Formal Requirements Engineering Using Observer Models

Andreas Nonnengart, Georg Rock, and Werner Stephan

German Research Centre for Artificial Intelligence
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
`nonnenga,rock,stephan@dfki.de`

Abstract. Today we are confronted with an enormous variety of formal software engineering approaches and tools. Among these are many that address the critical early stages of software development. However, only little attention has been paid to the integration of different specialised approaches and to the overall development process.

In this paper we present a technique for formal requirements analysis (observer models) that deals with particular perspectives on a system rather than with particular aspects of it.

A realistic gasburner example illustrates the overall approach.

1 Introduction

Thirty years ago (in the early seventies of the last century) the question what are formal methods was easy to answer. However, code verification in Hoare style systems failed not only because of its inability to cope with complexity but also because this restricted approach did not meet important needs of software engineering. Meanwhile the situation has drastically changed. We are confronted with an enormous variety of formal approaches and tools. Among these are many that address the critical early stages of software development. Although much progress has been made with respect to fully automatic methods little attention has been paid to the integration of different specialised approaches and to the overall development process.

Formal techniques for requirements analysis often deal with a particular aspect of the system to be designed. Examples of such aspects are properties of information flow, correctness of (cryptographic) protocols, and real time behaviour. In this paper we concentrate on real time analysis as one view on a system among others. It is not difficult to imagine a system that separates different applications by controlling the flow of information between them using authentication protocols as one of the security mechanisms and that in addition has to satisfy certain real time requirements.

Although it is well known that many severe errors occur already in the early stages of the development process it is the case also for later design stages like architectural design and implementation that they have to be treated formally if one aims at high assurance levels. For example, according to level EAL5 of

Common Criteria (CC) [7], a formal high-level design and a “correspondence proof” with respect to the so-called functional specification has to be provided. In the case of real time systems one has to define *how* the intended global behaviour is realized by separating the control component from its environment and by making assumptions (delays, cycle time) explicit. This scenario is described in section 3. Therefore, a number of particular views for requirements analysis have to be linked to a single abstract systems specification (**System Design Spec.**) that serves as a starting point for the refinement process (see Figure 1).

Rather than having a **satisfies** relation between a specification and a collection of simple properties that have to be established, requirements analysis will be based on its own descriptions (views) and postulated properties that refer to these descriptions. The description of a particular view will not necessarily use the same terminology as the system specification and often application specific formalisms and tools will allow for an efficient analysis. For establishing information flow properties a technique called non-interference, which is based on closure properties of sets of (system) traces, has to be used [8]. The analysis of protocols is based on a different kind of traces [10] that include steps of an attacker. A number of tools, like for example the one described in [9], have been used in this area. The real time view can be implemented by Hybrid Automata [2] or by Timed Automata [3]. Again, tools like HyTech [4] provide efficient techniques to establish real-time properties.

In the following we present a general technique called *observer models* to link abstract descriptions of the real time behaviour of a system to a system specification consisting of a control component, an environment, and a clock by means of an *observer mapping*. After an outline of the general technique we shall illustrate our approach using the specification of a controller for a gasburner (together with an environment and a clock) and a Hybrid Automaton that describes the global realtime behaviour of the system.

2 Observer Models for Realtime Properties

Requirements of a system to be developed are specified and analysed by possibly several different formalisms that are specific for a particular view on that system. The choice of the formalisms can be influenced by several factors: preferences or expertise of the user, special features that need certain constructs of the specification language, system support, and the re-use of already existing specifications. In Figure 1 each view is represented by an **Observer-Spec_i** following its own description technique and formalism. One of these specifications might contain a global description of the runs of a protocol while another view concentrates on realtime properties. In the following we shall assume that the realtime view is given by Hybrid Automata [2].

