Streams of Steam – The Steam Boiler Specification Case Study

Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies

Institut für Informatik

Technische Universität München

email: {broy,regensbu,schaetz,spiesk}@informatik.tu-muenchen.de

10. November 1998

Abstract: FOCUS is a mathematics-based methodology for the development of distributed reactive systems. This paper sketches the expressiveness and the usefulness of the well-tuned concepts of FOCUS by its application of the requirements specification of a steam boiler, see [Abr96]. We demonstrate the support for non-specialists in formal methods by using user-friendly description techniques, here the state-oriented specification with table notation techniques, and a broad range of refinement techniques, here the concept of data and state refinement, which eases the explicit modeling of fault-tolerant behavior. Additionally we prove its adaptability in different domains by handling an example of the "control theory" application field. Because of complexity of the above mentioned aspects we do not give a complete and detailed specification, but rather concentrate on the control task starting from classical control theory, the conceptual formal model of FOCUS and its possibilities for formal modeling and system development.

1 Introduction

Formal development methods like FOCUS or ProCoS, formal modeling techniques like TLA or UNITY and many others (see [ABL96] and [BMS96] for short descriptions and a comparison based on a example) have often been criticized for minor real-world usefulness. In fact engineers who recognized the need for more accurate and reliable description techniques during the development process ask for user-friendly, well-known notations and their applicability in different (industrially oriented) domains. The general aim of this paper is therefore to demonstrate how a structural approach for the development of a controller can profit from the use of a formal development method. The steamboiler controller is an example concerning control theory, a domain with a long tradition applied to physical systems that are "low-level" from (formal) software-engineer view. We show how user-friendly description-techniques can be used to formally and therefore concisely specify control theoretic behavior.

FOCUS is a mathematical framework for the specification, refinement, and verification of distributed, reactive systems (see [BDD93], [Bro93a], or [Bro93b]). Recent developments in FOCUS concentrate on establishing a closer connection between its theoretical framework and more industry oriented approaches (see, for instance, [Fuc93], [Spi94], [SHB95], [PS97], [Spi98], [Hin98a], [Hin98b]) by integrating well-known description techniques, giving methodological hints and applying them to industrial case-studies. In this context, applying FOCUS to the steam boiler case study ([Abr96]) led us to a couple of questions re-

quiring a closer look at certain especially methodological aspects. We concentrate on the following major contributions in this article:

1. FOCUS in a specific application domain: Control Theory
2. FOCUS with user-friendly description techniques: Tables and States
3. FOCUS for development of correct software: State Refinement

Dealing with the first point we explore the modeling of control theoretic aspects with the semantic basics of FOCUS, the stream processing functions. Control theory (see, for example, [KK94]) has developed well-tuned models for the description of systems with the objective to regulate and control physical devices and processes on a hardware oriented level. Thus, it seems quite natural to exploit this knowledge and to combine it with the concept of reactive systems demonstrating a way for giving precise formalizations of control tasks. This is a first step towards forming a standard approach suitable for high-level and therefore more abstract treatment of controlled physical processes.

After showing that the modeling of control theoretic aspects is possible within the framework of FOCUS, the second point deals with the user-friendly presentation of such specifications. We show how the requirements of such a control task can be expressed in a way accessible for the industrial user without losing the preciseness of the formal method. Engineers often use tables for specifying the behavior of mechanical or electrical devices. Tables allow a well-structured representation of large pieces of information and they are a well-known concept used in the process of clarifying the required system properties. often, they are easier to comprehend and to communicate than logical formulas[1]. The steam boiler example shows that a large set of requirements can be expressed using a tabular notation. We give a semantics for tables in the mathematical model of FOCUS and apply them to the steam boiler controller. Furthermore, we show the limitation of this notation, and demonstrate how the general framework helps to overcome this by combining tables with general predicative description techniques.

Based on a precise and formal but possibly very abstract description of the systems behavior, the concept of refinement allows the treatment of design decisions and the forming of more detailed specifications in a formal and correct way. Our third point deals with this important feature of FOCUS by demonstrating the handling of fault-tolerant behavior. While FOCUS offers a set of quite sophisticated refinement techniques, we will concentrate on two refinement notions, namely data refinement and state refinement. In particular, we demonstrate their embedding into the tabular framework and how the refinement concepts are adapted to the notation of states and tables explained by the steam boiler example.

Specifications of systems behavior covers a very important point into the software development process because they form the communication medium between the application expert, the requirements engineer, the system designer and the implementer. Formal specifications provides the basis for formal verification and therefore for software with guaranteed correct behavior. The overall aim of this article is to demonstrate how FOCUS successively combines different aspects of the engineering process. We show some facettes of this very

---

[1] See [WT94] for an intensive discussion of this subject.

expressive framework, in particular how the requirements engineering process profits from a precise and mathematical basis combined with industrial-strength description techniques. However, we want to point out that this article merely offers an outline how the described concepts of FOCUS can be put to work. We are very well aware of the fact that this is not a sufficiently detailed and complete specification of the steam boiler controller to meet industrial requirements.

The paper starts with a description of the steam boiler system. In section 3 we introduce the most essential concepts of control theory, and demonstrate their connection to the FOCUS mathematical model. In section 4 we explain how to use FOCUS in a state-based manner as well as using tables as a convenient notion. In section 5 we use the introduced concepts specifying the steam boiler controller. Using behavioral refinement techniques we extend the original behavior of the controller to cope with faulty sensorial input in section 6. Finally, we give a short conclusion in section 7.

2 The Steam Boiler System

The general purpose of the steam boiler system, as shown in Figure 1, is to ensure a safe operation of the steam boiler. The steam boiler operates safely if the contained amount of water never exceeds a certain tolerance, thus avoiding damage to the steam boiler and the turbine driven by the produced steam. Basically, the steam boiler system consists of

- the steam boiler itself,
- a measuring device for the water level,
- a pump to provide the steam boiler with water,
- a measuring device for the pump status,
- a measuring device for the amount of steam produced by the steam boiler,
- an operator desk,
- and a message transmission system for the signals produced.

During operation, the water level is kept within the tolerance level as long as possible, using the measuring devices and the pump and producing status information for the operator desk. But even with some devices broken, the system can still successfully monitor the steam boiler. If no safe operation is possible any longer, control is handed over to the operator desk. Additionally, the operator can stop the system at any time via the operator desk.

