Preliminary report for the

Dagstuhl-Seminar 9523:

Methods for Semantics and Specification

Organizers: Jean-Raymond Abrial
Egon Boerger
Hans Langmaack

Contents

Introduction

The goal of the workshop was to bring together

a) researchers representing the major formal methods and

b) representative potential users of such methods in industry

for a critical comparison of the advantages and drawbacks of such methods for practical applications.

In order to have a concrete basis for such an evaluation we have asked the participants to prepare a solution to a problem which was on the one side large enough to exhibit features of a real-life problem and on the other side small enough to be doable in a reasonable amount of time. Since we intend to contribute to an evaluation from the point of view of the use of formal methods for practical applications we have chosen a problem which comes from a non-computer-science application domain and in a form which is typical for industrial informal requirement specifications. The steam-boiler-control specification problem which has been sent out to the participants nine months before the workshop is reprinted below.

We have deliberately abstained from imposing any specific constraints on the expected solution. The idea was not to exclude any approach and to permit each participant to exhibit his favourite method by showing it at its best, be it by providing a formal requirement specification, an architectural design, a running program, a sequence of stepwise refinements or an analysis and proof of behavioural properties one wants to establish for the system. This decision to keep the specific task as free as possible allowed us also to neutralize the otherwise not acceptable circumstance that there was no possibility of interaction with the virtual customer. This freedom is reflected in the great variety of contributions which have been presented at the workshop.

Four months before the workshop some additional requirements for the physical behaviour of the physical components of the steam-boiler have been sent to the participants for possible inclusion into the formal model and as a possible test for the adaptability of the used method to requirement modifications. One month before the workshop A. Lötzbeyer from the FZI in Karlsruhe provided the participants also with an environment in which to run the implementations of proposed solutions.

The week in Dagstuhl was characterized by a fruitful, vivid, frank and critical discussion of a wide spectrum of different (also partial) solutions presented by researchers from the major approaches to formal methods in program development. The abstracts of these presentations are included below. Not only many new contacts have been established among the participating research groups. Above all a great number of specific comparisons of the different methods have become possible through the focus on the steam-boiler problem.

Through an evening session stimulated by some provocative statements by G. Goos and through a critical review and evaluation on Friday morning by T. Lindner, J. Loeckx, P.Y.A. Ryan and M. Sintzoff which was followed by a final discussion among the participants we arrived at formulating some specific criteria and parameters which can help to evaluate the various solutions and to compare the specific merits and drawbacks of the employed formal methods. We want to stress that this is not to be understood in the sense of grading but in the perspective of a contribution to the development of evaluation criteria by which various features of specification methods can be compared on objective grounds. We enclose below the list of parameters which we suggest to the participants as a guide for evaluating the final version of their (full or partial) solution to the steam-boiler problem with respect to the others.

The success of the workshop encouraged us to prepare a Case Study Book "Formal Methods for Industrial Applications: The Steam-Boiler Case Study". The participants and other interested researchers are asked to complete their solution of the steam-boiler problem in the light of the discussion in Dagstuhl and to submit it for a regular reviewing process until December 1995. We hope to get the reviews within three months after the submission. The final versions of the accepted papers are due two months after the reports (hopefully early Summer of 1996) so that the book can be out in the Fall of 1996. In order to promote the interaction among the different approaches a WWW page has been established at the University of Kiel through which every group can access the material made available by the other groups for comparison. The page can be accessed by:

`http://www.informatik.uni-kiel.de/~procos/dag9523/dag9523.html`

We hope to further contribute with this Case Study Book to the definition of useful evaluation criteria which can help to identify where we are and to establish the practical value of formal methods for industrial software engineering applications. We thank Schloss Dagstuhl for the excellent environment which helped a lot to turn our experiment into a fruitful experience.

The organizers
Jean-Raymond Abrial (Paris) Egon Boerger (Pisa) Hans Langmaack (Kiel)

Evaluation criteria

1. Does the solution comprise a requirements specification?
   a. for the system components?
   b. for the safety of the system?
   c. for the liveness of the system?
   d. for the performance of the system?

1'. Is the requirements specification of the solution:

- formal?
- rigorous?
- verbal?

2. Does the solution comprise a functional design?

   a. for the control program specification?
   b. for the water level specification?
   c. for the steam sensor specification?
   d. for the pump actuator specification?
   e. for the pumps specification?
   f. for the drain specification?
   g. for the operator input/output?

2'. Has the functional design of the solution been verified against the requirements specification?

- formally?
- rigorously?
- verbally?

3. Does the solution comprise an architectural design?

   a. for the control program specification?
   b. for the message passing specification?
   c. for the steam sensor specification?
   d. for the water level sensor specification?
   e. for the pump actuator specification?
   f. for the pumps sepcification?
   g. for the drain specification?
   h. for the operator input/output?

3'. Has the architectural design of the solution been verified against the requirements specification?

- formally?
- rigorously?
- verbally?

4. 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?

5. What are the comparable other solutions?

6. What other solutions complement the given solution?

7.
   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.
   b. How much preparation is needed to become sufficiently expert of the used specification framework in order to be able to produce a solution to such a problem in that framework? Indicate the number of weeks of training which you believe is needed for an average programmer to learn the method.

8. 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 everage programmer understand the solution?
   c. How much time do you believe is necessary for the average programmer whithout knowledge of the used specification method to learn what is needed to be able to understand the solution?