As already mentioned above a view will also include properties, called **OS-Properties_i** in Figure 1, that have to be established from the observer specification. Realtime requirements can be formulated and proven using tools like HyTech [4]. Note that we consider Hybrid Automata as a kind of comprehensive

Fig. 1. Observer Models

description of the entire system behaviour with respect to time constraints. As can be seen from our example, the description is *global* in the sense that it does not distinguish between the control system and its environment. States of the Hybrid Automaton therefore do not directly correspond to internal states of the controller. They rather describe certain situations that might occur in a run of the three components (controller, environment, clock) together. To demonstrate this we will start an example scenario in section 3 using the specification of a gasburner controller.

To integrate various views into a common formal development the **Observer-Spec_i** and the **OS-Properties_i** first have to be translated into the language of the **System Design Spec.** The resulting specifications are called **OS-SD-Spec** and **OS-SD-Prop-Spec**. The translation of Hybrid Automata into the specification language of VSE is described in [11,12]. The language used in VSE[6,5] is VSE-SL. This is similar to TLA but with some extensions concerning among others the handling of shared variables and the handling of assumption commitment specifications.

It can be shown that the **satisfies** relation between **OS-SD-SPEC_i** and **OS-SD-Prop-Spec_i** holds, if and only if the **satisfies** relation holds between **Observer-Spec_i** and **OS-Properties_i**. First of all this means that results obtained by using a tool like HyTech can be safely integrated into the overall development. However, since the language of **OS-SD-Spec** and **OS-SD-Prop-Spec** is more expressive (than that of Hybrid Automata) requirements specifications that are still “in the style of Hybrid Automata” but more general than these can be used in this context if one is inclined to use deductive techniques instead

of model-checking. As an example one might wish to consider state transitions where one of the parameters (like speed) changes arbitrarily.

We are still left with the problem of establishing a link between the system specification and the particular realtime view we have defined. This is done by a mapping (called **Obs. Map_i** in Figure 1) that (in the case of realtime requirements) interprets a given state of the interleaved computation of the controller, environment, and clock as a state of the (translation of the) Hybrid Automaton. It thereby turns the entire controller scenario into a model (in the sense of Hybrid Automata). For this we need to be sure that the translation faithfully preserves the semantics of Hybrid Automata.

In the following we present an instantiation of the general methodology that uses an abstract and global specification of the realtime behaviour of a gasburner scenario by a Hybrid Automaton on the one side and the specification of a controller that is connected to an environment by sensors and actors on the other. We outline the proof, that the automaton really describes the behaviour of the technical scenario with respect to a given mapping.

3 General Specification Scheme for Observer Models

The general scenario (see Figure 2), which we shall instantiate using a realistic yet abstract gasburner specification consists of three components: an environment, a controller and an observer/clock component. Which role do the individual components play now? Generally, given a system design one cannot decide always accurately, which parts are to be assigned to the environment and which parts belong to the controller itself. In the application of formal methods we are often interested in the safety critical parts of the system to be developed. The other parts are considered to be irrelevant for the safety of the system. These parts could consist for example of monitoring units¹. The fact is substantial that the control, which is to be refined later or at least is applicable for a refinement, should contain all the safety critical parts.

The behaviour of the environment is determined by the specification of its interfaces. I.e. the environment has to supply the values needed on the various interfaces (in time). To guarantee the right functioning of the system in this case we have to make assumptions about the correct behaviour of the environment². These assumptions can be used in the proof of the postulated properties of the system. If the environment component does not only exist as an interface definition, but also as a component with accurately specified behaviour, then one can prove these assumptions about the environment using the behaviour of the environment. Of course the type, range, and depth of the specification of

¹ Of course this is not always a uncritical unit. Just think of flight control that show the pilot the actual status of the plane.

² If one is interested in fault-tolerant systems, the possible faults can be described in the specification of the environment, so that the control system must detect these and behave accordingly.

Fig. 2. General scenario (Observer model instantiated)

both the environment and controller depends on the properties that should be fulfilled.

The environment and the system/controller are specified as usual temporal logic specifications. Both components can be structured into subcomponents.

