructive subset of Z, i.e. a subset that can be compiled into efficient code, whenever this is reasonable in a particular application. The use of a mathematical notation for modeling functional behavior enables us to prove abstract safety properties about the control system, such as provisions that the system may never enter certain hazardous states. Safety conditions imposed on data structures and data relationships should, of course, be specified using the full expressive power of the Z language.

The reactive and functional views of an embedded system can be checked against each other in many interesting ways: The basic idea is to systematically and consistently relate the state hierarchy and the transitions introduced in the statecharts with the state spaces and operations as defined by the Z schemas.

We would like to stress that we are *not* arguing in favor of a monolithically formal approach. Rather, our goal is to systematically embed mathematical elements into industrially used engineering techniques. As a consequence, some parts of our approach are "hard", i.e. fully precise, while others remain "softer", i.e. allow for a certain range of interpretations. In our view, such an approach may still serve to prove interesting safety properties while leaving more flexibility to be adapted to the actual circumstances in particular industrial application contexts.

References

[1] G. Booch. *Object-Oriented Analysis and Design with Applications.* Benjamin Cummings, 1994.

[2] D. Harel. Statecharts: A visual formalism for complex systems. *SCP*, 8(3):231-274, 1987.

[3] M. Spivey. *The Z Notation, A Reference Manual.* Prentice Hall, 1992.