Modelling, Specification and Verification of an Emergency Closing System

Werner Stephan and **Georg Rock** and **Michael Brodski**
Deutsches Forschungszentrum für
Künstliche Intelligenz (DFKI GmbH)
[German Research Center for Artificial Intelligence]
Email: stephan,rock,brodski@dfki.de

Abstract

In this paper we present a realtime modelling technique which is based on a global clock architecture with a discrete time scale. We argue that this model can be applied to a wide range of scenarios. Furthermore we illustrate the modelling technique by an example describing an emergency closing system which is specified in the VSE tool (Hutter *et al.* 1999; Rock, Stephan, and Wolpers 1997; 1999).

Introduction

In this paper we present a realtime modelling technique which is based on a global clock architecture with a discrete time scale. We argue that this model can be applied to a wide range of scenarios. Furthermore we illustrate the modelling technique by an example which describes an emergency closing system (ECS) that physically consists of several huge gates to isolate the North Sea from the Eastern Scheldt, a control system and several sensors to measure the water levels inside and outside the gates. The emergency control system which is part of the overall system keeps track of the changes of the water levels and closes the gates if the water level gets dangerous. The specification and the proofs are done with the VSE tool (Hutter *et al.* 1999; Rock, Stephan, and Wolpers 1997; 1999).

The paper is organized as follows: We first describe the general techniques to model realtime systems. After a discussion of the general structure of the ECS we present the VSE specification preceeded by a short introduction to the VSE tool. We close by mentioning future work in this area.

General Principles

In this section we discuss the general techniques used to model real time systems like the ECS. Since ECS is a very special system which in some aspects might not be considered typical for the kind of systems the methodology is designed for, the discussion in this section is more general than necessary for the formal model of the ECS.

Requirement Engineering Process

A formal model like the one of the ECS is only one constituent of the overall requirements engineering process. Starting point in the case of the ECS is a document containing the (informal) requirements specification.

The requirements are given by verbal descriptions and an informal, mainly graphical design of the system. The most important requirements deal with the behaviour of the system in time. While the verbal description is not very precise and therefore not sufficient in itself the graphical description contains many details of a particular solution, actually it is already close to a technical realization.

Our aim was to provide a formal requirements specification that does not refer to the internal structure of the system and then to prove that these requirements are satisfied by a system specification that is as close as possible to the design given in the document. In contrast to the technical description in the document, the formal system specification does not use concrete physical entities, like seconds, meters, and milliamperes. In a separate step one would have to choose concrete values that are in accordance with the constraints given by the formal specification.

Overall Structure of the Model

The formal model consists of three components: the *system* the *environment*, and a component containing a global *clock* (see Figure 1). The system takes as inputs values from sensors measuring various water levels and values from switches that are set by an operator. Basically it computes two output signals, one for closing the barrier and one for opening it again. The design is *fail safe* in the sense that the first signal going down means *close*.

Both, the environment and the clock are separated from the system to allow for a refinement of the abstract specification to the actual system. The environment comprises changes of the water levels as well as changes of the switches. In this case there are no complex assumptions about the possible behaviour of the environment. However, steps of the three components have to be synchronized to model assumptions about the behaviour in time. In particular not all states can be given a meaningful interpretation. Therefore the component containing the clock upon each tick *updates* certain *visible* variables. All the remaining variables are *internal* and cannot be accessed by an observer. Note that the synchronization among the components as well as the choice of observable variables implicitly formalize our way of looking at the system and are therefore

relevant for the notion of correctness[1].

The requirements specification only refers to the visible variables and has no access to the variables local to the system and also the synchronisation mechanism. However, it uses a variable *time* that gives the current time in each state.

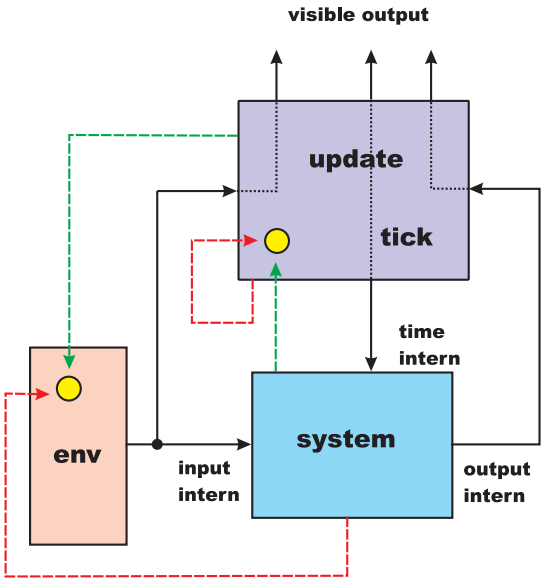

Figure 1: The complete scenario

Time

We assume a global *discrete* time scale with a *clock* that increases the value of a flexible variable *time* (of type *Nat*) by one upon each tick. By a small example (different from the ECS) we illustrate the general approach and argue that it is applicable for many scenarios where the granularity of the time scale can be fixed afterwards.