The specification of the observer/clock component differs in some sense from these usual specifications. One of the tasks of the observer is that it holds the time. But this fact does not influence the method described here.

The essential part of the observer is that it observes the system parts that are built from the Controller and the Environment components. These observations are filtered by the observer and communicated to the outside world. This filtration of the behaviour of the whole system constitutes a special view on the system that will be a realtime view in our example described in section 4. This is indicated by the right part of Figure 2 consisting of Observer-Spec_1 (instantiated by the Hybrid Automaton Gasburner) and the translation of this gasburner into a VSE specification (see OS-SD-SPEC_1 in Figure 2). The languages used in the real specification are VSE-SL (VSE-Specification Language) and Hybrid Automata as indicated in Figure 2.

In what follows, the general scenario is replaced by the real scenario of the gasburner.

4 Gasburner as Realtime Observation

The real scenario is included in the general one in Figure 2 by giving the instantiations for the Observer, Environment, and Controller components (that are the `TLSPECS Environment`, `Controller` and `Observer` respectively) and by giving instantiations for the `System Design Spec`, OS-SD-SPEC_1 and Observer-Spec_1

from Figure 1. A screenshot of the VSE development graph[5,6] of the implementation of the real scenario is given in Figure 3. In Figure 3 the composition of the environment, the controller and the observer is combined to the **gasburner** component representing the real gasburner.

Next we want to show that the VSE-observation (the VSE specification resulted from the translation of the hybrid gasburner into VSE[11] shown in Figure 3 as the temporal logic specification **gasprop**³ is satisfied by the combined **gasburner**. That way the VSE-observation represents a complex realtime property of this gasburner.

The definitions of the datatypes are located in the theories **def1**, **def2**, **def3**, and **Definition** in Figure 3. They define the possible values of the flexible variables of the component specifications.

In what follows we describe the scheduling of the three components and the specification of the components itself.

Fig. 3. Development Graph of the real Gasburner

³ This has its correspondence in the general scenario given in Figure 2 in the OS-SD-SPEC₁ specification.

4.1 Scheduling

The scheduling between the components of the gasburner, i.e. the environment, the controller, and the observer is realised with the help of the shared variable **who**. In the absence of such a scheduling variable, component steps are simply interleaved. Combining the components without scheduling would result in a situation where we did not know what component computes next. In particular we would not know when the observer makes its next observation. In order to define an observer component that is not too complicated and in order to guarantee that the observer makes an observation when significant changes happen we have introduced a scheduling where the computation of the components starts with the Observer followed by the Controller. After this first phase the computing order of the components is: Environment, Observer, Controller. Of course, there are other possibilities to schedule the components. One of these is to start with an initial sequence consisting of Observer, Controller and again the Observer and after that the repeated computation order is Environment, Observer, Controller, and Observer. It has revealed that this scheduling results in a more complicated behaviour description of the observer and in more complicated proofs. The implementation of the scheduling can be taken from the description of the various components shown in Figures 7, 4 and 5.

4.2 Environment

The specification of the environment is shown in Figure 5. It consists of the definition of the possible initial states and the actions the environment can take⁴. Note that, because of the initialisation shown in Figure 5, the VSE-observation's initial state is **leaking**.

The action the environment can take is described in the **ACTIONS**-slot by the action **env_act**. The environment is specified such that the **panel** can change arbitrarily, but the **leak_sensor** representing the sensor to measure whether unburned gas flows out has to work correct. This means that in a situation in which the **panel** is **off** or the gasflow is **blocked** the sensor is expected to deliver the right values. Specifying the environment this way results in the assumption that the environment is expected never to fail. Faulty sensors are excluded from the model⁵.

4.3 Controller

The TLSPEC of the controller is given in Figure 4. Again, its initial state description forces the gasburner observation to start in the **leaking** state.

The actions the controller can take are described in the **ACTIONS** slot by **A1** through **A7**. The actions are disjunctively connected to a single action **A1toA7**.