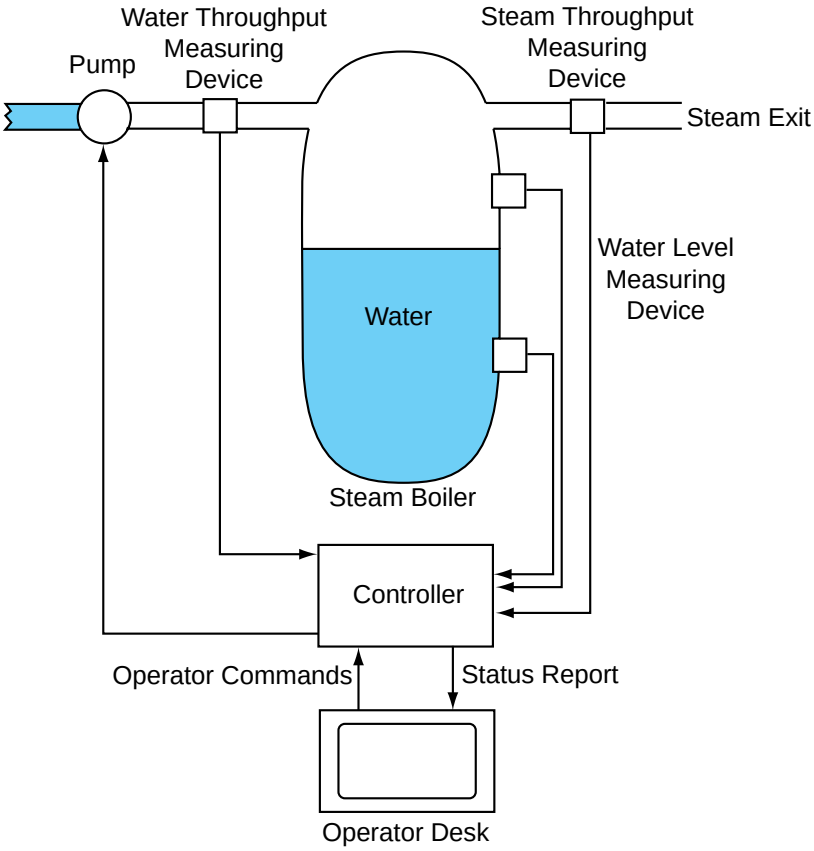

Figure 1: The Steam Boiler System

3 Streams Go Control Theory

One of the objectives of this case study, as mentioned above, is the modeling of a system that guarantees fault tolerance concerning its sensory input as one of its main features. In order to do so, we consider it necessary to model not only the system in question. We also model the environment of the system consisting of the controlled physical process as well as the physical devices like actors (pumps, valves) and sensors (steam flow measurement units, water throughput measurement units, water level measurement units). Together with the modeling of those components we are interested in the modeling of a possible failure of those components leading to natural definition of the requirements of a fault tolerant system.

Since control theory has a long tradition in the treatment of those models we consider it worthwhile to take a short look at the techniques used there. Therefore we introduce the most basic concepts of control theory in section 3.1. Control theory is generally applied to physical, hardware oriented systems that are `low level´ from our point of view, since FOCUS mostly deals with more abstract, software oriented systems. Thus, it is furthermore necessary to express the control theoretic approach in a framework more suitable for a more abstract point of view. Since stream processing functions (see, for instance, [BDD93]) proved to provide a powerful basis for the modeling of reactive, distributed systems, we will use them as our formal model.

Thus, after giving a short introduction to the concept of stream processing functions in section 3.2.1, we will map the control theoretic terms introduced in section 3.1 on appropriate concepts of the theory of stream processing functions. This is done in section 3.2.2.

The main issue of this section is the general demonstration of the usefulness and usability of control theoretic concepts in the framework of stream processing functions. The problem specific questions that arise during the application of this approach to the development of the steam boiler controller will be discussed in section 5.

3.1 Terms from Control Theory

In the control theoretic approach, a system always consists of four components, as depicted in

Figure 2:

- the controller, which is the component to be implemented,
- the plant or physical process, which is the part of the environment which is to be controlled,
- the control composer, which is the *virtual* component which adds noise to the controller output,
- the feedback composer, which is the *virtual* component which adds noise to the sensory input of the controller.

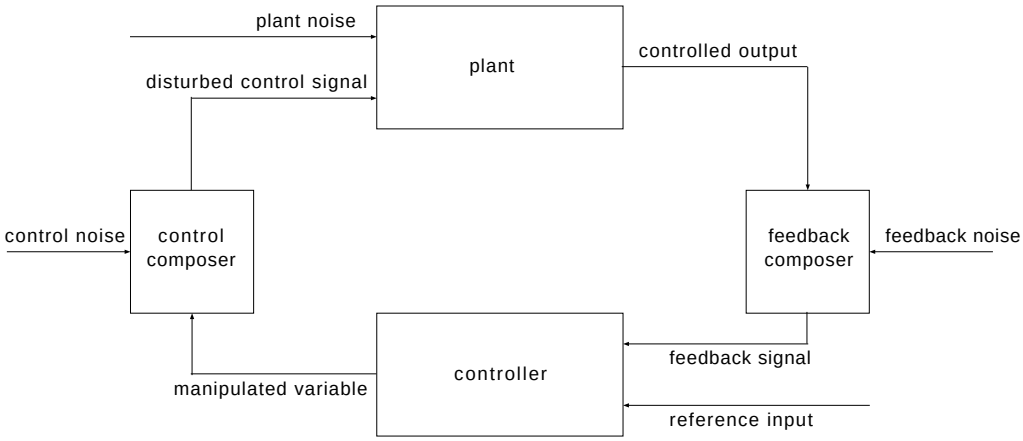

Figure 2: Control Theory System View

This point of view as explained above arises from the fact, that the control task is viewed as a closed control loop. The controller has to manipulate a set of variables influencing the behavior of a physical process or plant (`*manipulated variable´*) to tune this plant to a certain value. To do so, the controller relies on signals reporting the state of the plant (`*controlled output´*). However, both the manipulating values as well as the reporting signals are distorted by noise (`*control noise´*, `*feedback noise´*). The noise is added to the basic signals by corresponding composers (`*control composer´*, `*feedback composer´*) to produce the resulting distorted signals (`*disturbed control signal´*, `*feedback signal´*). Thus, noise can be

used to model faulty or even broken controlling and measurement devices. Furthermore, also the internal state of the controlled process may change due to uncontrolled features (`*plant noise´*). For simplification purposes we will, however, not regard the plant noise in the steam boiler example.

At a first glance, it may look artificial to construct such a detailed model of the environment if all we are interested in is the specification of the system controlling the steam boiler. Nevertheless, in sections 5.2, 5.3, and 5.4, it will become obvious that this detailed view of the environment will lead to a more concise and application adequate description of the steam boiler requirements. In particular, it will lead to a requirements specification independent of a certain implementation strategy.

Now, we relate the above scheme of a closed control loop to the steam boiler example. In order to do so, we have to instantiate

- the controller component of Figure 2 to the steam boiler controller of Figure 1 to be implemented,
- the plant or physical process of Figure 2 to the steam boiler facilities (like the steam boiler, the pumps, the valve, and the measurement devices) of Figure 1.

Furthermore, we have to give a detailed description of

- the virtual component adding noise to the control signal sent from the controller to the pumps and the valve,
- the virtual component adding noise to the sensory signal send from measurement devices for the water level, water and steam through-put, and pump state to the controller.

Since those components are virtual they have no real counterpart in the physical steam boiler system. Besides mapping the system components, we also have to define the messages exchanged between those components. Since our approach abstracts somewhat from specific features of the steam boiler (as, for instance, done in [Abr96]) like the number of pumps, we will only introduce the following messages:

- **from the controller to the physical system**: this signal is a compound signal with the attributes
  - *pump control signal,*
  - *valve control signal.*
- **from the controller to the operator desk**: here, only a simple signal is sent, indicating which state the controller or system is in; in our first approach, we only identify the states
  - *wait,*
  - *init,*
  - *normal,*
  - *emergency.*

- **from the physical system to the controller**: this signal is a compound signal with the attributes:
  - *water level,*
  - *steam throughput,*
  - *water throughput,*
  - *pump-state.*
- **from the operator desk to the controller**: here, only a simple signal is sent, indicating the issuing of a command from the operator desk; in our first approach, only the commands
  - *init,*
  - *stop,*
  - *reset*

  are identified.