Specification problem

Introduction

This section constitutes an informal specification of a program which serves to control the level of water in a steam-boiler. It is important that the program works correctly because the quantity of water present when the steam-boiler is working has to be neither too low nor to high; otherwise the steam-boiler or the turbine sitting in front of it might be seriously affected.

The proposed specification is derived from an original text that has been written by Lt-Col. J.C. Bauer for the Institute for Risk Research of the University of Waterloo, Ontario, Canada. The original text has been submitted as a competition problem to be solved by the participants of the International Software Safety Symposium organized by the Institute for Risk Research. It has been given to us by the Institut de Protection et de Sureté Nucléaire, Fontenay-aux-Roses, France. We would like to thank the author, the Institute for Risk Research and the Institut de Protection et de Sureté Nucléaire for their kind permission to use their text.

The text to follow is severely biased to a particular implementation. This is very often the case with industrial specifications that are rarely independent from a certain implementation people have in mind. In that sense, this specification is realistic. Your first formalization steps could be *much more abstract* if that seems important to you (in particular if your formalism allows you to do so). In other words, you are encouraged to *structure* your specification in a way that is not necessarily the same as the one proposed in what follows. But in any case, you are asked to demonstrate that your specification can be refined to an implementation that is close enough to the functional requirements of the "specification" proposed below.

You might also judge that the specification contains some loose ends and inconsistencies. Do not hesitate to point them out and to take yourself some appropriate decisions. The idea, however, is that such inconsistencies should be solely within the *organization* of the system and *not within its physical properties.*

We are aware of the fact that the text to follow does not propose any precise model of the physical evolution of the system, only elementary suggestions. As a consequence, you may have to take some simple, even simplistic, abstract decisions concerning such a physical model.

Physical environment

The system comprises the following units

- the steam-boiler

- a device to measure the quantity of water in the steam-boiler
- four pumps to provide the steam-boiler with water
- four devices to supervise the pumps (one controller for each pump)
- a device to measure the quantity of steam which comes out of the steam-boiler
- an operator desk
- a message transmission system

The steam-boiler

The steam-boiler is characterized by the following elements:

- A valve for evacuation of water. It serves only to empty the steam-boiler in its initial phase.
- Its total capacity C (indicated in litres).
- The minimal limit quantity M_1 of water (in litres). Below M_1 the steam-boiler would be in danger after five seconds, if the steam continued to come out at its maximum quantity without supply of water from the pumps.
- The maximal limit quantity M_2 of waters (in litres). Above M_2 the steam-boiler would be in danger after five seconds, if the pumps continued to supply the steam-boiler with water without possibility to evacuate the steam.
- The minimal normal quantity N_1 of water in litres to be maintained in the steam-boiler during regular operation ($M_1 < N_1$).
- The maximal normal quantity N_2 of water (in litres) to be maintained in the steam-boiler during regular operation ($N_2 < M_2$).
- The maximum quantity W of steam (in litres/sec) at the exit of the steam-boiler.
- The maximum gradient U_1 of increase of the quantity of steam (in litres/sec/sec).
- The maximum gradient U_2 of decrease of the quantity of steam (in litres/sec/sec).

The water level measurement device

The device to measure the level of water in the steam-boiler provides the following information

- the quantity q (in litres) of water in the steam-boiler.

The pumps

Each pump is characterized by the following elements

- Its capacity P (in litres/sec)
- Its functioning mode: on or off
- it's being started: after having been switched on the pump needs five seconds to start pouring water into the boiler (this is due to the fact that the pump does not balance instantaneously the pressure of the steam-boiler).
- it's being stopped: with instantaneous effect

The pump control device

Each pump controller provides the following information:

- the water circulates from the pump to the steam-boiler or, in the contrary, it does not circulate.

The steam measurement device

The device to measure the quantity of steam which comes out of the steam-boiler provides the following information:

- a quantity of steam v (in litres/sec).

Summary of constants and variables

The following tables summarize the various constants or physical variables of the system:

	Unit	**Comment**
		Quantity of water in the steam-boiler
C	litre	Maximal capacity
M_1	litre	Minimal limit
M_2	litre	Maximal limit
N_1	litre	Minimal normal
N_2	litre	Maximal normal

	Unit	Comment
		Outcome of steam at the exit of the steam-boiler
W	litre/sec	Maximal quantity
U_1	litre/sec/sec	Maximum gradient of increase
U_2	litre/sec/sec	Maximum gradient of decrease
		Capacity of each pump
P	litre/sec	Nominal capacity
		Current measures
q	litre	Quantity of water in the steam-boiler
p	litre/sec	Throughput of the pumps
v	litre/sec	Quantity of steam exiting the steam-boiler

The overall operation of the program

The program communicates with the physical units through messages which are transmitted over a number of dedicated lines connecting each physical unit with the control unit. In first approximation, the time for transmission can be neglected.

The program follows a cycle and a priori does not terminate. This cycle takes place each five seconds and consists of the following actions:

- Reception of messages coming from the physical units.
- Analysis of informations which have been received.
- Transmission of messages to the physical units.

To simplify matters, and in first approximation, all messages coming from (or going to) the physical units are supposed to be received (emitted) *simultaneously* by the program at each cycle.

