Specification Of Embedded, Real-time Systems

Skakkebæk, Jens Ulrik; Ravn, Anders P.; Rischel, Hans; Chaochen, Zhou

Published in:
Proceedings of the Fourth Euromicro workshop on Real-Time Systems

Publication date:
1992

Document Version
Publisher's PDF, also known as Version of record

Specification of Embedded, Real-Time Systems*

Jens U. Skakkebæk, Anders P. Ravn, Hans Rischel & Zhou Chaochen †
Department of Computer Science ‡
Technical University of Denmark, Bldg. 344
DK 2800 Lyngby, Denmark

Abstract

An approach to requirements specification and subsequent verification of designs for embedded, real-time systems is presented. A system is given by a conventional mathematical model for a dynamic system, where application specific state variables denote total functions of real time. Specifications are formulas in a real-time, interval temporal logic, where atomic predicates define durations of states. Requirements are specified by a conjunction of formulas, which reflect safety and functionality constraints on the total system. A design specifies the behaviour of components and the conjunction of component specifications can be shown to imply the requirements. Designs can be refined in a similar fashion.

1 Introduction

Requirements engineering [4] for software that controls physical systems must investigate safety and functional requirements for the system as a whole. Requirements typically delimit expected behaviours over time for a combination of given physical processes with planned sensors, actuators and programmed computers. Requirements engineering is ideally completed with precise specifications for components and the design binding them together. This paper illustrates an approach to requirements engineering which has evolved during our work with case studies within the Provably Correct Systems (ProCoS) project [1, 13].

The approach uses a conventional time-domain model of mathematical systems theory or control engineering [12] and develops predicates describing properties of a total system in three steps:

1. An application domain *system model* of the equipment and its intended environment of use is defined. This defines the overall states of the system as functions of time. Requirements constrain this model.
2. A *control model* extends the system model with an explicit *control strategy*. The strategy is verified to imply the requirements under a set of *assumptions* about the intended environment of use.
3. A *design model* for a distributed system defines separate specifications for a set of *interface units* and *programs*. Interface units relate system states to event values under certain timing and approximation constraints. Programs implement the control strategy by controlling timing and order of events, and by computing relevant event values.

The approach is based on *refinement* of models. Each refinement removes some freedom (choice, nondeterminism) by adding further constraints. At each stage the resulting model is verified to be contained in (or imply) the model of the previous stage.

The following sections introduce the specification language, and then discuss the system and control models with associated refinements in more detail, using a simple Railway Level Crossing as a running example. The approach has also been used on a simple Auto Pilot [14] (example due to Boyer and Moore) and on a Gas Burner [11].

2 Specification language

A system is described by a collection of *state variables* which are functions of *Time*, modelled by the real numbers. Properties of systems are expressed by constraints on the state variables. We wish to express requirements and design without explicit mentioning of time instants, and introduce a notation which is a real-time, interval logic [9] based on state durations: The Duration Calculus [2].

2.1 Syntax

We assume names for state variables X, Y, ... together with their *value domains* $Type_X$, $Type_Y$, ... given by declarations in a suitable specification language, here **VDM** (cf. [5]). The language should comprise names for constants and operators. The type $\mathbf{R}$ of real numbers should be available with the usual operators, and so should the type **Bool** of Boolean values with the usual propositional operators. The Boolean constants are denoted *tt* and *ff* (the names *true* and *false* are reserved for duration formulas). We use lower case names a, b, ... to denote *constants* or *static variables* of any type in the language. Static variables denote time-independent entities.

*This work is partially supported by the Commission of the European Communities (CEC) under the ESPRIT programme in the field of Basic Research Action proj. no. 3104: "ProCoS: Provably Correct Systems" and by The Danish National Engineering Research Council

†On leave from Software Institute, Academia Sinica, Beijing 100080, China

‡E-mail: jus@id.dth.dk, apr@id.dth.dk, hsr@id.dth.dk and zcc@id.dth.dk

State expressions and state assertions

A *state expression* is generated by a constant, a static variable, a state variable or any (type correct) expression $op(S_1, \dots, S_n)$ formed from an operator symbol op and state expressions $S_1, \dots, S_n$.

A *state assertion* is a state expression with type **Bool**.