In fact, in our first approach to the steam boiler example we even abstract from the detailed description of the structure of both measurement signal and control signal. Since their structure does not have any impact on the specification on the most abstract level, we will leave out this fine-grained structure, and only make use of the description of the operator command signals and the status report signal.

3.2 Using the Framework of Stream Processing Functions

As a formal basis for our approach we use stream processing functions as introduced originally in [Kah74]. Since for the understanding of this article only some basics are needed, we only give a short introduction of the elementary concepts as far as they do concern the case study. For a more complete introduction the reader is referred to the literature mentioned in section 1. Given the model of stream processing functions we apply it to the above-introduced section of control theory by mapping its terms on corresponding mathematical concepts.

3.2.1 A Short Introduction

Stream processing functions are used to model components performing computations and communicating with their environment by exchanging messages using directed and unbounded buffering channels[2]. To describe a component, we therefore note which - possibly infinite - sequence of incoming (received) messages leads to which - possibly infinite - sequence of outgoing (sent) messages. If a component has more than one channel to receive or send messages, we use tuples of such sequences. Thus, a stream processing function can be seen as a function that maps sequences of messages received on its input channels onto sequences of messages sent on its output channels.

---

[2] This form of communication is often referred to as „message asynchronous“ communication.

Nondeterministic behavior will be modeled by describing a component not only by one stream processing function but by a set of stream processing functions describing all the possible instances of its nondeterministic behavior.

Given a set of messages M, the set M^{ω} of streams over these messages is defined to be the set of all finite or infinite sequences over M. Furthermore, the following two elementary functions for the construction of streams will be defined:

- $\langle\rangle$: the empty stream, that is, the stream that contains no message,
- *m & s* : the stream that has m as its first element and continued by stream s.

Together with these constructor functions we define a few selector functions to be used in the upcoming formulas:

- *first*: the function selecting the first element of a stream, which is defined by[3]

$$first(\langle\rangle) = \bot \wedge first(m \,\&\, s) = m$$

- *rest*: the function selecting the rest of a stream chopping of the first element, which is defined by

$$rest(\langle\rangle) = \langle\rangle \wedge rest(m \,\&\, s) = s$$

- *rest(i,s):* the function selecting the i-th rest of a stream which is defined by

$$rest(0,s) = s \wedge rest(i+1,s) = rest(i,rest(s))$$

On the basis of streams we can now define the model for a component receiving messages on *m* channels with each input channel *j* transporting messages of type I_j, and sending messages on *n* channels with each output channel *k* transporting messages of type O_k. The model of such a component is a subset of the set of all prefix monotonous and continuous functions from $(I_1^{\omega} \times ... \times I_m^{\omega})$ to $(O_1^{\omega} \times ... \times O_n^{\omega})$. Their functionality is written as follows[4]

$$(I_1^{\omega} \times ... \times I_m^{\omega}) \rightarrow (O_1^{\omega} \times ... \times O_n^{\omega})$$

3.2.2 Mapping Terms

To conclude this section we show how to represent the model of control theory by the concepts of FOCUS. In order to do so, we will map the concepts introduced in section 3.1 on the concepts of 3.2.1. In Figure 3 we give a functional view of the steam boiler system by adding the necessary details to Figure 2.

---

[3] Here, $\bot$ denotes the undefined element.

[4] We use "$\times$" to denote the Cartesian product.

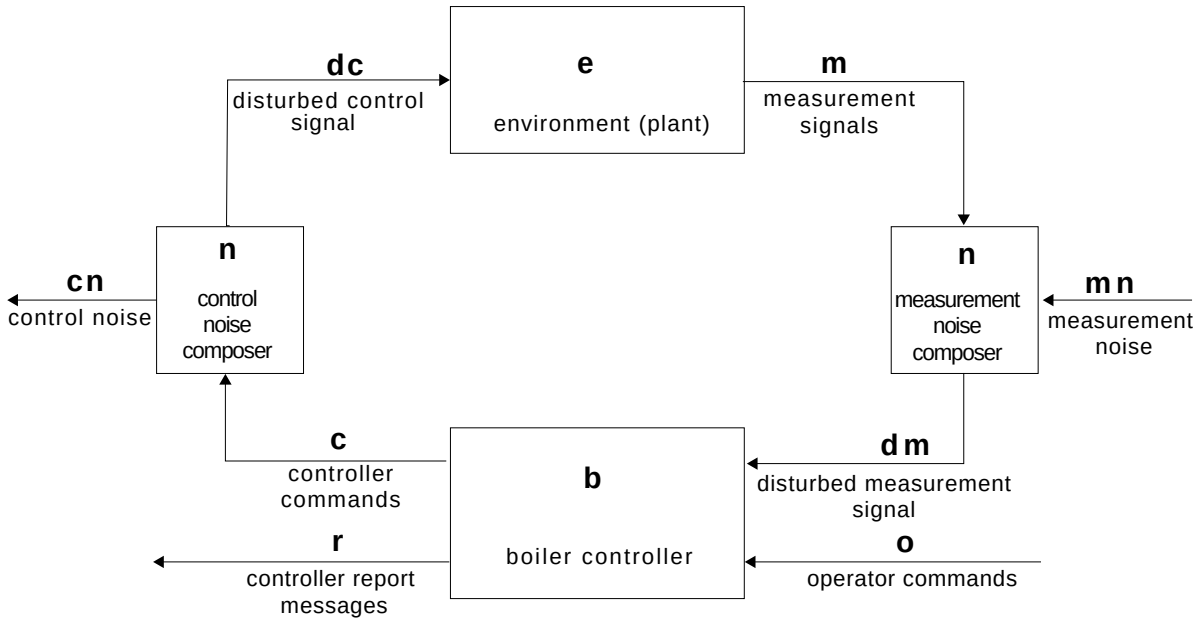

Figure 3: The functional system view

Before instantiating the functional units of the schematic control theory system (including the controller unit, the physical system, and the virtual components to add noise to the system), we have to define the message types used in the up-coming function specifications. As mentioned above, in the first approach we can ignore some details, and therefore define

- **C** to be the set of control commands issued by the controller, as well as received by the actuators of the physical unit, with no further information about the structure,
- **M** to be the set of measurement signals issued by the sensory units of the physical system, as well as received by the controller, with no further information about the structure,
- **O** to be the set of commands issued by the operator desk,
- **R** to be the set of commands issued by the controller to indicate the controller status,
- **C Noise** and **M Noise** to be the set of noise values added to control or measurement signals by the virtual components.

Based on these sets of messages, we can now define the functional components of the steam boiler system:

- The **boiler controller** is modeled by a set of functions that receive measurement signals from the sensory units and commands from the control desk and send control signals to the physical system plus status report signals to the operator desk. Thus we define their functionality by

$$b: (M^{\omega} \times O^{\omega}) \rightarrow (C^{\omega} \times R^{\omega})$$

- The **physical environment** is mapped onto a set of functions that receive control command messages from the controller and produce measurement data. Physics in

general only provide a model that relates current systems values (like water level, water- and steam through-put) and controlling input (like activation of the pumps and valves) with the system values of the next step[5]; using the above types, we therefore use functions *es* (**e**nvironment **s**tep functions) of the functionality[6]

$$es: M \times C \rightarrow M$$

where the system state corresponds to C. To lift this step-wise simulation of the environment up to a function continuously producing output receiving input, we have to embed this view in the stream-based approach which leads to functions *e* (**e**nvironment functions) of functionality

$$e: C^{\omega} \rightarrow M^{\omega}$$

This can be achieved using the following scheme of equations:

$$e(cs) = es\_lift(i,cs)$$

$$\textbf{\textit{where}}\ es\_lift(s,c\ \&\ cs) = es(s,c)\ \&\ es\_lift(es(s,c),cs)$$

Here, $i \in M$ is a possible initial physical state the environment can be in, and *es* a function from the set given above. Thus, the model of the environment is obtained by continuously applying the stepwise model *es* to the actual state and input, producing the new actual state and reporting this state as observable values.