Operation modes of the program

The program operates in different modes, namely: *initialization, normal, degraded, rescue, emergency stop.*

Initialization mode

The *initialization* mode is the mode to start with. The program enters a state in which it waits for the message STEAM-BOILER_WAITING to come from the physical units. As soon as this message has been received the program checks whether the quantity of steam coming out of the steam-boiler is really zero. If the unit for detection of the level of steam is defective—that is, when v is not equal to zero—the program enters the *emergency stop* mode. If the quantity of water in the steam-boiler is above N_2 the program activates the valve of the steam-boiler in order to empty it. If the quantity of water in the steam-boiler is below N_1 then the program activates a pump to fill the steam-boiler. If the program realizes a failure of the water level detection unit it enters the *emergency stop* mode. As soon as a level of water between N_1 and N_2 has been reached the program can send continuously the signal PROGRAM_READY to the physical units until it receives the signal PHYSICAL_UNITS_READY which must necessarily be emitted by the physical units. As soon as this signal has been received, the program enters either the mode *normal* if all the physical units operate correctly or the mode *degraded* if any physical unit is defective. A transmission failure puts the program into the mode *emergency stop*.

Normal mode

The normal mode is the standard operating mode in which the program tries to maintain the water level in the steam-boiler between N_1 and N_2 with all physical units operating correctly. As soon as the water level is below N_1 or above N_2 the level can be adjusted by the program by switching the pumps on or off. The corresponding decision is taken on the basis of the information which has been received from the physical units. As soon as the program recognizes a failure of the water level measuring unit it goes into *rescue* mode. Failure of any other physical unit puts the program into *degraded* mode. If the water level is risking to reach one of the limit values M_1 or M_2 the program enters the mode *emergency stop*. This risk is evaluated on the basis of a maximal behaviour of the physical units. A transmission failure puts the program into *emergency stop* mode.

Degraded mode

The *degraded* mode is the mode in which the program tries to maintain a satisfactory water level despite of the presence of failure of some physical unit. It is assumed however that the water level measuring unit in the steam-boiler is working correctly. The functionality is the same as in the preceding case. Once all the units which were defective have been repaired, the program comes back to *normal* mode. As soon as the program sees that the water level measuring unit has a failure, the program goes into mode *rescue*. If the water level is risking to reach one of the limit values M_1 or M_2 the program enters the mode *emergency stop*. A transmission failure puts the program into *emergency stop* mode.

Rescue mode

The *rescue* mode is the mode in which the program tries to maintain a satisfactory water level despite of the failure of the water level measuring unit. The water level is then estimated by a computation which is done taking into account the maximum dynamics of the quantity of steam coming out of the steam-boiler. For the sake of simplicity, this calculation can suppose that exactly n liters of water, supplied by the pumps, do account for exactly the same amount of boiler contents (no thermal expansion). This calculation can however be done only if the unit which measures the quantity of steam is itself working and if one can rely upon the information which comes from the units for controlling the pumps. As soon as the water measuring unit is repaired, the program returns into mode *degraded* or into mode *normal*. The program goes into *emergency stop* mode if it realizes that one of the following cases holds: the unit which measures the outcome of steam has a failure, or the units which control the pumps have a failure, or the water level risks to reach one of the two limit values. A transmission failure puts the program into *emergency stop* mode.

Emergency stop mode

The *emergency stop* mode is the mode into which the program has to go, as we have seen already, when either the vital units have a failure or when the water level risks to reach one of its two limit values. This mode can also be reached after detection of an erroneous transmission between the program and the physical units. This mode can also be set directly from outside. Once the program has reached the *Emergency stop* mode, the physical environment is then responsible to take appropriate actions, and the program stops.

Messages sent by the program

The following messages can be sent by the program:

- MODE(m): The program sends, at each cycle, its current mode of operation to the physical units.
- PROGRAM_READY: In *initialization* mode, as soon as the program assumes to be ready, this message is continuously sent until message PHYSICAL_UNITS_READY coming from the physical units has been received.
- VALVE: In *initialization* mode this message is sent to the physical units to request opening and then closure of the valve for evacuation of water from the steam-boiler.
- OPEN_PUMP(n): This message is sent to the physical units to activate a pump.
- CLOSE_PUMP(n): This message is sent to the physical units to stop a pump.
- PUMP_FAILURE_DETECTION(n): This message is sent (until receipt of the corresponding acknowledgement) to indicate to the physical units that the program has detected a pump failure.
- PUMP_CONTROL_FAILURE_DETECTION(n): This message is sent (until receipt of the corresponding acknowledgement) to indicate to the physical units that the program has detected a failure of the physical unit which controls a pump.

- LEVEL_FAILURE_DETECTION: This message is sent (until receipt of the corresponding acknowledgement) to indicate to the physical units that the program has detected a failure of the water level measuring unit.
- STEAM_FAILURE_DETECTION: This message is sent (until receipt of the corresponding acknowledgement) to indicate to the physical units that the program has detected a failure of the physical unit which measures the outcome of steam.
- PUMP_REPAIRED_ACKNOWLEDGEMENT(n): This message is sent by the program to acknowledge a message coming from the physical units and indicating that the corresponding pump has been repaired.
- PUMP_CONTROL_REPAIRED_ACKNOWLEDGEMENT(n): This message is sent by the program to acknowledge a message coming from the physical units and indicating that the corresponding physical control unit has been repaired.
- LEVEL_REPAIRED_ACKNOWLEDGEMENT: This message is sent by the program to acknowledge a message coming from the physical units and indicating that the water level measuring unit has been repaired.
- STEAM_REPAIRED_ACKNOWLEDGEMENT: This message is sent by the program to acknowledge a message coming from the physical units and indicating that the unit which measures the outcome of steam has been repaired.