⁴ The doubling of the variables has technical reasons. It simulates an identical output sent to several components.

⁵ Faulty sensors can be specified giving them the possibility to change non-deterministically. Such problems do not concern us in this paper, though.

```

TLSPEC Controller
  USING Definitions
  DATA INTERNAL cstate : OnOff_t
    OUT timer : nat; gasflow_env, gasflow : gas_t
    IN leak_sensor : bool; now : nat; panel : OnOff_t
    SHARED INOUT who : schedule_t
  ACTIONS
    A1 ::= cstate = on AND panel = off AND leak_sensor = F AND
      cstate' = off AND gasflow' = blocked AND
      gasflow_env' = blocked AND UNCHANGED(timer)
    A2 ::= cstate = on AND panel = off AND leak_sensor = T AND
      cstate' = off AND gasflow' = blocked AND
      gasflow_env' = blocked AND timer' = now
    A3 ::= cstate = on AND panel = on AND leak_sensor = F AND
      UNCHANGED(cstate, gasflow, gasflow_env, timer)
    A4 ::= cstate = on AND panel = on AND leak_sensor = T AND
      cstate' = off AND gasflow' = blocked AND
      gasflow_env' = blocked AND timer' = now
    A5 ::= cstate = off AND panel = on AND now >= timer + (30 * c) AND
      cstate' = on AND gasflow' = run AND gasflow_env' = run AND
      UNCHANGED(timer)
    A6 ::= cstate = off AND panel = on AND now < timer + (30 * c) AND
      UNCHANGED(cstate, gasflow, gasflow_env, timer)
    A7 ::= cstate = off AND panel = off AND
      UNCHANGED(cstate, gasflow, gasflow_env, timer)
    A1to7 ::= who = contr AND who' = env AND A1 OR A2 OR ... OR A7
  /*Definition of the behaviour of the gasburner */
  SPEC INITIAL gasflow = run AND gasflow_env = run AND
    cstate = on AND timer = 0 AND who = obs
    TRANSITIONS [A1to7] {gasflow, gasflow_env, timer, cstate}
  TLSPECEND

```

Fig. 4. Specification of the Controller

Since in every moment of execution only one of these actions is enabled, only this very action can be taken unless the controller stutters.

Up to now we have specified the controller together with its environment. By adding a clock component that mimics the flow of time (the change of the variable *now*) we end up in a system for which we can prove certain realtime properties.

As described in section 1 our methodology is different. It is not our aim to prove single realtime properties of a system, rather we want to check its entire realtime behaviour. How this is realised is explained in the next sections.

4.4 Observer

As can be recognised by looking at the specification of the controller and the environment, a simple refinement mapping [1] is not enough to prove the refine-


```

TLSPEC Env
PURPOSE "Specification of the Environment"
USING Definitions
DATA OUT panel_contr, panel_obs : OnOff_t
      OUT leak_sensor_obs, leak_sensor : bool
      IN gasflow_env : gas_t
      SHARED INOUT who : schedule_t
ACTIONS
env_act ::= /*scheduling*/
          who = env AND who' = obs AND
          (/*switch the gasburner on or off*/
            (panel_contr' = on AND panel_obs' = on) OR
            (panel_contr' = off AND panel_obs' = off)) AND
          /*Environment behaves correct*/
          IF (panel_obs' = off OR gasflow_env = blocked)
          THEN (leak_sensor' = F AND leak_sensor_obs' = F)
          ELSE (leak_sensor' = T AND leak_sensor_obs' = T) OR
              (leak_sensor' = F AND leak_sensor_obs' = F) FI
SPEC INITIAL leak_sensor = T AND leak_sensor_obs = T AND
          panel_obs = on AND panel_contr = on AND who = obs
TRANSITIONS [env_act] {panel_obs, panel_contr, who,
                       leak_sensor, leak_sensor_obs}
TLSPECEND