Durations and duration terms

For any state assertion P, $\int P$ is a *duration*. A *duration term* (of type $\mathbf{R}$) is a duration, a real constant, a real static variable or the term $op(r_1, \dots, r_n)$, where op is an n-ary operator symbol of type $\mathbf{R}$ and $r_1, \dots, r_n$ are duration terms.

The symbol ℓ is used as an abbreviation for the duration term $\int tt$.

Duration formulas

If A is any n-ary predicate symbol on $\mathbf{R}$ and $r_1, \dots, r_n$ are duration terms, then $A(r_1, \dots, r_n)$ is an *atomic duration formula*. Atomic duration formulas, the symbols *true* and *false*, $(\neg \mathcal{D}_1)$, $(\mathcal{D}_1 \vee \mathcal{D}_2)$, $(\mathcal{D}_1 \ ; \ \mathcal{D}_2)$, and $(\forall x)\mathcal{D}_1$, where x is a static variable and $\mathcal{D}_1$, $\mathcal{D}_2$ are duration formulas, are *duration formulas* of type **Bool**. We use standard abbreviation $\wedge$, $\Rightarrow$, $\Leftrightarrow$ for both state assertions and duration formulas, and we introduce abbreviations for commonly used duration formulas:

Abbreviation	Formula	Legend
$\lceil\rceil$	$\ell = 0$	point
$\lceil P \rceil$	$\int P = \ell \wedge \ell > 0$	almost everywhere P
$\Diamond \mathcal{D}$	$true \ ; \ \mathcal{D} \ ; \ true$	somewhere $\mathcal{D}$
$\Box \mathcal{D}$	$\neg(\Diamond \neg \mathcal{D})$	always $\mathcal{D}$
$\mathcal{D}_1 \longrightarrow \mathcal{D}_2$	$\mathcal{D}_1 \ ; \ true \Rightarrow \mathcal{D}_1 \vee (\mathcal{D}_1 \ ; \ \mathcal{D}_2 \ ; \ true)$	$\mathcal{D}_2$ follows $\mathcal{D}_1$

The following precedence rules are used

first:	$\neg$, $\Box$, $\Diamond$
second:	$\vee$, $\wedge$, ;
third:	$\Rightarrow$, $\longrightarrow$

2.2 Semantics

A (particular) behaviour $\mathcal{B}$ of a system assigns a function $[0, \infty) \rightarrow Type_X$ to each state variable X, and selects a value $\mathcal{V}(x)$ for each static variable x. Each state expression then denotes a function obtained by evaluating the expression for each point of time. For a state assertion P it seems reasonable to demand *finite variability*: For each behaviour, any observation interval can be divided into finitely many subintervals with P constant on each (open) subinterval.

An observation *interval* is a closed and bounded interval $[b, e] \subset [0, \infty)$. For a given interval the duration $\int P$ of a state assertion P denotes the real number

$$\int_b^e (\textbf{if } P(t) = tt \textbf{ then } 1 \textbf{ else } 0)dt$$

which is the measure of the set of points where P has value tt. For any behaviour $\mathcal{B}$ and interval $[b, e]$ duration terms denote real values and atomic duration formulas denote Boolean values.

The values of composite duration formulas are obtained by the usual interpretation of the logical operators and quantification, cf. e.g. [6]. The value of a "*chop*" formula $\mathcal{D}_1 \ ; \ \mathcal{D}_2$ is tt if and only if the interval $[b, e]$ can be divided into $[b, m]$ and $[m, e]$ such that $\mathcal{D}_1$ is tt on $[b, m]$ and $\mathcal{D}_2$ is tt on $[m, e]$.

A duration formula $\mathcal{D}$ *holds* on the interval $[b, e]$ for the behaviour $\mathcal{B}$ just when $\mathcal{D}$ has value tt on $[b, e]$ with any assignment $\mathcal{V}$ of values to the static variables.

The formula $\mathcal{D}$ *holds from start* for the behaviour $\mathcal{B}$ just when it holds on any interval of the form $[0, T]$ for the behaviour $\mathcal{B}$.

A duration formula $\mathcal{D}$ is *valid* (a tautology) just when it holds for every behaviour $\mathcal{B}$ and every interval $[b, e]$. It is sufficient for a formula to be valid, that it holds from start for every behaviour $\mathcal{B}$.