Figure 2 shows a possible behaviour of a system that reacts to an input signal ($signal_1$). A safety requirement would

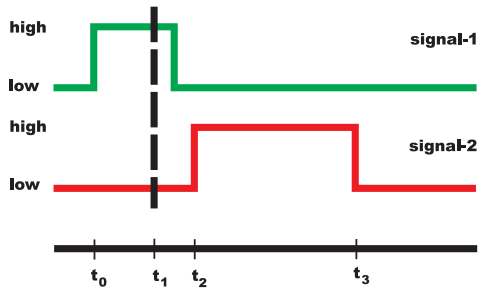

Figure 2: A possible behaviour of a real time system

be the following: If $signal_1$ is high for at least $\frac{1}{500}$ sec starting from t_0 and $signal_2$ is low at t_0, then after at most $\frac{1}{100}$ sec $signal_2$ will be high for at least $\frac{1}{4}$ sec. If for the time points t_0, t_1, t_2 and t_3 shown in the diagram, we have the conditions

$$\begin{aligned} t_1 - t_0 &\geq \frac{1}{500} \text{ sec} \\ t_2 - t_0 &\leq \frac{1}{100} \text{ sec} \\ t_3 - t_2 &\geq \frac{1}{4} \text{ sec} \end{aligned}$$

then this particular behaviour fulfils the requirement.

The aim is to provide a formal specification of the system, (assumptions about) the environment, and the clock, where in more complicated cases the system specification might by structured into several components running in parallel. In a separate specification we formalise safety requirements like the one above. Since in the formal model we use a discrete time scale we have to replace concrete durations by constants that stand for an arbitrary but fixed number of time steps. The above mentioned requirement could be formulated as a temporal logic formula in VSE as follows:

$$\begin{aligned} &\Box \forall\, t_0.(((t_0 = time \wedge signal_2 = low \wedge \\ &\qquad \Box(t_0 \leq time \leq t_0 + d_1 \rightarrow signal_1 = high)) \\ &\rightarrow \\ &\exists\, t_2.((t_2 - t_0) \leq d_2 \wedge \\ &\quad \Box((t_2 \leq time \leq t_2 + d_3) \rightarrow signal_2 = high))) \end{aligned}$$

In this example the premise that $signal_1$ is high for at least d_1 time units and the allowed delay of d_2 time units take into account the fact that the system might need a certain time d_4 to react and also a certain time d_5 to compute new output values. In the case of ECS both reaction time and computation time are considered to be zero according to the given documents. For d_3 it holds that $t_3 - t_2 \geq d_3$. In order to prove the timing requirements the specification has to make certain assumptions about relations between the durations mentioned. In our example the assumptions $d_4 \leq d_1$ and also $(d_4 + d_5) \leq d_1$ are required.

Having successfully proved the safety requirements we are free to choose concrete values for a time step. If all durations are given as rational numbers the only constraint is that a single step has to be small enough to have *all* durations as multiples of this value.

Verification Support Environment

In this section we first describe the Verification Support Environment (VSE) tool which was used to specify the model and to verify the properties of the system. In the second part we go into more detail with respect to the specification itself and give some hints to the proofs done in VSE.

The VSE Methodology

The VSE system is a tool for the formal development of software systems. It consists of: A basic system for editing and type checking specifications and implementations written in the specification language VSE-SL, a facility to display the development structure, a theorem prover for treating the proof obligations arising from development steps as for example refinements, a central database to store all aspects of the development and an automatic management of dependencies between development steps.

[1] Perhaps this could be compared to a test bed.

Compared to VSE I (Hutter *et al.* 1996a; 1996b), which was based on a simple, non-compositional approach for state based systems, VSE II (Hutter *et al.* 1999) is extended with respect to comprehensive methods in order to deal with *distributed* and *concurrent systems* (Rock, Stephan, and Wolpers 1997) and with respect to an even more efficient and uniform proof support which makes use of implicit structuring of the arising proof obligations. The basic formalism used in VSE II is close to TLA (Temporal Logic of Actions) (Lamport 1994). A refined *correctness management* allows for an evolutionary software development.