- The **virtual components adding noise** to the control and measurement signals are mapped to stream processing functions from signals and noise to signals:

$$(C^{\omega} \times (C\ Noise)^{\omega}) \rightarrow C^{\omega}$$

and

$$(M^{\omega} \times (M\ Noise)^{\omega}) \rightarrow M^{\omega}$$

respectively. In both cases, we will write *n* for such a **n**oise-adding function.

This completes the mathematical scheme to define the meaning for our system components. Thus, all necessary terms introduced in the control theory section are translated into the FOCUS framework by giving appropriate function types. However, no behavior is assigned to those functions so far. In the following section we use tables as a user-friendly way to formulize the requirements for the steam boiler controller.

4 Streams, States, and Tables

As seen in section 3.2.1, stream processing functions provide a powerful and sophisticated framework. For the unfamiliar user it might sometimes appear quite complicated. To im-

---

[5] Since, as generally done with embedded systems, we use a discrete model of time, we can define the next step of the system execution to be the state at the next discrete time step.

[6] Since it might be impossible to obtain a realistic mathematical model of the physical environment using only the state information in C, a more detailed state might be necessary. Such a state might not only consist of the actual values but also the derivates of steam and water throughput or water level.

prove the comprehensibility and the acceptance, it is necessary to offer presentations that are at the same time precise and easily accessible by the non-formalist. Thus, in section 4.1 we introduce the notion of a state as a means to structure the specification and offer a tabular notation for state transitions.

4.1 Conceptual and Concrete States

The use of states for specification purposes comes in two different flavors, *conceptual states* and *concrete states.* By the term "conceptual state" we characterize those states which are used to structure the *specification,* often called control states. They arise from the requirements specification by "clustering" situations that show similar behavior. Conceptual states are *not* understood as concrete states. Concrete states, on the other hand, are implementation states. In particular, these states have to be mapped to a realization using variables of various types. Concrete states are also often called data states.

Depending on the notion of conceptual or concrete states, two different forms of specifications can be characterized:

- specification of state-dependent stream processing functions,
- state-based specification of stream processing functions.

The first term characterizes stream processing functions which explicitly use states as arguments together with stream tuples. Formally, those functions can be characterized using predicates of the form

$$(S \rightarrow ((I_1^\omega \times ... \times I_m^\omega) \rightarrow (O_1^\omega \times ... \times O_n^\omega))) \rightarrow \mathbb{B}$$

with S being a set of states, $I_1,...,I_m$ sets of input messages, $O_1,...,O_n$ sets of output messages, and $\mathbb{B}$ the set of Boolean values. This class of specifications is discussed in detail, for instance, in [DW92], [Spi94] or [Den95]. Since this class, however, uses concrete states, these specifications require the implementation to be a certain state-based one. Since we are interested in more abstract specifications here, in the following we will use the second class of specifications.

The second class characterizes those specifications, which are themselves parameterized with states, and characterize stream processing functions that are not necessarily state-based. Formally, this class can be described by the set of functions described by a predicate of the form

$$S \rightarrow (((I_1^\omega \times ... \times I_m^\omega) \rightarrow (O_1^\omega \times ... \times O_n^\omega)) \rightarrow \mathbb{B})$$

using the above set identifiers. This form of specification is also used in [Den95].

4.2 Writing Tables

In the introduction we have argued that the formal specification of the steam boiler system can be written in an easily readable and understandable but still precise manner by tables. In

the following we describe how we use the tables in our specifications and which entries we allow in the tables.[7]

A table consists of rows and columns in which the information is inscribed. Each table has a heading-line separated from the rest. Each entry in the heading-line encloses a short information about the column below and in which context the reader should interpret the column-entry.

The tables that we use in the steam boiler specification describe transitions. They consist of the following columns:

- one column (with the header “From“) for the actual state of the component before the transition,
- one column-block (with the header “Input”) for the messages which are actually readable at the input-channels,
- one column-block (with the header “Output”) for the messages which will be written to the output-channels as a reaction to the read input-messages, and
- one column (with the header “To”) for the state of the component after the transition.

For table entries we allow both single values for the states or the messages and sets of those values. Finally, we even allow using values for indexed entries, describing whole classes of entries. The parameters used are described following the header marked by a “with“ statement.

For the understanding of our tables it is essential that we read the input-messages in the actual state simultaneously from all input channels. We then write all output-messages simultaneously to the output-channels and change into the following state. In the presented steam boiler specification we use one table written in the form described above to specify the whole behavior of the controller component in all states to be specified at high level of abstraction. Once the table gets too large to comprehend, it is split up in tables for each state (as also, e.g., in [Spi94]).

We use the following scheme for the above introduced table form; here, the column-blocks are distinguished by different shades of grey.

From	Input	Output	To
s1	i11	o11	s2
s1	i12	o12	s3
s2	i21	o21	s1
s2	i22	o22	s3

Figure 4: Example Table

[7] The use of tables as formal description technique has been discussed using many different approaches; see, e.g., [Jan93] , [Par92], or [Bro98].

4.2.1 Formalizing Tables

The formalization of tables, a quite straightforward task, has been carried out in several semantical frameworks (see, e.g., [Jan93], [Par92], or [Bro98]). Using stream processing functions as our formal basis, we will give a short formalization of tables fitting in our framework.

The basic intuition underlying the formalization is to interpret each state s by a predicate P_s characterizing the stream-processing functions modeling the input output behavior of the component in the state s. In order to do so, each table entry of the form

From	Input	Output	To
s1	in	out	s2

is translated into the predicate

$$\exists f'.\ (P_{s2}(f') \wedge f(in \ \& \ is) = out \ \& \ f'(is))$$

stating the fact, that

- given a function *f*,
- which is receiving *in* as next input, then
- the characterized function will output *out* as next output,
- and then behave like a function *f'* being in state *s2*.

For the translation of the complete table, all of the predicates generated above associated to one particular state are grouped by conjunction. Here we assume that i11 ≠ i12, that is, the transitions can be distinguished by the input messages. Thus, for the table depicted in Figure 4 we obtain the following formalization:

$$P_{s1}(f) = \forall is.\ (\exists f'.\ (P_{s2}(f') \wedge f(i11 \ \& \ is) = o11 \ \& \ f'(is)) \wedge \exists f'.\ (P_{s3}(f') \wedge f(i12 \ \& \ is) = o12 \ \& \ f'(is)))$$

$$P_{s2}(f) = \forall is.\ (\exists f'.\ (P_{s1}(f') \wedge f(i21 \ \& \ is) = o21 \ \& \ f'(is)) \wedge \exists f'.\ (P_{s3}(f') \wedge f(i22 \ \& \ is) = o22 \ \& \ f'(is)))$$

So far we have explained the formalization of tables using single values as legal entries in the input and output message columns. Since in the upcoming example we will make use of the above mentioned set-valued entries, this translation scheme has to be extended. This adjustment is simply achieved by replacing the definition of the input and output variable *i* and *o*. This transformation is demonstrated in the following formalization:

$$(i \in IN \Rightarrow \exists f', o\ .\ (P_{s2}(f') \wedge o \in OUT \wedge f(i \ \& \ is) = o \ \& \ f'(is)))$$

The corresponding table entry for this predicate is

From	Input	Output	To
s1	IN	OUT	s2

with IN and OUT denoting sets of values.