```

Fig. 5. Specification of the Environment

ment. One reason for that is that in a refinement mapping only those variables can be mapped that are known in the actual state. In this sense a refinement mapping is a filter that maps the states of the implementing system to the states of the implemented system. The refinement model described here uses the usual refinement mapping extended by an observer component. The observer component together with the refinement mapping represents an external observer filtering the observed behaviour and mapping these filtered behaviours to special observer behaviours represented in our case by the translated hybrid gasburner shown in section 4.5. As already mentioned in section 1 observations may look at the system from different angles. This could be a data flow perspective or, as in our example here, a realtime perspective. The responsibility of the observer is to map the states of the real gasburner to the (virtual) states of the translated hybrid gasburner. In the easiest case the mapping only consists of a variable mapping which simply renames or recomputes the values of the variables according to certain given functions. The mapping of the variable **state** of the translated hybrid gasburner is somewhat more complicated since there is no immediate correlation between variables of the implementing system and the variable **state**. Moreover, the mapping does not only depend on the actual state of the implementing system but also on some other information given by the behaviour of the observer. This situation is illustrated in Figure 6. Some of the steps of the hybrid gasburner in Figure 6 relate to stuttering steps and some

Fig. 6. Refinement Relation

relate to real steps. Thus the Observer calculates steps of the hybrid gasburner from the steps of the environment and the controller. In addition the observer makes the clock available to all the system components.

We now come to the actual specification of the observer (given in Figure 7). In its specification it should be avoided to rebuild the specification of the hybrid gasburner shown in section 4.5. This can be accomplished by certain syntactical restrictions on the observer's actions. For instance only the variables of the controller and the environment are allowed in the enabling conditions.

The observer has to handle the following situations that might occur in the hybrid gasburner: The system is in its initial state and gas flows out of the nozzle without being burned, the system changes from leaking to non-leaking, the system remains leaking or it changes from non-leaking to leaking. To recognise these situations in the observer component it is not enough to only have a look at the values of the input variables in the actual state. Looking at these variables the observer could not recognise whether there was a situation change from leaking to non-leaking or not. In order to detect all these situation changes the observer stores parts of the previous observation in internal variables (**gas_int** and **leak_sens_int**).

