Assertional Specification and Verification using PVS of the Steam Boiler Control System

Jan Vitt[1] and Jozef Hooman[2]

[1] Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Preusserstraße 1-9, 24105 Kiel, Germany
e-mail: jav@informatik.uni-kiel.d400.de
[2] Dept. of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
e-mail: wsinjh@win.tue.nl

Extended Abstract

An assertional method is used to derive a correct implementation of a steam boiler control system. To specify and verify (real-time) properties of the control system we use a compositional framework. During top-down program construction each design step is verified by means of compositional proof rules, reasoning with the specifications of components without knowing their implementations. To represent the intermediate stages of top-down design, we use a mixed formalism in which one can freely combine assertional specifications and constructs of the programming language.

The application of a formal method in the design process leads to a large number of verification conditions, including many trivial proof obligations. Therefore the tool PVS (Prototype Verification System) has been used to support the design. Our mixed framework has been formulated in terms of the PVS specification language, a strongly-typed higher-order logic. This enables us to verify the correctness of each refinement step by means of the interactive proof checker of PVS.

Our design of the control system comprises three stages where the design in a later stage is based on the ones in earlier stages. In the first stage, we consider only real-time aspects of the steam boiler control system and assume that neither communication channels nor physical components fail. Next, in the second stage, failures of communication channels are allowed and a control component that is able to detect such failures is derived. It is an extension of the real-time controller derived before. Finally, in the last stage, also failures of physical components are taken into account.

In the first stage, first a formal description of the top-level specification of the complete system, i.e., control system plus continuous processes (such as inflow of water and outflow of steam) is formulated. Further, relevant properties of the physical processes to be controlled are formalized. Then we can describe a control strategy in terms of a continuous interface and prove that it leads to the top-level specification. To implement the control strategy we introduce a water sensor and a pump system which transform the continuous interface into discrete events. This leads to a specification of a discrete control program. The correctness of these design steps is proved by means of PVS, using only the specifications of the components in terms of their real-time communication interface. Finally, an implementation of the discrete controller is derived, using a simple real-time programming language.

The next stages, where failures of channels or components are allowed, we first consider failing channels that might loose messages. A communication axiom is modified to represent the assumptions about the behavior of incorrect channels. Then the design of the previous stage is adapted, making assumptions about the correctness of channels explicit. Further we show that a detectable failure of a channel leads to an alarm message. A failing component is treated similarly, formulating a failure hypothesis about the effect of faults and making all assumptions about the correctness of components explicit. A steam sensor is introduced to deal with a failing water level sensor.

Evaluation and Comparison

1. Which parts of the system are specified at which level and in which way? (verbally, rigorously, formally?)

The system configuration is somewhat reduced from the original problem description; the operator desk is not modeled and only a single pump and corresponding pump controller are considered. Moreover, only normal mode is fully dealt with. In particular, initialization mode is not treated; proper initialization is assumed. The aspects that are included are completely formal, that is, there is a formal specification of the components (control program, water level sensor, steam sensor, pump actuator, pumps) and a formal derivation of the control program.

2.a. Does the solution comprise an implementation of a control program? b. Has the control program been linked to the FZI simulator? c. Has some experimentation been done with the control program?

The resulting control program has been written in a pseudo programming language, modeled in the PVS language. It has not been linked to the FZI simulator and no experimentation has been done.

3. What are the comparable other solutions and what are the major differences w.r.t. them? What other solutions complement the given solution and in which respect?

Close to our approach is the solution given by [RS], since they also start from a formal specification (expressed in the Duration Calculus) and then derive an implementation (in Occam) by means of a number of design steps. Similarly, [CW1] is strongly related, where additionally the use of a theorem prover is relevant. An alternative approach to program refinement is given in [BSS] using action systems. Comparable are further solutions that use some form of logic to express properties of the system, such as the temporal logic TRIO [GM], TLA [LM], or VDM [LP].

Complementary to our solution are papers that put more emphasis on requirements engineering. For instance, [BCPR] contains an analysis of the informal requirements. Interesting is also that validation has been supported by the Larch Prover to prove properties from the specification. This is closely related to the use of "putative" theorems as proposed by the designers of PVS. Also complementary are approaches that concentrate more on simulation and the design of an executable program. For instance, given a formal specification, [AT] considers deductive synthesis of programs. Finally, we mention the work done by [LW] which complementary since they focus on performance and optimization.

4. a. How much time has been spent on producing the solution? Give the number of person months, if possible for the various parts of the solution.

The work described here has been done as the master's thesis work of the first author and took about nine months. The first part, in which no faults are considered, was rather straightforward since we could build on previous examples of real-time control systems. Only the complete correctness proof of the final control program, including timing aspects, was rather intricate and time consuming. Later stages of the design, considering failures, took more time due to limited experience in this field. Therefore, a considerable amount of time has been spent on possible methods to formalize failures, their effect, and their detection.

5. What are the premises for a good understanding of the proposed solution? a. Is a detailed knowledge of the used formalism needed? b. Can an average programmer understand the solution? c. How much time do you believe is necessary for the average programmer without knowledge of the used specification method to learn what is needed to be able to understand the solution?

According to our experience, it requires some basic knowledge of first-order logic (the higher-order aspects of PVS seem easy understandable) and some basic programming knowledge (e.g. about types, records, enumerations, etc.) to understand our PVS specifications. Given this background, usually a few days of introduction and hands-on experience is sufficient to start working with the PVS system. The basic principles of compositional method for distributed (real-time) systems can be learned in a course of fifteen hours.

Acknowledgment

An anonymous reviewer is thanked for useful suggestions and a valuable contribution to the evaluation described in this section.

This article was processed using the LaTeX macro package with LLNCS style