Finally, we can use parameterized entries; this is done using a free variable, which is defined in the header block of the table. Here each entry is translated into a parameter header; the corresponding predicate is obtained by quantifying over the free variable according to its definition in the header block. Thus, for the table

From	Input	Output	To
s1	E11(p)	A11(p)	s2
where p ∈ P			

we give the translation

$$P_{s1}(f) = \forall p \in P$$
$$\forall i, is. ((i \in E11(p) \Rightarrow \exists f', o.$$
$$(P_{s2}(f') \wedge o \in A11(p) \wedge f(i \,\&\, is) = o \,\&\, f'(is)))$$

As usual, the free variable p in the table is translated to universal quantification. This table scheme is used, for instance, section 5.3.

5 Approaching the Steam Boiler

After the short introduction in the concepts of tables, we now formalize the requirements of the steam boiler controller to demonstrate the various facets of our approach. We present the translation between tables and formulas using the example of the steam boiler. We show that the table technique leads to a more readable form than the predicate logic formulas with stream processing functions.

5.1 Steam Boiler States

To specify the requirements of the steam boiler controller we will use a state transition approach. As mentioned before, we will use conceptual states to structure the specification by clustering similar behavior. This will simplify the approach in two respects:

- A major percentage of the requirements of the steam boiler controller can be described as simple transitions between the introduced conceptual states; thus, using an appropriate tabular or graphical representation, the major part of the functionality of the controller can be easily described.
- The refinement of this first requirements specification can be achieved analogously by the introduction of new states; thus, even the refinement of the requirements specification can be performed in a structured way.

To be able to demonstrate how refinement is performed in our approach, the first requirements specification will leave out the treatment of broken sensors. In the first approach, we will therefore identify the following four different states:

1. The **wait** state: The state the system is in after start-up, ready to receive the command to initialize the steam boiler.
2. The **init**ialization state: The state the system is in during the initialization phase, when the controller is getting ready for the normal operation mode.

3. The **normal** state: The state the system is in if all components are working properly.
4. The **emergency** state: The state the system is in whenever it becomes uncontrollable, and control is handed over to the operator desk.

As mentioned above, these states are conceptual states; in particular, these states have to be mapped onto concrete states for an implementation leading to a more complex state space where each state consists of a collection of values recorded from the sensory input signals. An efficient implementation of such a state space might collect relevant information like water level, steam and water throughput, and the corresponding derivatives, as well as the pump and valve state. Figure 5 gives an overview of the above-described states of the steam boiler controller.[8]

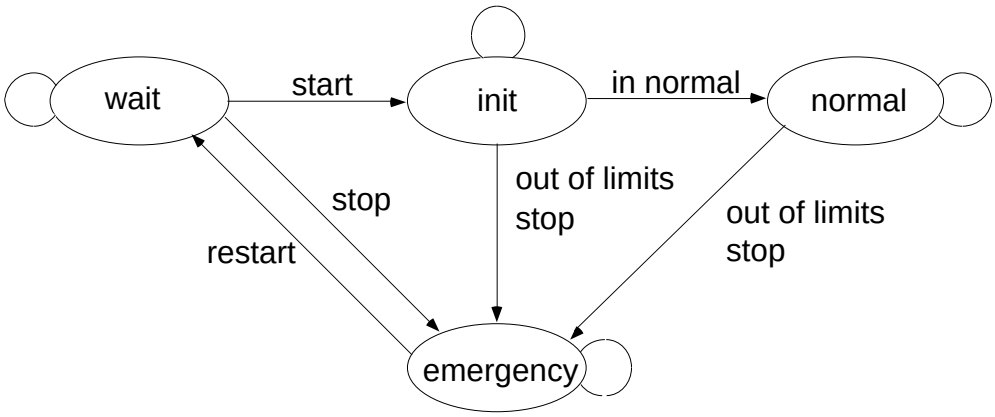

Figure 5: The Steam Boiler States

The figure relates the states by denoting the transitions between the states with input events. Those input events characterize sets of sensory input signals that will cause the controller to change to the corresponding state. The formal characterization is given in section 5.2.

5.2 Formalizing Transitions

In the following table we will formalize a set of simple requirements that concern the step-wise behavior of the controller. We will start with the description of the initial wait state of the controller:

Requirement 1: Being in the state `wait´, the controller will change to the state `init´ given a start command.

Requirement 2: Being in the state `*wait*´, the controller will change to the state `emergency´ given a stop command.

Requirement 3: Being in the state `wait´, the controller will stay in the state `wait´ given no command.

---

[8] This form of graphical representation of state transition diagrams is, for instance, used in [HMS98].

The formalization of these requirements is straightforward and does not need any further explanation:

From	Operator	Measure	Report	Control	To
wait	start	M	init	C	init
wait	stop	M	emergency	C	emergency
wait	o where o ≠ start ∧ o ≠ stop	M	wait	C	wait

Here and in the following tables, we use

- o for an arbitrary operator command,
- m for an arbitrary measured value out of M,
- c for an arbitrary control value out of C.

Of course, the set of possible input messages can be described more implicitly as done in the previous requirement. This is demonstrated in the description of some requirements of the init state. The corresponding requirements are:

Requirement 4: Beginning in the state `init´, the controller will change to the state `emergency´ given a stop command.

Requirement 5: Beginning in the state `init´ and receiving measurement input corresponding to a normal system state recorded without noise, the controller will change to state `normal´ and report this change if not aborted by an operator signal.

Requirement *4* is just the corresponding rephrasing of requirement 2 and needs no further explanation. Requirement *5* however, makes use of a more implicit characterization of the set of possible input values.

From	Operator	Measure	Report	Control	To
init	stop	M	emergency	C	emergency
init	o where o ≠ stop	m where ∃ s mn. norm(s) ∧ ok(mn) ∧ m = n(s, mn)	normal	C	normal

Here, we made use of the following auxiliary functions:

- The boolean function *norm: M → T* checks whether the state of the controlled system (the system parameters) are within the normal limits, thus describing a legal system state.

- The Boolean function *ok: M Noise* $\rightarrow$ *T* checks whether no noise is added to the sensory data recorded by the measurement.

It should be noted that the table and thus the corresponding specification are not complete in the sense that many possible situations are not covered that may occur during the execution of the controller. Nevertheless, the underlying semantics makes very clear what the meaning of such an under-specified situation is:

> *If a reaction in a possible situation is not explicitly determined, since the table contains no restriction on the described system, every possible behavior is legal.*

This follows quite naturally from the formalization.