Let us have a more detailed look at one of the possible situations. Assume that the condition **leak_to_non_leak** (see Figure 7) is satisfied. This means that the actual values of the variables **gasflow** and **leak_sens** are **blocked** and **F**, respectively. Moreover, assume that the previous observations of the observer where that there was a running gas flow and a leak, i.e. **gas_int** is **run** and **leak_sens_int** is **T**. This situation indicates that the system happened to be in a leaking situation and changed to a non-leaking situation. The observer reacts on this change by setting the **state** variable to **non_leak** and by resetting the variable **x** to 0. This corresponds exactly to the behaviour of the hybrid system in such a situation. That way the behaviour of the hybrid gasburner is filtered out of the behaviour of the controller and the environment.

```

TLSPEC Observer
USING Definitions
DATA IN panel_obs:OnOff_t;leak_sensor:bool;gasflow:gas_t;timer:nat
    SHARED INOUT who:schedule_t
    OUT panel_out:OnOff_t;leak_sens_out:bool;gasflow_out:gas_t;
        now, now_out:nat;x,t_obs:nat;state:state_t
    INTERNAL leak_sens_int:bool;gas_int:gas_t
ACTIONS
/*Condition for the initial phase*/
init_cond ::= leak_sens = T AND leak_sens_int = T AND
                gasflow = run AND gas_int = run
/*Condition for the leaking to non_leaking phase*/
leak_to_non_leak ::= gasflow =blocked AND gas_int = run AND
                leak_sens = F AND leak_sens_int = T
/*Condition for the non_leaking to leaking phase*/
non_leak_to_leak ::= gasflow =run AND leak_sens = T AND
                leak_sens_int = F
/*Special case for initial phase*/
obs1 ::= state' = leaking AND unchanged(x, ...,gas_int, now, now_out)
/*leaking to non_leaking action*/
obs2 ::= state' = non_leaking AND x' = 0 AND t_obs' = t_obs AND
        now' = now AND now_out' = now_out AND gas_int' = gasflow
        AND leak_sens_int' = leak_sens
/*non_leaking to leaking action*/
obs3 ::= state' = leaking AND x' = 0 AND
        now' = now AND now_out' = now_out AND t_obs' = t_obs AND
        leak_sens_int' = T AND gas_int' = run
/*Action describing the remaining in non_leaking state. Enabled if:
    not init_cond AND not leak_to_non_leak AND not non_leak_to_leak*/
obs4 ::= state' = non_leaking AND x' = x + 1 AND now' = now + 1 AND
        now_out' = now_out + 1 AND t_obs' = t_obs AND
        leak_sens_int' = leak_sens AND gas_int' = gasflow
/*Observer action with scheduling and setting of output variables*/
obs_act ::= who = obs AND who' = contr AND gasflow_out' = gasflow AND
        panel_out' = panel_obs AND leak_sens_out' = leak_sensor AND
        IF init_cond THEN obs1
        ELSE IF leak_to_non_leak THEN obs2
        ELSE IF non_leak_to_leak THEN obs3 ELSE obs4
        FI FI FI
SPEC INITIAL who = obs AND panel_out = on AND leak_sens_out = T AND
        gasflow_out = run AND gas_int = run AND leak_sens_int = T
        AND now = 0 AND now_out = 0 AND x = 0 AND t_obs = 0 AND
        state = leaking
TRANSITIONS [obs_act] {now, gasflow_out, panel_out,
        leak_sens_out, who, leak_sens_out}
TLSPECEND

```

Fig. 7. Specification of the Observer

4.5 The Gasburner as VSE Specification

As usual we illustrate hybrid automata as annotated graphs. The hybrid gasburner is then picturised as follows:

The meaning of the variables of the hybrid gasburner is as follows. y accumulates overall time. x represents a control clock that guarantees that the system remains for at most 1 time unit within location Leak and for at least 30 time units within location Non-Leak. t counts leakage time, i.e. the amount of time the system resides within Leak.

The translation of the hybrid gasburner (see [11]) then results in the VSE gasburner specification:

TLSPEC gasburner

PURPOSE "Specification of the gasburner. "

USING definition

DATA OUT $x, y, t : \text{nat}$

OUT state : state_t

ACTIONS

phi_1 ::= state = leaking AND state' = non_leaking AND
x' = 0 AND UNCHANGED(y, t)

phi_2 ::= state = non_leaking AND state' = leaking AND
x ≥ 30 * c AND x' ≤ c AND x' = 0 AND UNCHANGED(y, t)

phi_1_star ::= state = leaking AND state' = non_leaking AND
NOT x + 1 ≤ c AND x' = 0 AND UNCHANGED(y, t)

psi_1 ::= state = leaking AND state' = leaking AND
x + 1 ≤ c AND y' = y + 1 AND
x' = x + 1 AND t' = t + 1

psi_2 ::= state = non_leaking AND state' = non_leaking AND
x' = x + 1 AND y' = y + 1 AND UNCHANGED(t)

SPEC INITIAL x = 0 AND y = 0 AND t = 0 AND state = leaking

TRANSITIONS [phi_1, phi_2, phi_1_star, psi_1, psi_2]
{x, y, t, state}

FAIRNESS WF(phi_1_star) {x, y, t, state},

WF(psi_1) {x, y, t, state},

WF(psi_2) {x, y, t, state}

SATISFIES Safety

TLSPECEND

This VSE gasburner specification constitutes the **gasprop** TLSPEC given in Figure 3 and represents a complex realtime property we want to prove from the gasburner.