Messages received by the program

The following messages can be received by the program:

- STOP: When the message has been received three times in a row by the program, the program must go into *emergency stop*.
- STEAM_BOILER_WAITING: When this message is received in *initialization* mode it triggers the effective start of the program.
- PHYSICAL_UNITS_READY: This message when received in *initialization* mode acknowledges the message PROGRAM_READY which has been sent previously by the program.
- PUMP_STATE(n, b): This message indicates the state of pump n (open or closed). This message must be present during each transmission.
- PUMP_CONTROL_STATE(n, b): This message gives the information which comes from the control unit of pump n (there is flow of water or there is no flow of water). This message must be present during each transmission.
- LEVEL(v): This message contains the information which comes from the water level measuring unit. This message must be present during each transmission.

- STEAM(v): This message contains the information which comes from the unit which measures the outcome of steam. This message must be present during each transmission.
- PUMP_REPAIRED(n): This message indicates that the corresponding pump has been repaired. It is sent by the physical units until a corresponding acknowledgement message has been sent by the program and received by the physical units.
- PUMP_CONTROL_REPAIRED(n): This message indicates that the corresponding control unit has been repaired. It is sent by the physical units until a corresponding acknowledgement message has been sent by the program and received by the physical units.
- LEVEL_REPAIRED: This message indicates that the water level measuring unit has been repaired. It is sent by the physical units until a corresponding acknowledgement message has been sent by the program and received by the physical units.
- STEAM_REPAIRED: This message indicates that the unit which measures the outcome of steam has been repaired. It is sent by the physical units until a corresponding acknowledgement message has been sent by the program and received by the physical units.
- PUMP_FAILURE_ACKNOWLEDGEMENT(n): By this message the physical units acknowledge the receipt of the corresponding failure detection message which has been emitted previously by the program.
- PUMP_CONTROL_FAILURE_ACKNOWLEDGEMENT(n): By this message the physical units acknowledge the receipt of the corresponding failure detection message which has been emitted previously by the program.
- LEVEL_FAILURE_ACKNOWLEDGEMENT: By this message the physical units acknowledge the receipt of the corresponding failure detection message which has been emitted previously by the program.
- STEAM_OUTCOME_FAILURE_ACKNOWLEDGEMENT: Through this message the physical units acknowledge the receipt of the corresponding failure detection message which has been emitted previously by the program.

Detection of equipment failures

The following erroneous kinds of behaviour are distinguished to decide whether certain physical units have a failure:

- PUMP: (1) Assume that the program has sent a start or stop message to a pump. The program detects that during the following transmission that pump does not indicate its having effectively been started or stopped. (2) The program detects that the pump changes its state spontaneously.

- PUMP_CONTROLLER: (1) Assume that the program has sent a start or stop message to a pump. The program detects that during the second transmission after the start or stop message the pump does not indicate that the water is flowing or is not flowing; this despite of the fact that the program knows from elsewhere that the pump is working correctly. (2) The program detects that the unit changes its state spontaneously.

- WATER_LEVEL_MEASURING_UNIT: (1) The program detects that the unit indicates a value which is out of the valid static limits–i.e. between 0 and C. (2) The program detects that the unit indicates a value which is incompatible with the dynamics of the system.

- STEAM_LEVEL_MEASURING_UNIT: (1) The program detects that the unit indicates a value which is out of the valid static limits—i.e. between 0 and W. (2) The program detects that the unit indicates a value which is incompatible with the dynamics of the system.

- TRANSMISSION: (1) The program receives a message whose presence is aberrant. (2) The program does not receive a message whose presence is indispensable.

Solutions

Evolving Algebra Solution for the Steam Boiler Problem

Egon Börger
Università di Pisa, Italy
Igor Ðurđanović
Universität-GH Paderborn, Germany
Uwe Glässer
Universität-GH Paderborn, Germany
Yuri Gurevich
University of Michigan, Ann Arbor, USA
Elvinia Riccobene
Cittá Universitaria, Catania, Italy

Computation models and commercial specification methods seem to be worlds apart. The evolving algebra project started as an attempt to bridge the gap by improving on Turing's thesis. We sought more versatile abstract machines able to simulate arbitrary algorithms in a direct and coding-free way on their natural abstraction level (rather than implementing algorithms on lower abstraction levels).

The EA thesis asserts that evolving algebras are such versatile machines, and ealgebras are used to model discrete dynamic systems. They are executable when appropriate. (There are tools for running evolving algebras.) The use of the successive refinement method is facilitated by the ability to reflect arbitrary abstraction levels.

There is an active and growing evolving algebra community. Evolving algebras have been used to specify languages (e.g. C++, Prolog and VHDL), to specify real and virtual architectures (e.g. APE, PVM and Transputer), to validate standard language implementations (e.g. of Prolog, Occam), to verify numerous distributed and/or real-time protocols, to prove complexity results, etc.. Some examples may be found in the Proceedings of IFIP 1994 World Computer Congress.