2.3 Specifications and refinement

A *specification* for a system is a duration formula $\mathcal{D}$. A behaviour $\mathcal{B}$ *satisfies* the specification if $\mathcal{D}$ holds from start for $\mathcal{B}$.

For specifications $\mathcal{D}_1$ and $\mathcal{D}_2$ we say, that $\mathcal{D}_2$ is a *refinement* of $\mathcal{D}_1$ if any behaviour satisfying $\mathcal{D}_2$ also satisfy $\mathcal{D}_1$. It follows, that $\mathcal{D}_2$ is a *refinement* of $\mathcal{D}_1$ if the duration formula $\mathcal{D}_2 \Rightarrow \mathcal{D}_1$ is valid.

2.4 Proof system, Verification

It is almost certain, that the general Duration calculus is undecidable, and hence we cannot expect to find a complete set of axioms and proof rules. The proofs in this paper may, however, be based on the set of axioms and proof rules given below. In the following, P denotes a state assertion, r a non-negative real number and $\mathcal{D}$ denotes a formula in the Duration Calculus.

Axiom 1 $\int ff = 0$

Axiom 2 $\int P \geq 0$

Axiom 3 $\int P_1 + \int P_2 = \int (P_1 \vee P_2) + \int (P_1 \wedge P_2)$

Axiom 4 $(\int P = r_1) \ ; \ (\int P = r_2) \Leftrightarrow \int P = r_1 + r_2$

Axiom 5 *If* $P_1 \Leftrightarrow P_2$ *is a valid state assertion then* $\int P_1 = \int P_2$ *is an axiom*

The following induction rule is sound due to the finite variability of states.

Induction Rule If $\mathcal{D}(\lceil\rceil)$ is provable, and $\mathcal{D}(X \vee (X \ ; \ \lceil P \rceil) \vee (X \ ; \ \lceil \neg P \rceil))$ is provable from $\mathcal{D}(X)$, then $\mathcal{D}(true)$ has been proved.

It has a dual, backward induction rule.

3 System model

The first step in formalising requirements to a system is to construct a system model. Requirements are constraints on this model. This section builds a system model for our running example: A railway level crossing.

3.1 Informal description

The following informal description is based on the description found in [10] and on discussions with engineers from DSB (Danish National Railways):

The railway level crossing is a crossing between a single track railway and a road. For simplicity it is assumed that all trains passing the crossing will travel in the same direction. The crossing is shown in figure 1.

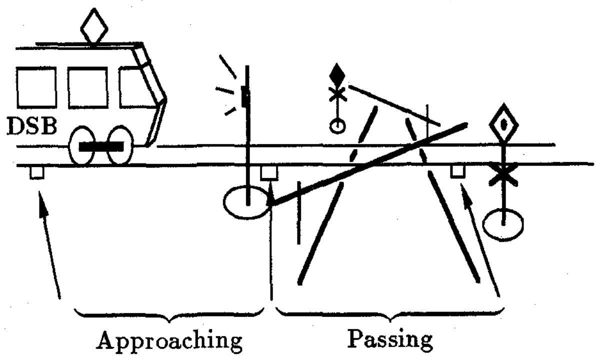

Figure 1: Railway Crossing

Road traffic is controlled by a gate at each side of the railway. The gates close only when road traffic is not stuck in the crossing.

Train traffic is controlled by a signal on the side of the tracks along which the trains approach. The signal indicates: "stop" or "go" for oncoming trains.

Sensors keep track of trains in the system. A sensor is placed in a reasonable distance from the signal such that a train will reach the first sensor point before it reaches the signal. A train enters the system, whenever it is determined by this sensor. A train has left the crossing, whenever the sensors determine that the rear end of the train has passed the crossing.

When a train approaches, the gates are closed, provided road traffic is not stuck in the crossing. The signal is only set to "go" after the gates have closed.

When no trains are approaching or passing, the signal must be set to "stop" and the gates opened.

The main objective of the system controlling the gates and the signal is to ensure safety: The system must never allow train and road traffic to pass the crossing at the same time.

Furthermore, the system must ensure that both road traffic and trains are able to pass the crossing within some reasonable time.