VSE is based on a methodology to use the structure of a given specification (e.g. parameterisation, actualisation, enrichment, or modules) to distribute also the deductive reasoning into local theories (Rock, Stephan, and Wolpers 1999). Each theory is considered as an encapsulated unit, which consists of its local signature and axioms. Relations between different theories, as they are given by the model-theoretic structure of the specification, are represented by different links between theories. Each theory maintains its own set of consequences or lemmata obtained by using local axioms and other formulas included from linked theories.

This method of a structured specification *and* verification is reflected in the central data structure of a *development graph* (see Figure 3), the nodes of which correspond to the units mentioned above. It also provides a graphical interface for the system under development.

Concurrent System Specifications

The system architecture of the ECS case study shown in Figure 1 indicates the components constituting the complete system including the environment specification. In the description of these units elementary specifications and different structuring operators for state based systems are used.

Elementary Specifications

For the specification of state transition systems a specification language close to TLA (Abadi and Lamport 1991; Lamport 1994; Abadi and Lamport 1995) is used. In addition to the theory of compositional development presented in (Abadi and Lamport 1995), which covers the composition of systems using input and output variables, *shared variables* are supported by the structuring operators in VSE II. The form of a specification of a component, also discussed in (Abadi and Lamport 1995), is

$$\exists\, x_1, \ldots, x_n.(\text{INIT} \wedge \Box[\text{SYS-STEPS}]_{\bar{v}} \wedge \text{FAIR}),$$

where SYS-STEPS are the *actions* (steps) made by the system, $\bar{v}$ is the stuttering index, which contains flexible variables of the system, INIT is a predicate which holds initially, $x_1, \ldots, x_n$ are the internal variables and FAIR stands for the fairness requirements of the system.

Structuring of Specifications

VSE II provides operators to structure state-based specifications. We only focus on the **combine**-operator which models the concurrent execution of components. As can be seen in Figure 3, the `SVKO_combine` specification consists of the (concurrent) composition of the `environment_data`, the `SVKO_system` and the `Update` components. Concurrency is modelled by considering all possible *interleavings* of actions of the combined systems. Basically a behavior σ, which represents a sequence of states of the specified system, is a behavior of the combined system if and only if it is a behavior of every component of the system. However, in order to model the concurrent execution of, say $\mathcal{S}_1$ and $\mathcal{S}_2$, by conjunction, we have to allow environment steps in the (local) specifications of $\mathcal{S}_1$ and $\mathcal{S}_2$. In (Abadi and Lamport 1995) environment steps are modelled by stuttering. This technique only works for communication by input-output variables, not in connection with shared variables. A more general approach (Rock, Stephan, and Wolpers 1999; Hutter *et al.* 1999) is to associate a "colour" with each component and to mark each step in a behavior by the colour of the component which has done the step.

Structured Deduction

Structuring specifications as described above supports readability and makes it easier to edit specifications in that the user might use local notions. However, the system exploits structure beyond this purely syntactical level. Components of a combined system can be viewed as systems in their own right where certain parts can be observed from outside while most of the inner structure, including the flow of control and local program variables are hidden.

In particular we can prove properties of a combined system like ECS in a modular way. This means that we attach local *lemma bases* to components where local proofs are conducted and stored. Exchange of information between lemma bases is on demand. This approach has two main advantages: First, the given structure of the specification is used to reduce the search space in the sense that large parts of the overall system are not visible and second, storing of proofs local to certain lemma bases and making the export and import of information (between lemma bases) explicit supports the *revision* process.

The Formal Model

The specification of the ECS is modelled with the VSE tool described in the preceeding section. The development graph of the specification is shown in Figure 3. The structure of the development graph is very similar to the description of the scenario given in Figure 1. The three nodes `environment_data`, `SVKO_system` and `Update` from Figure 3 correspond to the `env`, `system` and `update` nodes of Figure 1, respectively. The node `SVKO_combine` in Figure 3 represents the composed system consisting of three concurrent components. The properties the system has to satisfy are specified in the temporal logic specification `SVKO_safety`.