In the talk, we quickly explain evolving algebras and the EA thesis but the bulk of the talk is devoted to the solution of the steam boiler problem. The given informal specification has been very faithfully reflected in the *ground model* of the steam boiler system. It is not that we were so happy with the informal specification. In a real situation we could negotiate with the designer but this was impossible in this case, and thus we reflected the given description very faithfully (the customer is always right). The presentation was supposed to convince one that the formalization is faithful indeed. In the paper, we discuss also a more abstract presentation of the system as well as a number of refinements of the ground model all the way to a running program presented during the demo session. The final version of the paper will contain also proofs of various properties of the system.

Designing the Steam Boiler Control Using an Assertional Method supported by the Interactive Proof Checker PVS

Jan Vitt

Chr.-Albrechts-Universität Kiel, Germany

Jozef Hooman

Eindhoven University of Technology, The Netherlands

We aim at a formal derivation of an implementation of the steam boiler control system. Using a mixed formalism in which programs and assertional specifications are combined within a single framework, the top-level specification is refined in a top-down manner. Having the formalism defined in the specification language of PVS (Prototype Verification System), the interactive proof checker of PVS is used to check the correctness of each refinement step, thereby guaranteeing the correctness of the resulting program.

As a start, we consider a real-time version of the steam boiler control system which contains only one pump and one pump controller. Further we assume absence of failures and consider the initialization phase as terminated, i.e., we focus on the normal mode. To design the control system we follow a standard approach that has been applied successfully to similar examples. In the first step 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 given. Next, relevant properties of the physical processes to be controlled are formalized and a control strategy in terms of a continuous interface is given. Then we prove that this strategy and the assumptions about the physical properties lead to the top-level specification. To implement the control strategy we introduce a water sensor and a pump system to transform the continuous interface into discrete events. The pump system is refined further into a pump and a pump controller. Next the control program is specified and we show that it implies the control strategy, using specifications of actuators and sensors. This reduces the problem to a conventional programming problem where we have to implement a program according to its (real-time) specification.

In a second version, communication channels can fail, i.e., they can loose messages. By a slight adaptation of the previous design, we obtain a control component which switches into emergency stop mode after detection of a failure on a channel. Finally, failures of physical components are considered. As an example we have chosen a failing water sensor. Then, a steam sensor and a sensor control which receives values from the steam sensor are added to the system. Compared to the original description of the steam boiler system, we have designed the steam boiler control system only partially. However, the correctness of all design steps has been verified by means of PVS.

Varieties of Steam Boilers

Anders P. Ravn
Technical University of Denmark

The Steam Boiler problem is a challenge to formal methods because it involves issues ranging from the interface to control theory and control engineering over fault-tolerant and reactive real-time systems to implementation and simulation. In the presentation, I suggest to document a development using a layered approach:

1. *requirements specification* for the overall dynamical system,
2. *functional design* in terms of real-time state machines interacting with and thus controlling the dynamical system.
3. *architectural design* defining a collection of cooperating sequential processes,
4. *programming* in hardware or software of these processes and their interfaces,
5. *integration* of the completed system.

A significant point is that layers have to be linked by interface documents, preferably based on transformations between the specialized notations and theories used to develop each layer. Development of such links may be the major technical challenge to formal methods supporting a single layer.

The presentation illustrates requirements specification using Duration Calculus and Z-schemas, while later presentations by Wang and Rischel illustrate a corresponding functional design, and a presentation by Schenke illustrates transformations to an architectural level with occam-like processes.

A previous survey is found in: Jifeng 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 *Lecture Notes in Computer Science*, pages 288–335. Springer-Verlag, 1994.

A Duration Calculus Approach to Specifying a Steam-boiler Control System

Wang JuAn
University of Macau

Li XiaoShan
UNU/IIST, Macau

This report presents a general method to cope with hybrid control system specification and verification by making full use of both the simplicity of finite automata and the excessive expressing power of Mean Value Duration Calculus. The method consists of four steps: First, specify the requirements of the whole system through Mean Value Duration Calculus formulas. Second, use two real-time communicating automata to characterize the design decisions, which describe the plant and control subsystems respectively. Third, translate the automata into Mean Value Duration Calculus formulas. Finally prove the design decisions conforming to the requirements specification. Taking the steam-boiler problem raised by J.-R. Abrial[1] as a running example, we demonstrate the above method in three versions of abstractions. Each version represents an abstract level for capturing the original problem requirements and provides a complete procedure of applying the above method. In addition to specifying safety property, emphasis is put on capturing non-functional requirements such as performance, reliability, and optimization. Some formal design techniques about finding an optimal implementation conforming to the system specification are also discussed.

Keywords: Specification, Verification, Mean Value Duration Calculus, Steam-boiler control, Hybrid systems.

Specification and Development of the Steam Boiler Control Program

Jean-Raymond Abrial
Paris, France

We presented a partial version of the formal specification and development of the Boiler Case Study. We insisted on the necessity of a "pre-specification" where some equations relating dynamically the variables of the system are established. Then, these equations are sorted and the architecture of the system is constructed little by little by taking account of the equations in turn — the architecture is thus constructed by working against the future flow of control in the system.

We have shown how such a structure can "resist" some modifications of the original specification.