5.3 Complex Transitions

Not all properties of the steam boiler controller are simple state transitions according to the above scheme. Let us have a look at the following property:

> *Requirement 6: If every possible output that may be produced by the controller will take the system out of the limits in absence of noise, then the controller must indicate this by producing `emergency´ and changing into state `emergency´.*

At first glance, this requirement does not look formalizable according the above-introduced scheme. Nevertheless, even those requirements can be formalized analogously. In order to do so, the point of view must be slightly changed. While the specification in [RSB95] used a global system view, the table specification is always restricted to the interface of the controller itself. Nevertheless, this does not restrict the formalization of the above requirement, if the requirement is restated in an appropriate way, restricting it to inputs and outputs of the controller:

> *Requirement 6': If the measurement was produced by a noiseless recording of a legal system state, which will be changed into an illegal state by any possible noiseless manipulating controller output, then the controller must produce an arbitrary control signal as well as an `emergency´ report and then change to the `emergency´ state.*

Formally, this set of input values can be described by:

$$\{ m \mid \exists\, s\ mn.\ \forall\, c\ cn.\ norm(s) \wedge ok(mn) \wedge m = n(s,mn) \wedge ok(cn) \Rightarrow \neg norm(es(s,n(c,cn))) \}$$

expressing the fact that

- on the one hand the measurement message is obtained by adding no noise to an acceptable system state, while
- on the other hand from this system state each control signal with no noise added will take the physical system to a state that is no longer acceptable.

Thus, we get the following table entry:

From	Operator	Measure	Report	Control	To
normal	O	m where ∃ s mn.∀ c cn. norm(s) ∧ ok(mn) ∧ m = n(s,mn) ∧ ok(cn) ⇒ ¬norm(es(s,n(c,cn)))	emergency	C	emergency

Another requirement, which is of similar complexity, concerning the set of possible output values is the following property:

Requirement 7: If there is an output, which will keep the system within the limit, and there is no `stop´-signal issued by the operator, and there is no noise, then the controller has to produce such a manipulation signal, report a `normal´ state and remain in the state `normal´.

According to the above case, Requirement 7 is restated in a way suitable for a table representation:

Requirement 7': If no `stop´ signal is issued by the operator, and the input is of a kind resulting from a measurement of a legal system state without noise, and the system can be kept in the legal limits by a noiseless manipulating signal, then the controller has to produce such a signal, report a `normal´ situation and remain in the state `normal ´.

Here, besides the characterization of the set of possible input values

$$\{ m \mid \exists\, s\ mn\ cs\ c.\ norm(s) \wedge ok(mn) \wedge m = n(s,mn) \wedge ok(cn) \wedge norm(es(s,n(c,cn)))\}$$

also the set of possible output values

$$\{ c \mid \exists\, s\ m\ mn\ cn.\ norm(s) \wedge ok(mn) \wedge m = n(s,n) \wedge ok(cn) \wedge norm(es(s,n(c,cn)))\}$$

must be characterized appropriately.

Thus, we obtain the following table entry:

From	Operator	Measure	Report	Control	To
normal	o where o ≠ stop	m where ∃ s mn cs c. norm(s) ∧ ok(mn) ∧ m = n(s,mn) ∧ ok(cn) ∧ norm(es(s,n(c,cn)))}	normal	c where ∃ cn. ok(cn) ∧ norm(es(s,n(c,cn)))	normal
where s ∈ S ∧ norm(s)					

Note that we used an entry parameterized by the environment state *s* because the control output *c* does depend on the state *s* in form of the received measurement. In the above cases, the state environment state *s* could be defined locally, since the control output did not depend on *s* but was chosen arbitrarily.

5.4 Global Properties

Besides requirements that basically consist of simple and complex state transitions there is a third class which cannot be formulated in such a table framework. This is due to the fact that these requirements do not describe properties of the steam boiler controller that can be formalized using one computation step. One of these properties is the description of the initialization phase:

Requirement 8: During the initialization phase the controller issues a sequence of manipulating values that will eventually lead the physical system into a normal state in case there is no noise.

It is remarkable that - again - this requirement can be stated without mentioning any possible realization of this sequence of initializing sequences. It can even be phrased in such an abstract way that it does not only apply to the steam boiler example, but to any similar control task. Here, too, the formal phrasing is quite straightforward:

$$
\begin{aligned}
&ok^*(mns) \wedge ok^*(cns) \wedge nostop^*(hsin) \Rightarrow \\
&\quad \exists\, m\ n\ c\ hsout. \\
&\qquad (c, hsout) = b(mn(mns, m), hsin) \wedge \\
&\qquad m = e(cn(cns, c)) \wedge \\
&\qquad ok(nth(n,c)))
\end{aligned}
$$

Here we made use of some auxiliary functions:

- The Boolean function $ok^*: (M\ Noise)^\omega \rightarrow T$ checks whether the sequence of noise values added will leave the signals undisturbed. This function is defined to be

$$ok^*(mn) = \forall n.ok(nth(n, mn))$$

- The Boolean function $nostop^*: O^\omega \rightarrow T$ checks whether the operator command sequence does not contain any stop command.

$$nostop^*(o) = \forall n.nth(n, o) \neq stop$$

Both definitions make use of the function $nth : \surd \times A^\omega \longrightarrow A$, selecting the n-th element of a stream if such an element exists. For a formal definition of *nth* see [Reg94].

This property, in contrast to the above introduced requirements, is a pure liveness property. Unlike the other properties, which are safety properties, it cannot be expressed by properties of the state transitions in a stepwise computation. Since these global requirements are not stated in the tabular notation as given before, we have to define how these two forms are formally combined. According to the definition of the formalization of table entries as given in section 0 each entry is translated into a predicate with the function co modeling the controller as its free variable. Since the above formalization of the global requirement is of the same form, the combination is straightforward: it just has to be combined by conjunction with the predicates obtained by the translations of the table entries.

6 Refinement of Data and States

As pointed out in [Bro94] or [Bro93a], one of the major advantages of the model of stream processing functions is its modular refinement concept. It offers several notions of refinement like:

- *Behavioral Refinement*: Refining the behavior of a system by adding more properties to its specification, restricting behavior in cases that were unspecified before.
- *Glass Box Refinement*: Refining a system by breaking it up into components of communicating components (structural refinement) or by giving a concrete state transition system (state-based refinement).
- *Interface Refinement*: Refining the interface that consists of the input and output channels of a component and the message sets communicated on them, by breaking up messages that have been atomic before.

In the following we give a short introduction to the basic idea of refinement in the context of stream processing functions. Then, we demonstrate how the refinement of data and states is carried out within the framework of stream processing functions. Finally, we will apply these concepts to the above introduced tables and use the steam boiler controller as an example to demonstrate the techniques.

6.1 General Refinement Concepts

The central purpose of refinement is the adding of implementation decisions and the modeling of additional system details. In the steam boiler approach two forms of refinement might be applied:

- **Data Refinement**: This refinement technique describes the successive elaboration of details in the process of system modeling. In the steam boiler context, we can decide that the sensory input of the controller consists of measurements of the water level, water and steam trough-put or the pump state. A further refinement might be to split up the pump into three pumps, requiring to split up the pump control and the pump measurement system into three streams of values.
- **State/Transition Refinement**: This form of refinement describes the successive elaboration of behavioral details and thus the elimination of under-specification. In the steam boiler context, we might consider only behavior in case of undisturbed control and measurement signals, leading to simple states like **wait**, **init**, **normal** and **emergency**. In a second step we might then add requirements for slightly disturbed signals introducing a **degraded** mode.

Since refinement, in general, is a formal concept for specification manipulation, the basic concept behind every refinement notion should be as simple as possible to allow for sufficiently manageable proofs. Therefore we do not only introduce and demonstrate the above used refinement concepts, but also rephrase them in the setting of tabular specifications. Since, as mentioned above, the tabular notation is somewhat limited in its expressiveness, not all properties can be described that way. We therefore give a short outline about the applicability of this technique.