4.6 Example Behaviour of the Real Gasburner

This section mainly consists of the table shown in Figure 8 that provides a possible initial part of the behaviour of the real gasburner (see the upper part of the table) including the mapping done by the observer component and the refinement mapping⁶. The table shows how the filter consisting of the observer component works and therefore how it maps the behaviour of the real gasburner to the hybrid gasburner (see the lower part of Table 8). The proof given in section 4.8 finally states that every behaviour of the real gasburner is mapped to a behaviour of the hybrid gasburner and so that the refinement relation is established. For readability we use the abbreviations *n_l* for **non_leaking** and *bl*

	init	obs	contr	env	obs	contr	env	obs	contr	env	obs	...
cstate	on	on	off	off	off	off	off	off	of	off	off	...
timer	0	0	0	0	0	0	0	0	0	0	0	...
gasflow	run	run	bl	bl	bl	bl	bl	bl	bl	bl	bl	...
panel	on	on	on	on	on	on	on	on	on	on	on	...
leak_sens	T	T	T	F	F	F	F	F	F	F	F	...
now	0	0	0	0	0	0	0	1	1	1	2	...
gas_int	run	run	run	run	bl	bl	bl	bl	bl	bl	bl	...
leak_sens_int	T	T	T	T	F	F	F	F	F	F	F	...

y	0	0	0	0	0	0	0	1	1	1	2	...
x	0	0	0	0	0	0	0	1	1	1	2	...
t	0	0	0	0	0	0	0	0	0	0	0	...
state	l	l	l	l	n_l	n_l	n_l	n_l	n_l	n_l	n_l	...

Fig. 8. Example behaviour with refinement mapping

for **blocked** in Table 8. If we just consider the lower part we can see that this is also a possible behaviour of the hybrid gasburner.

4.7 Handling of the Constant *c*

The constant *c*, as introduced in [11], plays a special role in the example presented here. It is used to get an exact discretisation of the continuous behaviour of the original hybrid gasburner. Moreover, it is used in the specification of the Controller (see Figure 4) in the actions A5 and A6.

The role of the natural constants *c* is the same as in the translation of hybrid automata into VSE [11]. It makes the system independent from the chosen time unit. Let us assume, for example, the value of *c* is 10. Then the clock has to make 300 steps, before the barrier is reached. If, however, *c* is 1000 the clock has

⁶ The refinement mapping is shown in the development graph of the real gasburner in Figure 3. It consists of a mapping of the variables to be implemented to terms constructed from the variables of the implementing system

to tick 30.000 times to reach the barrier. I.e. if we consider seconds as the basic time unit then steps take $\frac{1}{10}$ of a second or a millisecond, respectively.

However, the real proof of the (safety) properties is done independent from specific values of the constant c . Therefore it is valid for all possible integer values and thus for all granularities (even the infinitesimal).

4.8 Refinement Proof

In this chapter we shall sketch the refinement proof indicated in Figure 3 by a VSE **satisfies**-link. The proof is done locally to the observer component. During the proof one immediately realises that assumptions about the behaviour of the environment of the observer are needed. These assumptions deal with two different proof situations. First, we have to know whether we are in a leaking or a non-leaking state. We insert this knowledge by the following invariants into the proof.

$$\begin{aligned} &\Box(\textit{intern}(\textit{observer}) \wedge \textit{init_cond} \rightarrow \textit{state} = \textit{leak}) \\ &\Box(\textit{intern}(\textit{observer}) \wedge \textit{leak_non_leak} \rightarrow \textit{state} = \textit{leak}) \\ &\Box(\textit{intern}(\textit{observer}) \wedge \textit{non_leak_leak} \rightarrow \textit{state} = \textit{non_leak}) \\ &\Box(\textit{intern}(\textit{observer}) \wedge \textit{non_leak_non_leak} \rightarrow \textit{state} = \textit{non_leak}) \end{aligned}$$

The definitions of *init_cond*, *leak_non_leak*, *non_leak_leak* and *non_leak_non_leak* are given below.