Calculi for Reactive Systems

Burghard v. Karger
Chr.-Albrechts-Universität Kiel, Germany

The relational calculus has been remarkably successful in reasoning about possibly non-deterministic systems, provided that their behaviour can be fully characterised by observation of just their initial and final states. Many alternative models have been proposed for reactive systems, whose behaviour between initiation and termination is also of significance. A common feature of these calculi is that past observations cannot be cancelled or undone by what comes after. As a consequence, the converse operation of the relational algebra must be restricted. Doing this we obtain the *Sequential Calculus*, which provides a common framework of laws applicable to many of these alternative models. The absence of a converse operation also endows time with a direction and enables us to treat temporal logic within the sequential calculus.

An Algebraic Specification of the Steam-Boiler Control System

Michel Bidoit
LIENS, C.N.R.S. & Ecole Normale Supérieure, Paris, France
joint work with **C. Chevenier, C. Pellen, and J. Ryckbosch**
Direction des Etudes et Recherches, Electricité de France

We describe how to derive an algebraic specification of the steam-boiler control system from the informal requirements provided by J.R. Abrial. The aim of this formalization process is to analyze the informal requirements, to detect inconsistencies and loose ends, and to translate the requirements into a formal, algebraic, specification. During this process we have to provide interpretations for the unclear or missing parts. We explain how we can keep track of these additional interpretations by localizing very precisely in the formal specification where they lead to specific axioms. Hence we take care of the traceability issues. We explain as well how the formal specification is obtained in a stepwise way. Emphasis is put on how to specify the failures of the steam-boiler. Finally we will briefly discuss validation and verification issues. For this case study we use the Pluss algebraic specification language and the Larch Prover.

Using Timed Automata for the Steam Boiler Controller

Gunter Leeb, Nancy Lynch
Massachusetts Institute of Technology, USA

In this paper we model a steam boiler and its controller as a hybrid system. We apply the Timed Automata method to the steam boiler controller problem as defined for the Methods and Semantics Seminar. Our method allows us to describe different parts of the system independently and to prove formally that our steam boiler controller model guarantees the required safety and performance properties. We present a sufficient model for the boiler environment and another for the controller behavior. In addition this method provides efficient means for proving incrementally increasingly complex model abstractions. We show this in proving that a particular 4-Point Controller preserves the properties proved for the general controller.

Specification and Refinement of Steam Boiler Using Refinement Calculus and Action Systems

Michael Butler, Emil Sekerinski, Kaisa Sere
Åbo Akademi, Turku, Finland

We give an abstract high-level specification of the steam boiler control program together with its physical environment within the action systems framework.

The abstract specification is refined to an implementation within the *refinement calculus* by carrying out a number of data refinement steps. The application of data refinement guarantees that the correctness of the specification is preserved by the implementation.

The idea of partitioning the state space of an action system is used as a way of splitting the system into an open system modelling the controller and an open system modelling the environment.

The approach relies on finding appropriate abstractions for modelling the controller and the physical environment in a simpler way than with a concrete implementation. Abstraction in the initial specification is achieved by:

1. unifying the different ways of system failure into one notion of failure,
2. reducing the number of modes by unifying the *Normal*, *Degraded*, and *Rescue* modes,
3. modelling all actuators, sensors, and controller variables as part of a state space, thus abstracting from their distribution and the message passing protocol.

In later steps these abstractions are refined by applying data refinement to both the controller and the physical system, resulting in an action system specification which is close to an implementation in an imperative programming language.

A functional modelling of the steam boiler

Manfred Broy, Franz Regensburger, Bernhard Schätz
Technische Universität München, Germany

The main goal of our approach is to give a formal requirement specification of the steam boiler problem that specifies WHAT to do and does not already start to formalize HOW the controlling has to be done. This is a crucial point since the problem solving strategy should not be part of the very first formal specification of a problem to leave more freedom for later implementation.

In order to give such a (pure) requirement specification we adopted a technique from control theory and modeled the steam boiler as a closed loop control system including noise channels, noise composers, and the physical environment itself.

The explicit introduction of noise allows us to specify the intended behavior of the controller in certain failure situations without formulating a strategy how the controller could recognize this failure.

For the formal specification of the steam boiler as a closed loop control system we use the framework of stream processing functions. The formal language that is used for the specification is HOLCF (Higher Order Logic of Computable Functions) which itself is implemented in the logical framework Isabelle.

At the moment our requirement specification is still on a generic level. In future work we will refine it to obtain a more concrete requirement specification of the steam boiler problem.

A Modular Approach to the Steam Boiler Problem using Duration Calculus

Simon Mørk, Anders P. Ravn, Hans Rischel
Technical University of Denmark

This talk presents a formalization of requirements to the steam-bolier system and of a top-level functional design. It is proved that the design implements the requirements. As encouraged by Abrial, we have approached the problem with an open mind, and we therefore present out own formalization of the steam-boiler system. Our goal is to replace the monolithic system description of the problem statement with a modular description of a distributed system with "intelligent" sensors and actuators.

The modular description of the steam-boiler system is obtained by subdividing the state space into parts, localized in separate components. The computations related to the states for a component is then described in a separate specification. This specification includes handling of errors for the associated equipment. In this way we get a simpler and more abstract set of interfaces for the central control program, and the computations described in the problem are divided between the sensors, actuators, and the program. The specification of the central control program can therefore concentrate on the core of the control problem: the interplay between data from different components.

We also get a modular subdivision of the operation mode for the system: The global operation mode in the original problem statement is replaced by local modes in each component. This gives a reduction of the number of modes and a simplification of the mode handling for the central control program, as part of the system mode can be handled locally in the intelligent sensor and actuator components.

