Refinement from a control problem to programs*

Michael Schenke[1] and Anders P. Ravn[2]

[1] FB Informatik, Universität Oldenburg
Postfach 2503, D-26111 Oldenburg, Germany
E-mail: schenke@arbi.informatik.uni-oldenburg.de
[2] Department of Information Tecnology
Technical University of Denmark, bldg. 344
DK-2800 Lyngby, Denmark

1 Extended Abstract

The distinguishing feature of the presented refinement approach is that it links formalisms from a top level requirements notation down to programs together in a mathematically coherent development trajectory. The approach uses Duration Calculus, a real-time interval logic, to specify requirements for a control task, exemplified by a steam boiler. The same formalism is used to refine requirements to a functional design. Through a suitable transformation this is taken to an event and action based formalism. Finally components in this design for a distributed architecture are transformed to occam-like programs.

The Steam-Boiler control problem is a good example of a major application area for real-time computing systems. The overall task is to control a mechanical, chemical etc. plant or device under a variety of operating modes which are caused by changing operating conditions, for instance caused by faulty equipment. Such plants are often safety critical and must therefore be developed using the highest standards of engineering. In particular, programs must be the result of a rationally documented design process.

The approach to the Steam-Boiler control problem as demonstrated in this paper has been developed by the ESPRIT BRA "Provably Correct Systems" (ProCoS). The goal of the ProCoS project was to create a mathematical basis for the development of embedded real-time computer systems. A starting point in the development of such a system is to use a precise temporal logic, the Duration Calculus (DC) as requirements language. The requirements are then systematically transformed to occam-like programs. This part has been applied to the steam-boiler. ProCoS has furthermore developed correct compilation methods for such programs to conventional processors and to hardware. An overview of the ProCoS project is found in [1].

Scope. We want to demonstrate a deep development path with a reasonable level of detail. In order to do so without burdening the reader with repetitive

* This work is partially funded by the Commission of the European Communities (CEC) under the ESPRIT programme in the field of Basic Research Project No. 7071: "PROCOS II: Provably Correct Systems".

constructions, we elide duplicated developments, e.g. we consider only the active phase of the steam boiler, not the initialization, and we do not replicate fault handling components for every sensor or actuator. These do not contribute anything to the understanding of our method. The facets of the steam boiler that are covered or deliberately omitted are:

1. A formal specification of safety and liveness requirements to the overall, hybrid system consisting of continuous plant and the discrete control modes is presented in Section 2[3]. Performance is not considered in the control law that is shown to refine the requirements.
2. A functional design with specifications for sensors, actuators, and a control program is specified. The design concentrates on the operating mode and the fault detection and tolerance mechanisms. It is presented in Section 3. We omit initialization, human operator interaction and multiple pumps. Fault handlers are illustrated by the most interesting one for the pump. For a version where other facets are included, see [2]. It is outlined, how it is formally shown that the design refines the requirements.
3. Section 4 contains a design for a distributed architecture with components that are specified in the Systems Language (SL) as communicating sequential processes using a network of point-to-point channels. The components include: main control, and devices with associated fault handlers. The component specifications are derived by syntax directed transformation rules from the global design.
4. In Section 5 occam-like real-time programs for the components are derived by another syntax directed transformation from the architectural SL-design. The programs have not been compiled and linked to a simulator. This has been demonstrated in further work on correct compilation [1].

2 Conclusion

We have demonstrated a method for the development of proven correct occam programs. As far as we know the present solution is the only one which uses a real time refinement calculus and not ad hoc transformations. In so far it is difficult to compare the presented solution with other ones. The solution which is the closest to ours is the one from [2].

In our solution we have not presented in detail, how we are going to tackle the fault tolerance part. Although our method is able to cope with this problem, we feel that concerns should be separated. So it might be advisable that for more practical applications the functional and real time parts of a specification are proven correct by means of a development calculus like ours, whereas fault tolerance should be handled by a method which fits particularly for that.

The total amount of time that has been spent on producing the solution has been

[3] The exposition in this section is extracted from the presentation in [2]

– Lyngby team: 3 mm, for requirements and global design.
– Oldenburg team: 3 mm, for interface and transformation to programs.

In both cases significantly larger design were developed in intermediate stages.

Experience shows that the amount of work that has to be spent by a good MSc student on learning the method is fairly high. On the other hand the calculus is intended to be used on the syntactic level. So no deep understanding of the semantics is required. Our experience with students shows that a master's thesis is a fair estimation for the work required. This includes a fair understanding of the semantics.

For non scientific application there is an additional premise for a good understanding: An average programmer must be willing to stick to the limited number of development steps offered by a calculus rather than doing free style programming. Of course also a certain experience in the use of the method is necessary, but we believe that it does not exceed the average learning overhead for any software development method.

References

1. J. He, C. A. R. Hoare, M. Fränzle, M. Müller-Olm, E.-R. Olderog, M. Schenke, M. R. Hansen, A. P. Ravn, and H. Rischel. Provably correct systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *LNCS*, pages 288–335. Springer-Verlag, 1994.
2. H. Rischel, J. Cuellar, S. Mørk, A. P. Ravn, and I. Wildgruber. Development of safety-critical real-time systems. In M. Bartošek, J. Staudek and J. Wiedermann, editors *Proceedings of SOFTSEM'95*, volume 1012 of *LNCS*, pages 206–335. Springer-Verlag, 1995.

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