6.2 Data and State Refinement

In the context of the steam boiler we will demonstrate the refinement technique using both data and state refinement. The corresponding setting will be the specification of the behavior in a case where the measurement units fail. Again, we see here our task in the specification of the requirements and not in the description of an implementation strategy.

6.2.1 Refinement of Messages

Since we now want to talk about the potential failures of the water level measurement device we explicitly have to model this part of the sensory input to the controller. Thus, the set of input messages to the controller will be modeled by the set

$$M = W \times M'$$

obtained by the Cartesian product of W and M', where

- **W** is the set of possible values of the water measurement signal, and
- **M'** is the set of possible values of the remaining sensory signals, again with further details ignored.

Of course, also the corresponding noise has to be adapted defining

$$M\ Noise = W\ Noise \times M'\ Noise$$

Together with the refinement of the message types we have to give a refinement of the appropriate functions operating on these messages. Here, we just informally introduce these definitions which are necessary for the understanding of the following requirements

- The boolean function *noisy: M´ Noise* $\rightarrow \mathbb{B}$ checks whether the noise added to the measured sensory data is significant.
- The boolean function *ok: W Noise* $\rightarrow \mathbb{B}$ checks whether no noise is added to the sensory data recorded by the water level measurement unit.
- The boolean function *ok: M´ Noise* $\rightarrow \mathbb{B}$ checks whether no noise is added to the sensory data recorded by the measurement units other than the water level unit.

Furthermore, to have a reasonable definition of absence of noise we have to define

$$ok((wn, mn')) = ok(wn) \wedge ok(mn')$$

for all corresponding values of water level noise wn and all values of noise mn´ of the remaining units.

6.2.2 Refinement of States

After refining the messages, we are now ready of expressing the intended refinement of the controller's behavior. The requirements specification introduced so far does not deal with the case of device failure. It will therefore be extended to describe a legal behavior in case the water level measurement device and control devices are unbroken. The behavioral restriction is expressed in the following four requirements:

Requirement 9: If the measurement signal is distorted by significant noise, but the water level measurement and the controlling signal are undisturbed, if there is no `stop´ - signal issued, and there is a control signal which keeps the system in the limits under these circumstances, then a controller in normal mode will produce such a signal, report the `degraded´ mode, and switch to `degraded´ mode.

Requirement 10: If the measurement signal is distorted by significant noise, but the water level measurement and the controlling signal are undisturbed, if there is no `stop´-signal issued, and there is a control signal which keeps the system in the limits under these circumstances, then a controller in degraded mode will produce such a signal, report the `degraded´ mode, and remain in `degraded mode´.

Requirement 11: If any possible noiseless control signal will take the system out of the limits, the controller in `degraded´ state must produce an arbitrary control signal as well as an `emergency´ report and then switch to `emergency´ mode.

Requirement 12: If the control signal remains undisturbed, the system is in the limits, the water measurement devices is unbroken, and the controller reports `degraded´ state, the system is kept in the limits by the control signal.

The tabular descriptions of Requirement 9, Requirement 10, Requirement 11, and Requirement 12 correspond to the tabular description of

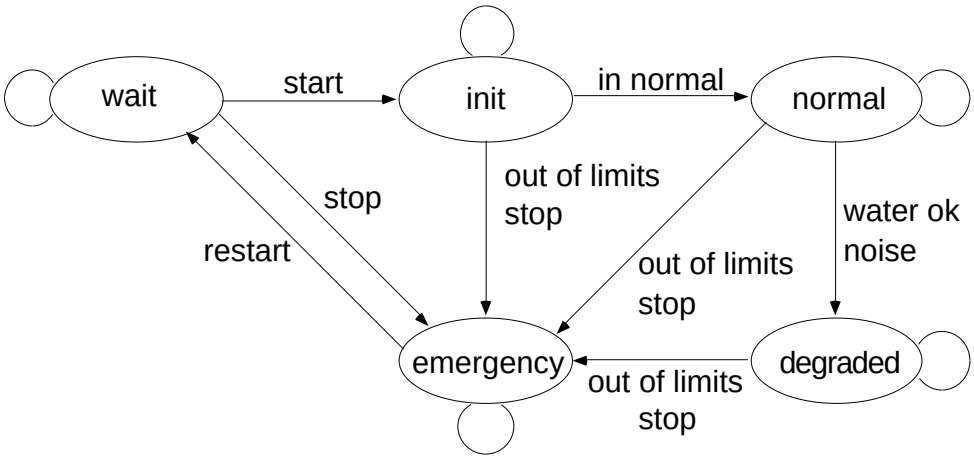

Figure 6: Refined States

6.3 Refining Tables

As shown in 6.2, refining a state based specification of a component by refining its messages or its conceptual states is quite straight-forward, but somewhat technical. Nevertheless, using these techniques in a restricted context, like tables, turns them into a much simpler approach.

The addition of some new input messages gives a more detailed view of a component; so the specification must enclose the reactions of the component to these new input messages and the table must be expanded with these new cases. The reaction of the component to the

new input messages will be described with possibly new output messages as well as new states in the “to”-column. Because the table specification should always give a complete specification of the behavior according to the actual abstraction degree we must complete the table specification with the new actual states in the “from”-column and their postulated behavior.

In the following table we show these extensions of the steam boiler specifications. The reader will see that the elimination of under-specifications in the development of a system description with different abstraction degrees can be written as a simple extension of the tables with the new states. Therefore each refinement step gives a more complete version of the table. The underlying semantical framework stays the same as we have described in the previous chapters.

From	**Operator**	**Measure**	**Report**	**Control**	**To**
normal	o where o ≠ stop	m where ∃ wn mn´ c cn. ok(wn) ∧ ok(cn) ∧ noisy(mn´) ∧ m = n(s,(wn,mn´) ∧ norm(es(s,cn(c,cn)))	degraded	c where ∃ cn. ok(cn) ∧ norm(es(s,cn(c,cn)))	degraded
degraded	o where o ≠ stop	m where ∃ wn mn´ c cn. ok(wn) ∧ ok(cn) ∧ noisy(mn´) ∧ m = n(s,(wn,mn´) ∧ norm(es(s,cn(c,cn)))	degraded	c where ∃ cn. ok(cn) ∧ norm(es(s,cn(c,cn)))	degraded
degraded	O	m where ∃ wn mn´ cn. ok(wn) ∧ ok(cn) ∧ noisy(mn´) ∧ m = n(s,(wn,mn´) ∧ ∀c.¬norm(es(s,cn(c,cn)))	degraded	C	degraded
degraded	stop	M	emergency	C	emergency
where s ∈ S ∧ norm(s)					

7 Conclusion and Outlook

In the presented paper we gave an outline on how different aspects of the steam boiler example can dealt with main concepts of the FOCUS method. The treatment of the steam boiler example as given above is by no means complete. It rather concentrates on the modeling of the control task in FOCUS, on the representation of the specification by tables and on the stepwise refinement of the specification than treating all the properties in detail, thus modeling only some facets of the steam boiler.

The aim of the article is not to give a complete specification of the steam boiler controller with all its states and features, as it is done in [ML96] or [LL96]. Instead, we tried to show how FOCUS offers a methodological approach for the development of an implementation of a control task beginning at a very high level of description and ending at the level of abstract implementation. The case study has shown several promising aspects. For a full evaluation

of the usefulness, however, the steam boiler example has to be developed to a more detailed degree, and a comparison to the other approaches has to be drawn.