The device that controls the system is the **R***ailway Crossing* **C***ontrol* **S***ystem* (RCS). The RCS gets input data from the train sensors and the gate sensors, and sends commands to the signal and the gates.

3.2 State variables

The *state* models a signal, gates, road traffic and trains.
The *Signal* can be "stop" (*ff*) or "go" (*tt*)

$Signal : \mathbf{Bool}$

The *gates* can be *open*, *closed*, *opening* or *closing*

$Gates : \{open, closed, opening, closing\}$

The *road traffic* can be *stopped* at the crossing, be *stuck* in the crossing or be *free* to cross

$Traffic : \{stopped, stuck, free\}$

Trains are either approaching or passing

$Appr : \mathbf{N}\text{-}\underline{set}$
$Pass : \mathbf{N}\text{-}\underline{set}$

where a train, identified by a unique, natural number i, is approaching ($i \in Appr$) if some part of the train is between the sensor and the signal, and passing ($i \in Pass$) if some part of the train is between the signal and the end of the crossing. Trains are *active* if either *approaching* or *passing* (or both). We define three state assertions expressing the state of the trains

$$\begin{array}{ll} Passing & \widehat{=}\ Pass \neq \emptyset \\ Approaching & \widehat{=}\ Appr \neq \emptyset \\ Active & \widehat{=}\ Act \neq \emptyset \end{array}$$

where $Act = Appr \bigcup Pass$

3.3 Requirements

The requirements concern *safety* and *function*

$$Req \mathrel{\widehat{=}} \Box(SafReq \wedge \bigwedge_{i=1}^{3} FunReq\ i)$$

Safety requirement

If the gates are not closed or road traffic is stuck in the crossing, the trains must not pass

$$\begin{array}{l} SafReq \mathrel{\widehat{=}} \lceil (Gates \neq closed) \vee (Traffic = stuck) \rceil \\ \qquad \Rightarrow \lceil \neg Passing \rceil \end{array}$$

Functional requirements

1. The road traffic should maximally be held back for a predefined period of time, *Tstopped*

$$FunReq1 \mathrel{\widehat{=}} \lceil Traffic = stopped \rceil \Rightarrow \ell \leq Tstopped$$

2. When all trains have left the railway crossing, the gates must be open for at least time, *Topen*

$$\begin{array}{l} FunReq2 \mathrel{\widehat{=}} \lceil Active \rceil;\ \lceil \neg Active \rceil;\ \lceil Active \rceil \\ \qquad \Rightarrow \int (Gates = open) > Topen \end{array}$$

3. Provided the road traffic is not stuck, a single train must be able to pass within time, *Tactive*

$$\begin{array}{l} FunReq3 \mathrel{\widehat{=}} \lceil i \in Active \wedge (Traffic \neq stuck) \rceil \\ \qquad \Rightarrow \ell \leq Tactive \end{array}$$

4 Control model

We now turn to selecting the assumptions and a control strategy, such that the controlled system can be proved to satisfy the requirements under the given assumptions.

4.1 Assumptions

The assumptions constrain the system environment. In the case of the railway level crossing, it is necessary to constrain the behaviour of both the road traffic and the trains, as well as the devices

$$ASM \mathrel{\hat{=}} \Box(\bigwedge_{i=1}^{2} RTAsm\,i \wedge \bigwedge_{i=1}^{5} TAsm\,i \wedge \bigwedge_{i=1}^{3} DAsm\,i)$$

Road traffic assumptions

1. When running freely, the road traffic might eventually either be stopped properly by the gates or become stuck in the crossing. Stopped and stuck road traffic might in turn become free again

$$\begin{aligned} &RTAsm1 \mathrel{\hat{=}} \\ &\quad (\lceil Traffic = stopped \rceil \longrightarrow \lceil Traffic = free \rceil) \\ &\wedge\ (\lceil Traffic = free \rceil \longrightarrow \\ &\qquad (\lceil Traffic = stopped \rceil \vee \lceil Traffic = stuck \rceil)) \\ &\wedge\ (\lceil Traffic = stuck \rceil \longrightarrow \lceil Traffic = free \rceil) \end{aligned}$$

2. Road traffic is stopped, if and only if the gates are not open

$$\begin{aligned} &RTAsm2 \mathrel{\hat{=}} \\ &\lceil Traffic = stopped \rceil \Leftrightarrow \lceil Gates \neq open \rceil \end{aligned}$$