$$\begin{aligned} \textit{init_cond} &\hat{=} \textit{leak_sens} = T \wedge \textit{leak_sens_int} = T \wedge \textit{gasflow} = \textit{run} \\ &\quad \wedge \textit{gas_int} = \textit{run} \\ \textit{leak_non_leak} &\hat{=} \textit{gasflow} = \textit{bl} \wedge \textit{gas_int} = \textit{run} \wedge \textit{leak_sens} = F \\ &\quad \wedge \textit{leak_sens_int} = T \\ \textit{non_leak_leak} &\hat{=} \textit{leak_sens} = T \wedge \textit{leak_sens_int} = F \wedge \textit{gasflow} = \textit{run} \\ \textit{non_leak_non_leak} &\hat{=} \neg(\textit{init} \vee \textit{leak_non_leak} \vee \textit{non_leak_leak}) \end{aligned}$$

The formula *intern(observer)* represents the internal behaviour of the observer component, i.e. the behaviour of the observer without the hiding quantification.

In the second proof situation we need knowledge about the behaviour of the controller in the **non_leak_leak** situation. Again this knowledge is inserted by the following assumptions:

$$\begin{aligned} &\Box((\textit{leak_sens} = T \wedge \textit{leak_sens_int} = F \wedge \textit{gasflow} = \textit{run}) \\ &\quad \rightarrow \textit{now} \geq \textit{timer} + 30 * c) \\ &\Box((\textit{leak_sens} = T \wedge \textit{leak_sens_int} = F \wedge \textit{gasflow} = \textit{run}) \rightarrow x = \textit{now} - \textit{timer}) \end{aligned}$$

It is evident that the proofs of these assumptions need knowledge about the controller component as well as about the scheduling. Finally, the proof is performed locally to the observer component and is exported as a lemma to the global proof obligation.

5 Conclusion and Future Work

We have presented a methodology, observer models, for formal requirements engineering. Its applicability is illustrated with the help of a realistic gasburner example that is presented in detail.

One of the open issues in this context is the question how to refine a specification without doing the whole proof work again. This problem seems to be very similar to that of refinement in the security area, for example in protocol analysis.

References

1. Martín Abadi and Leslie Lamport. The Existence of Refinement Mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.
2. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifaki, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
3. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. T. A. Henzinger and P.-H. Ho. HYTECH: The cornell hybrid technology tool. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, pages 265–293. Springer Verlag, Lecture Notes in Computer Science, vol. 999, 1995.
5. D. Hutter, B. Langenstein, J. H. Siekmann, C. Sengler, W. Stephan, and A. Wolpers. Deduction in the verification support environment (vse). In *Formal Methods Europe (FME)*, LNCS. Springer, 1996.
6. Dieter Hutter, Heiko Mantel, Georg Rock, Werner Stephan, Andreas Wolpers, Michael Balser, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. VSE: Controlling the complexity in formal software developments. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, editors, *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*, Boppard, Germany, 1999. Springer-Verlag, LNCS 1641.
7. U. Institute and o Standards. Common criteria for information technology security evaluation, 1999.
8. Heiko Mantel. Possibilistic definitions of security - an assembly kit. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, Cambridge, England, 2000. IEEE Computer Society Press.
9. Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
10. Jonathan K. Millen. CAPSL: Common authentication protocol specification language. The MITRE Corporation, Technical Report MP 97B48, 1997. <http://www.csl.sri.com/~millen/capsl>.
11. A. Nonnengart, G. Rock, and W. Stephan. Expressing Realtime Properties in VSE-II. In *ESA Workshop on On-Board Autonomy*, volume WPP-191, pages 447–454, October 2001.
12. A. Nonnengart, G. Rock, and W. Stephan. Using Hybrid Automata to Express Realtime Properties in VSE-II. In Ingrid Russel and John Kolen, editors, *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, pages 640–644. AAAI Press, 2001.