Many aspects are not treated at all. Two of the most important aspects not treated here in sufficient detail are: the development of the failure model (which means mainly a design of a sophisticated state space), and the methodical development of state-transition systems using table-based techniques. See, for instance, [Bro98] for a more elaborated treatment of tables.

Additional contributions of FOCUS supporting the development of applications in the control theory domain are, for instance, ANDL (**A**gent and **N**etwork **D**escription **L**anguage see [SS95]), a formal syntax for the FOCUS specification language, a formal semantics based on HOLCF (see [Reg94]), the extension of *Isabelle*'s (see [Pau94]). HOL to LCF, as well as a translation of ANDL to HOLCF. Furthermore, a tool prototype for a development on graphical and state-based description techniques, called AUTOFOCUS (see, for example, [HSSS96], [HSE97], or [HMS98]), is implemented. Therefore, it might be interesting to rephrase the steam boiler case study in this context and to carry out several refinement steps using computer aided verification.

As a whole the major contribution of this paper lies in the control theoretical approach, which made the formalization of several overall aims of the controller surprisingly simple, sufficiently abstract from any possible implementation, and alike to the textual phrasing of the requirements. Using further case studies it should be investigated whether this scheme can be generally applied to problems in the control theoretical domain

Acknowledgement

It is a pleasure to thank Ursula Hinkel and Max Breitling for their helpful comments on earlier versions of this paper.

Literature

[Abr96] Abrial, J.-R. Steam boiler control specification problem. In: Jean-Raymond Abrial, J.-R., Börger, E., and Langmaack, H. (eds): *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*. Lecture Notes in Computer Science 1165, Springer-Verlag , 1996.

[Bro93a] Broy, M.: (Inter-)Action Refinement: The Easy Way. In: Broy, M. (ed*.): Program Design Calculi*. Springer NATO ASI Series, Series F: Computer and System Sciences, Vol. 118, pp. 121-158, Berlin, Heidelberg, New York: Springer 1993

[Bro93b] Broy, M.: Functional Specification of Time Sensitive Communicating Systems. ACM Transactions on Software Engineering and Methodology 2:1, January 1993, 1-46

[Bro94] Broy, M. Specification and Refinement of a Buffer of Length One. Working Material of Marktoberdorf Summer School. Technische Universität München. 1994

[Bro95] Broy, M.: Advanced Component Interface Specification. In: Takayasu Ito, Akinori Yonezawa (Eds.). *Theory and Practice of Parallel Programming*, International Workshop TPPP'94, Sendai, Japan, November 7-9, 1994, Proceedings, Lecture Notes in Computer Science 907, Springer 1995

[Bro98] Broy, M. Pragmatic and Formal Specification of System Properties by Tables. Technical Report Technische Universität München, TUM-I9802, 1998.

[BDD93] Broy, M., Dendorfer, C., Dederichs, F., Fuchs, M., Gritzner, T., Weber, R. *An Introduction to FOCUS*. Technical ReportTechnische Universität München. TUM-9202-2. 1993.

[Den95] Dendorfer, C. Methodik funktionaler Systementwicklung. Ph.D. Thesis. Technische Universität München, 1995.

[DW92] Dendorfer, C., Weber, R. Development and Implementation of a Communication Protocol - An Exercise in FOCUS, Technical Report, TUM-I9207. Technische Universität München, 1992.

[Fuc93] Fuchs, M. Funktionale Spezifikation einer Geschwindigkeitsregelung. Technical Report TUM-I9301. Technische Universität München. 1993

[Hin98a] Hinkel, U. Formale, semantische Fundierung und eine darauf abgestützte Verifikationsmethode für SDL. Ph.D Thesis. Technische Universität München, 1998.

[Hin98b] Hinkel, U. Home Shopping - Die Spezifikation einer Kommunikationsanwendung in FOCUS. Technical Report TUM-I9808. Technische Universität München, 1998.

[HMS98] Huber, F. Molterer, S. Schätz, B. Slotosch, O. Vilbig, A. Traffic Lights – An AUTOFOCUS Case Study. IN: 1998 International Conference on Application of Concurrency to System Design, pp. 282-294. IEEE Computer Society, 1998.

[HSE97] Huber, F. Schätz, B. Einert, G. Consistent Graphical Specification of Distributed Systems. In: Fitzgerald, J (ed.). FME´97, Lecture Notes in Computer Science 1313. Springer 1997.

[HSS96] Huber, F. Schätz, B. Schmidt, A. Spies, K. AUTOFOCUS-A Tool for Distributed System Specification. In: Bengt, J. Parrow, J. (eds.) Proceedings FTRTFT´96. Lecture Notes in Computer Science 1135, Springer 1996, pp. 476-470.

[Jan93] Janicki, R. Towards a Formal Semantics of Tables. Technical Report CRL 364. McMasters University. 1993.

[Kah74] Kahn, G. The Semantics for a Simple Language for Parallel Programming. In: Rosenfeld, J.L. (ed.) *Information Processing´74*. pp. 471-475. North Holland, 1974.

[KK84] Kaspers, W. Küfner, H.-J. *Messen, Steuern, Regeln*. Vieweg, Braunschweig. 1984.

[LL96] Leeb, G. Lynch, N. Proving Safety Properties of the Steam Boiler Controller. In: Jean-Raymond Abrial, J.-R., Börger, E., and Langmaack, H. (eds): *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*. Lecture Notes in Computer Science 1165, p. 318-338, Springer-Verlag , 1996.

[ML96] Merz, S. Lesske, F. Steam Boiler Control Specification Problem: A TLA Solution. In: Jean-Raymond Abrial, J.-R., Börger, E., and Langmaack, H. (eds): *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*. Lecture Notes in Computer Science 1165, p.339-358, Springer-Verlag , 1996.

[Par92] Parnas, D. Tabular Representation of Relations. Technical Report CRL 260. McMasters University. 1992.

[Pau94] Paulson, L. Isabelle - A Generic Theorem Prover. Lecture Notes in Computer Sience 828. Springer. 1994

[PS97] Philipps, J., Schmidt. A. Traffic Flow by Data Flow. Technical Report TUM-I9718. Technische Universität München, 1997.

[Reg94] Regensburger, F. HOLCF - Eine konservative Erweiterung von HOL um LCF. Ph.D. Thesis. Technische Universität München. 1994.

[RSB95] Regensburger, F., Schätz, B., Broy, M. A Functional Model of the Steam Boiler. Slides presented at the Dagstuhl Workshop "*Methods for Semantics and Specification*", 1995.

[SHB95] Schätz, B. Hußmann, H. Broy, M. Graphical Development of Consistent System Specifications. In: J. Woodcock, M.-C. Gaudel (eds), FME' 96, Lecture Notes in Computer Science 1051, Springer 1996, pp. 248-267.

[SS95] Schätz, B., Spies, K. Formale Syntax zur logischen Kernsprache der FOCUS Entwicklungsmethodik. Technical Report TUM-9529. Technische Universität München. 1995.

[Spi94] Spies, K. Funktionale Spezifikation eines Kommunikationsprotokolls. Technical Report TUM-I9414. Technische Universtiät München, 1994.

[Spi98] Spies, K. Eine Methode zur formalen Modellierung von Betriebssystemkonzepten. Ph.D. Thesis. Technical Report TUM-I9809. Technische Universität München, 1998.

[WT95] Wilder, A. Tucker, J. System Documentation Using Tables. Technical Report CRL-306. McMasters University. 1995