Train assumptions

1. Trains must only pass if the signal is "go"

$$TAsm1 \mathrel{\hat{=}} \lceil Passing \rceil \Rightarrow \lceil Signal \rceil$$

2. An active train travels in one direction only (approaches initially and passes finally)

$$\begin{aligned} &TAsm2 \mathrel{\hat{=}} \\ &\quad (\lceil i \notin Act \rceil \longrightarrow \lceil i \in Appr \wedge i \notin Pass \rceil) \\ &\wedge\ (\lceil i \in Appr \wedge i \notin Pass \rceil \longrightarrow \lceil i \in Pass \rceil) \\ &\wedge\ (\lceil i \in Pass \rceil \longrightarrow \lceil i \notin Act \rceil) \end{aligned}$$

3. The last train in a series of trains passes the crossing before leaving the crossing

$$\begin{aligned} &TAsm3 \mathrel{\hat{=}} \\ &\quad (\lceil \neg Active \rceil \longrightarrow \lceil Approaching \wedge \neg Passing \rceil) \\ &\wedge\ (\lceil Approaching \wedge \neg Passing \rceil \longrightarrow \lceil Passing \rceil) \\ &\wedge\ (\lceil Passing \rceil \longrightarrow (\lceil \neg Active \rceil \vee \lceil Active \rceil)) \\ &\wedge\ (\lceil Active \rceil \longrightarrow \lceil Passing \rceil) \end{aligned}$$

4. The trains do not linger if the signal is "go"

$$TAsm4 \mathrel{\hat{=}} \lceil Signal \wedge Active \rceil \Rightarrow \ell \leq Tsched$$

5. The railway tracks are not overloaded with trains

$$\begin{aligned} TAsm5 \mathrel{\hat{=}}\ & \lceil Active \rceil ; \lceil \neg Active \rceil ; \lceil Active \rceil \\ & \Rightarrow \ell > Tinactive + Twait \\ & \qquad + Tgateopen + Topen \end{aligned}$$

The assumptions 1, 2 and 4 are examples of "obvious" parts of train behaviours; but we have to state them explicitly in order to use them in a verification.

Device assumptions

1. It takes at most *Tgateclose* for the gates to close, if the road traffic is not stuck in the RLC

$$\begin{aligned} DAsm1 \mathrel{\hat{=}}\ & \lceil Gates = closing \wedge Traffic \neq stuck \rceil \\ & \Rightarrow \ell \leq Tgateclose \end{aligned}$$

2. It takes at most *Tgateopen* for the gates to open

$$DAsm2 \mathrel{\hat{=}} \lceil Gates = opening \rceil \Rightarrow \ell \leq Tgateopen$$

3. The physical properties of *Gates* constrain the value of *gates* to the cycle: *open*, *closing*, *opening*, *open* (in that order)

$$\begin{aligned} &DAsm3 \mathrel{\hat{=}} \\ &\quad (\lceil Gates = open \rceil \longrightarrow \lceil Gates = closing \rceil) \\ &\wedge\ (\lceil Gates = closing \rceil \longrightarrow \lceil Gates = closed \rceil) \\ &\wedge\ (\lceil Gates = closed \rceil \longrightarrow \lceil Gates = opening \rceil) \\ &\wedge\ (\lceil Gates = opening \rceil \longrightarrow \lceil Gates = open \rceil) \end{aligned}$$

4. The signal switches between "stop" and "go"

$$\begin{aligned} DAsm4 \mathrel{\hat{=}}\ & (\lceil \neg Signal \rceil \longrightarrow \lceil Signal \rceil) \wedge \\ & (\lceil Signal \rceil \longrightarrow \lceil \neg Signal \rceil) \end{aligned}$$

4.2 Control strategy

The control strategy is selected by the system designer with the purpose of defining an implementable control satisfying the requirements under the assumptions. The following strategy is a formalisation of the obvious finite state control, cycling through phases with passive, approaching and passing trains. This is expressed by the predicate

$$RCS \mathrel{\hat{=}} \Box(\bigwedge_{i=1}^{7} RCS\,i)$$

Approaching trains

1. The gates will remain open when no trains are present

$$\begin{aligned} RCS1 \mathrel{\hat{=}}\ & (\lceil \neg Active \rceil \wedge (\lceil Gates = open \rceil ; true)) \\ & \Rightarrow \lceil Gates = open \rceil \end{aligned}$$