Synchronisation

According to the document we started with it can be assumed that the system immediately (i.e. without any delay) notices a change of the input variables caused by the environment or the clock and needs no time to compute an output. Obviously the concurrent execution of the three components has to be restricted appropriately to model this assumption. For example, if the enviroment changes the water-

level and the clock ticks twice before the system has computed possibly new output values, then there can be an intermediate state where a safety requirement is violated. So the clock has to be blocked until the system has finished its computation. Note that also in cases where there are certain (restricted) reaction and computation times a similar, slightly more complicated scheduling regime is necessary to rule out behaviours where reaction or computation takes to much time. In all these situations *fairness* which forces a step to be executed sometimes in the future is not enough.

Technically the scheduling among the three components is realized by shared variables acting as guards (indicated by circles within the components in Figure 1). An action (step) of a component can only be executed if the guard's value is $true$.

Unless there was a tick of the clock the variable $time$ has the same value. Hence the environment having changed the input of the system and the system having computed certain outputs the clock should tick, because otherwise we would have two different situations with the same time stamp.

But even if the clock ticks frequently enough for the intermediate states we have different input values at the same point of time. Moreover, immediately after the environment has changed some input values the output values of the system might not have the values requested by the requirements specification.

To overcome this problem we distinguish between internal variables and variables visible to the outside. The environment, the system, and the clock change only internal variables. Whenever the clock ticks the corresponding visible variables are updated with the current values of the internal ones. The observable variables remain the same in the intermediate states mentioned above. If the internal variables are finally *hidden* (by existential quantification over flexible variables) no regular behaviour can be observed for them in the resulting system. Note that still there has to be a tick of the clock and the new values of the internal variables become visible to the outside whenever the system has updated its output values.

There are many ways of implementing these rules. We have chosen a liberal implementation shown in Figure 1. The underlying datatype definitions are made in the *theories*

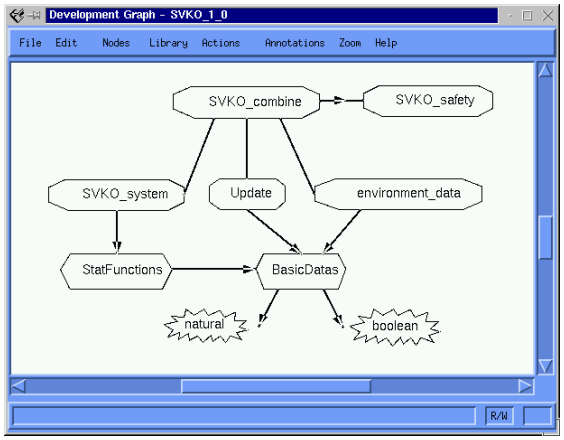

Figure 3: The VSE development graph of the ECS

`StatFunctions` and `BasicDatas` (see Figure 3). The *theories* `natural` and `boolean` are predefined theories in the VSE system.

In the following we describe the specifications of the different components.

Environment Specification

In our application scenario, the physical environment consists of the natural changes of waterlevels which have various complex causes. Since we do not want to specify these causes and since our main interest is in specifying the real-time behavior of the `SVKO_system` we have specified the environment as abstract as possible. There is only one action which represents the change of the (inside and outside) waterlevel sensors. These changes are transmitted to the `SVKO_system` and to the `Update` component.

System Specification

The data delivered from the environment to the system are the values of the inside and outside waterlevel sensors. These data are used in the `SVKO_system` to compute the values of two signals: the `CLOSE` and the `OPEN` signal. If the `CLOSE` signal is true, then the system should close the gates and if the conditions for the `OPEN` signal are true then the system should again open the gates.

The computations the system does are modelled in a single action and are separated into two parts: a static and a dynamic part. In the static part we have modelled diagrams from the original (graphical) description of the ECS as abstract datatypes and their corresponding functions. A very simple example is a $2/3$ voter. It returns true if at least two of the three inputs are true and false otherwise. This voter is specified simply as a predicate `voter2from3` with the following axiomatic definition:

```
ALL sig1, sig2, sig3 :
    voter2from3(sig1,sig2,sig3) <->
    (sig1 = T AND sig2 = T) OR
    (sig2 = T AND sig3 = T) OR
    (sig1 = T AND sig3 = T)
```

The specification of the dynamic part is more complex. This is mainly concerned with the description of the timing behavior of the system. To give an example think of a monostable multivibrator which starts working by a high to low trigger and then keeps the output signal low for at least `d` time units and leaves the signal high in all other cases. To remember the values on the input line of the multivibrator in order to determine if such a trigger has happened, we have inserted a flexible variable (`CLOSE_TRIGGER`) which stores the "old" value on the line and the current value. Depending on the relation between the old and the new value the multivibrator is active or not. This again is specified by a timer (`timer_CLOSE`) which is set in case that the multivibrator is active and which remains unchanged in all other cases. The following specification part models such a multivibrator:

```
IF (CLOSE_TRIGGER = T AND
    CLOSE_TRIGGER' = F)
THEN timer_CLOSE' = time + d
ELSE timer_CLOSE' = timer_CLOSE
```

In this formula we refer to a flexible variable `time` which represents the actual time in the system. This variable is sent from the `Update` component to the `SVKO_system` component.