The Steam Boiler: From Duration Calculus to OCCAM

Michael Schenke
Universität Oldenburg, Germany

The aim of the method presented here is to develop an occam program from an architectural description in special duration calculus formulae, so-called "implementables" [HH94]. The development is done in order to show the applicability of the approach to the development of provably correct systems as lined out in [HH94] and described in [Sch94] and [OS95]. In the talk we give the intermediate forms of the specification at various stages. Most important will be the change from a state based view like the one from DC to an event based view like the one from occam. So at some stage we shall introduce events. Events may be names for state changes or for

synchronisations. They will be introduced in the syntax of SL, a language designed particularly for the specification of real time systems in the Esprit BRA ProCoS. The idea of this part is to successively remove the implementables and replace them by pieces of SL syntax. We shall exhibit the following steps:
1.) Before the events are introduced, the phase description often is still too abstract. A technique called phase splitting enables us to describe a phase in terms of newly introduced subphases. In this article phase splitting will only be used in connection with synchronisation.
2.) Phase changes (events) which change only one observable do not involve internal synchronisation of different components. They normally express a communication with the environment.
3.) Implementables that deal with synchronisation are removed by synchronising communications which in accordance with the occam paradigm appear in the SL specifications of both components involved.
4.) Already at this stage it is possible to argue within an SL specification. In general there are several possible replacements of SL syntax pieces for other such syntax pieces. Here we shall show only one such transformation, the introduction of so-called trace assertions.
5.) The introduction of time restrictions finally makes the quantitative requirements on timing dependant on events rather than on phases.
6.) The development of the final program can be done in two steps: the introduction of a recursion construct which is not part of the occam syntax and secondly the resolution of the recursion by means of standard techniques. Since the latter step does not contain anything new we skip it and simply show the complete final program.

References

[HH94] Jifeng 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 *Lecture Notes in Computer Science*, pages 288–335. Springer-Verlag, 1994.

[Sch94] M.Schenke. Specification and Transformation of Reactive Systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 288–335. Springer-Verlag, 1994.

[OS95] E.-R.Olderog, M.Schenke. Design of Real-Time Systems: Interface between Duration Calculus and Program Specifications. STRICT 1995, to appear in Workshops in Computing.

The Dagstuhl Steam Boiler Controller Problem — The TLT Solution

Jorge Cuellar, Isolde Wildgruber
Siemens Corporate Research and Development, Munich, Germany

TLT is a language for writing requirement specifications, abstract programs, implementations, and their properties. A program consists of several sections (or submodules) which manipulate shared (or local) *variables* or communicate through *actions* with each other and the environment. In these sections it is possible to formulate assumptions over the most general environment, or to trigger events (actions or changes in variables) as immediate response to other events, or to start "spontaneous" instructions (not triggered by external events).

Each TLT section corresponds to a logical module of the program, in the sense that each one can be translated to a transition relation or transition predicate independently of the others (since they contain disjount sets of variables), while the global transition relation is exactly the conjunction of the relations for the individual modules.

In this solution, we distinguish carefully between *program variables* and *physical variables*, which may not be seen directly by the program. To reason about them and to draw conclusions about the controllability of the system (including the detection of physical failures) we use Lamport's TLA. In this logic we express the assumptions about the relation of the physical variables to the sensor and actuator values and we describe the dynamics of the physical system.

DCSL solution of the steam boiler problem

Peter Päppinghaus
Siemens Corporate Research and Development, Munich, Germany

A solution to the steam boiler problem using DCSL (Distributed Control Specification Language) was presented. DCSL is a language for specifying structure and behaviour of (finite) state systems. DCSL is currently being developed with the aim to support the design of industrial size distributed control systems using formal methods. A DCSL description specifies a system of asynchronously communicating FSA's (finite state automata). It consists of a class library of generic classes specifying constraints on the controller behaviour in a logical language, and a structure description of an individual aggregate of components, from which a system of communicating FSA's is automatically synthesized. Each class is equipped with a set of named interfaces, and each interface contains declarations of input-, output-, token-, communication- and history-variables as well as assumptions about the behaviour of its environment and commitments to this environment. A tool environment to support system development with DCSL is currently being implemented. This environment will contain a compiler to synthesize FSA's, a graphical interface for the design of individual aggregates and to animate a synthesized solution using an FSA interpreter, and finally interfaces to a model checker and a theorem prover so that properties of a DCSL design can be formally verified.

A Z specification of the boiler

Pascal Bernard
IUT de Nantes, France

The boiler and its controller are specified together with the Z notation. The purpose of this specification is to relate the evolution of the physical system with the corresponding actions of the controller. The faithful formalisation of the informal specification revealed that it was consistent (in the sense that in no case the controller has to take contradictory decisions) but very loose. This specification allows to prove that the system is safe when failures are detected as epected. It provides a framework for stating liveness properties of the whole system. The proof of some expected liveness properties would require a refinement of the specification.

In order to make the specification, a method was developed. This method applies for the specification of systems where the control takes virtually no time and holds again after every period of a given duration. The method provides a structure of the specification, each part of which concerning a partial view of the system at a given stage of the life of the system. The combination of these parts is easy to handle with the schema calculus of Z. In particular, the exceptions are treated in an easy and elegant way.