Update Specification The specification of the `Update` component models mainly two properties of the whole system specification. First it filters the signals visible to the outside world, and second represents the global clock and increments the $time$ variable. The specification of the `update` component consists of a single action which increments time by 1 in every step and makes the variables representing the `CLOSE` and the `OPEN` signal and the variables representing the waterlevel sensor values visible to the outside world.

Property Specification

The ECS should satisfy some safety properties which are important for the right functioning of the system. We have proven among others the following two properties:

1. $\Box(\neg(\texttt{OPEN} = \texttt{T} \wedge \texttt{CLOSE} = \texttt{T}))$
2. $\Box((time = t_0 \wedge Change\_Sensor\_Sig) \Longrightarrow$
   $\Box((t_0 < time \leq t_0 + d + 1) \Longrightarrow \texttt{Close} = \texttt{T}))$

Property 1 says that the `OPEN` and the `CLOSE` signal are never true at the same time. Property 2 says that if the waterlevels get dangerous (expressed by the formula $Change\_Sensor\_Sig$) at time t_0 then the system reacts by setting the `CLOSE` signal to true for at least d time units beginning at time $t_0 + 1$.

The proofs of these properties are all done locally in the `SVKO_system` component. Property 1 could be proved without the use of assumptions whereas the proof of property 2 is more complex. It needs assumptions which have to be guaranteed[2] by the environment of the `SVKO_system`.

Future Work

In the requirements engineering phase of this project we have drawn pictures as for example given in Figure 2 to analyse the timing behavior of the system we want to specify. There we recognize that this way of analysing a realtime system is not very satisfying since we are only able to describe one special behavior with such a diagram. A more abstract and structured way is to use timed automata (Alur and Dill 1994) or hybrid automata (Henzinger 1996) to specify the timing behavior of a system and to integrate them in a formal software development process which is fully supported by a tool like VSE.

References

Abadi, M., and Lamport, L. 1991. The existence of refinement mappings. *TCS* 82(2):253–284.

Abadi, M., and Lamport, L. 1995. Conjoining specifications. *TOPLAS* 17(3):507–534.

Alur, R., and Dill, D. L. 1994. A theory of timed automata. *Theoretical Computer Science* 126:183–235.

Henzinger, T. A. 1996. The theory of hybrid automata. In *Proceedings of the 11th LICS*, 278–292. IEEE Comp. Soc. Press.

[2]This must be proved in the VSE system.

Hutter, D.; Langenstein, B.; Sengler, C.; Siekmann, J. H.; Stephan, W.; and Wolpers, A. 1996a. Deduction in the Verification Support Environment (VSE). In Gaudel, M.-C., and Woodcock, J., eds., *Proceedings Formal Methods Europe 1996: Industrial Benefits and Advances in Formal Methods*. SPRINGER.

Hutter, D.; Langenstein, B.; Sengler, C.; Siekmann, J. H.; Stephan, W.; and Wolpers, A. 1996b. Verification support environment (VSE). *High Integrity Systems* 1(6):523–530.

Hutter, D.; Mantel, H.; Rock, G.; Stephan, W.; Wolpers, A.; Balser, M.; Reif, W.; Schellhorn, G.; and Stenzel, K. 1999. VSE: Controlling the Complexity in Formal Software Development. In Hutter, D.; Stephan, W.; Traverso, P.; and Ullmann, M., eds., *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*. Boppard, Germany: Springer-Verlag, LNCS 1641.

Lamport, L. 1994. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems* 16(3).

Rock, G.; Stephan, W.; and Wolpers, A. 1997. Tool support for the compositional development of distributed systems. In *Tagungsband 7. GI/ITG-Fachgespräch Formale Beschreibungstechniken für verteilte Systeme*, number 315 in GMD Studien. GMD.

Rock, G.; Stephan, W.; and Wolpers, A. 1999. Modular reasoning about structured TLA specifications. In Berghammer, R., and Lakhnech, Y., eds., *Tool Support for System Specification, Development and Verification*, Advances in Computing Science, 217–229. Springer, Wien-NewYork.