This specification shows that it is possible to handle cases of a fair size with Z. This is obtained by structuring. The method proposed can be reused or generalised to handle other cases.

A Canadian Approach to the Specification and Implementation of the Boiler System

Dan Craigen
ORA Canada

This presentation is divided into 4 sections:

1. Technology Transfer (Innovation diffusion)
2. History of the boiler problem.
3. A four country comparison of techniques and (nuclear) regulatory assessment.
4. A discussion of the Canadian approach.

In the first section of the talk I present an innovation diffusion model (developed by Everett Rogers) to assess the likely adoption trajectory by industry of formal methods. The second and third sections provide historical context for both this workshop and for our own work with the boiler. Finally, I describe the three phase Canadian approach supported by the Atomic Energy Board of Canada. Summaries of the work performed by the Canadian company GARD in developing a formal software requirements specification (SRS) and our own work (in conjunction with a Canadian software house—CGI) in using EVES and an assurance process to provide an implementation of the boiler (as specified by the SRS) are presented.

A Steam-Boiler Control Specification with Statecharts and Z

Matthias Weber
Technische Universität Berlin, Germany
joint work with **Robert Büssow** and **Maritta Heisel**

This talk sketches an approach that tries to integrate a mathematical specification technique with a well-known engineering technique to yield a practicable methodology for the specification of safety-critical control systems. A starting point is the technology of Statecharts, which is currently being adopted in industry for the specification of embedded systems. To cope with the growing complexity and the safety requirements of these systems, I propose an integration of the specification language Z into Statecharts, where Z is used to model the data structures and data transformations within the system.

A widely used technique of 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. The approach presented in this talk follows this philosophy and combines three system views, specified with structural diagrams, Statecharts, and Z, respectively, into a coherent modeling technique.

The talk sketches some key ideas of this combination; after that I present a solution to the steam boiler control specification problem. Throughout the presentation, I make an effort to stick faithfully to the specification of this problem as given on pages 8 to 16, especially with respect to the physical interface of the control software. This solution constitutes a system specification of the steam-boiler. At some places I will motivate and describe deviations or adaptations.

Structural synthesis of programs from refined user requirements (Programming boiler control in NUT)

Mattin Addibpour, Enn Tyugu
Royal Institute of Technology, Sweden

The technique of structural synthesis of programs based on automatic proof search in intuitionistic propositional calculus and implemented in the NUT system is used for solving the boiler problem. The goal of the experiment is to bridge a gap between the language of requirements and the implementation. In the NUT system, an appropriate set of concepts (represented as classes) has to be developed for each kind of problems. The concepts for simulating the boiler and programming a boiler model are the following: device, boiler, pump, pumpcontroler, levelmeter, flowmeter, simulator and sysmodel. These concepts are represented as sets of equations taken directly from requirements specification. The control algorithm is represented as a collection of rules written as relations in NUT. The rules are triggered by conjuctions of propositions describing a state of the system at an observable time moment. Also these rules are derived directly from requirements specification after introducing a proper collection of propositions which are implicitly present in the text of requirements. Formally, the boiler control system can be considered as an attributed automaton, whose atribute models are represented as classes and transitions are specified by rules. This technique has been used earlier, in particular, for protocol simulation in NUT. We could demonstrate with the present experiment that the presentation of requirements given as an example allowed us to extract rules immediately from the requirements text. We expect to use the presented technique in combination with a number of formal requirements refinement techniques, providing a reliable implementation tool for refined requiremets.

On the Formalization of Requirements — using Action Semantics

Peter D. Mosses
BRICS, Aarhus, Denmark

Action Semantics has been used successfully to improve the pragmatics of semantic descriptions of programming languages, allowing scaling up to practical languages (e.g., Standard Pascal, ANDF) as well as providing other desirable features such as reusability, modifiability, and readability. The actions used as semantic entities are composed using combinators that correspond to fundamental operational concepts.

Here, we tentatively explore the use of action notation for formally specifying the requirements for the steam-boiler. The control of the steam-boiler is specified as a single compound action, exploiting nondeterminism to express looseness of required behaviour, e.g., that in the initialization phase, the steam-boiler may (apparently) continue to fill or empty the boiler even after the water-level is in the normal range. It is claimed that the action specification could, with rather little investment of effort, be understood by those who wrote the original informal specification, and thus be used as the basis for clarifying the intended requirements – perhaps leading to a major rewrite of the informal specification before investing further effort in the problem... However, it remains to be seen whether the action specification provides an adequate basis for software development in general, or for reasoning about the consequences of the requirements.

Case Study "Production Cell"

Thomas Lindner
Forschungszentrum Informatik, Karlsruhe, Germany

The case study Production Cell, a comparative study currently featuring 35 contributions, is summarized and evaluated. The study, taken from an industrial plant near Karlsruhe, served as a testbed for formal approaches to reactive system design and verification. In order to stimulate a controversial discussion, the evaluation was presented rather provocative. As a major point we see the need to establish an engineering discipline for the application of formal methods which establishes heuristics for system identification, validation, and proof technology.

Furthermore, initial steps towards an evaluation of the case study Steam Boiler were proposed. We see the need for

- assessing what are the key difficulties in the Steam Boiler problem,
- classifying the contributions according to what has been done,

- assessing at least a rough estimate of the effort involved, and
- checking the usefulness of the different mathematical models which were used by the various formal apporaches.