2. If trains are present, the gates are open for at most *Treact*

$$RCS2 \mathrel{\hat{=}} \lceil Gates = open \wedge Active \rceil \Rightarrow \ell \leq Treact$$

3. It takes at most *Tnts* before the signal is "go" when the gates have closed

$$\begin{aligned} RCS3 \mathrel{\hat{=}}\ & \lceil (Gates = closed) \wedge \neg Signal \wedge Active \rceil \\ & \Rightarrow \ell \leq Tnts \end{aligned}$$

Passing trains

4. The gates remain closed as long as the signal is "go"

$$RCS4 \mathrel{\widehat{=}} \lceil \mathit{Signal} \rceil \Rightarrow \lceil \mathit{Gates} = \mathit{closed} \rceil$$

5. The signal remains "go", while trains are present

$$\begin{array}{l} RCS5 \mathrel{\widehat{=}} (\lceil \mathit{Active} \rceil \wedge (\lceil \mathit{Signal} \rceil ;\ \mathit{true})) \\ \qquad \Rightarrow \lceil \mathit{Signal} \rceil \end{array}$$

6. The signal will only indicate "go" for at most *Tinactive* after the trains have left

$$RCS6 \mathrel{\widehat{=}} \lceil \neg \mathit{Active} \wedge \mathit{Signal} \rceil \Rightarrow \ell \leq \mathit{Tinactive}$$

7. The gates will remain closed for at most *Twait* after all trains have left

$$\begin{array}{l} RCS7 \mathrel{\widehat{=}} \lceil (\mathit{Gates} = \mathit{closed}) \wedge \neg \mathit{Active} \wedge \neg \mathit{Signal} \rceil \\ \qquad \Rightarrow \ell \leq \mathit{Twait} \end{array}$$

4.3 Verification of requirements

In order to verify that the formal specification of the Control System satisfies the requirements, one must prove: $ASM \wedge RCS \Rightarrow Req$.

The verifications all have the same structure: From an *arbitrary* interval, where the antecedent part of the requirement holds, it is shown through several steps of deduction that the consequent holds. The assumptions and the control strategy can be used freely in the deduction, since they all hold for an arbitrary interval (the $\Box$ distributes over conjunction). Since it is shown to hold for an arbitrary interval it also holds for any subinterval.

The safety requirement

The safety requirement is verified without any timing constraints

$$\begin{array}{lll} & \lceil (\mathit{Gates} \neq \mathit{closed}) \vee (\mathit{Traffic} = \mathit{stuck}) \rceil & \{\text{Traffic}\} \\ \Rightarrow & \lceil (\mathit{Gates} \neq \mathit{closed}) \vee (\mathit{Traffic} \neq \mathit{stopped}) \rceil & \{\text{RTAsm2}\} \\ \Rightarrow & \lceil (\mathit{Gates} \neq \mathit{closed}) \vee (\mathit{Gates} = \mathit{open}) \rceil & \{\text{Gates}\} \\ \Rightarrow & \lceil (\mathit{Gates} \neq \mathit{closed}) \vee (\mathit{Gates} \neq \mathit{closed}) \rceil & \\ \Rightarrow & \lceil \mathit{Gates} \neq \mathit{closed} \rceil & \{\text{RCS4}\} \\ \Rightarrow & \lceil \neg \mathit{Signal} \rceil & \{\text{TAsm1}\} \\ \Rightarrow & \lceil \neg \mathit{Passing} \rceil & \end{array}$$

The functional requirements

The proof of the first functional requirement ensuring the flow of the road traffic relies on the maximum opening and closing times for the gates, and the calculated maximum time the gates can be closed:

$$\begin{array}{lll} & \lceil \mathit{Traffic} = \mathit{stopped} \rceil & \{\text{RTAsm2}\} \\ \Rightarrow & \lceil \mathit{Gates} \neq \mathit{open} \rceil & \{\text{DAsm3}\} \\ \Rightarrow & (\lceil \mathit{Gates} = \mathit{closing} \rceil \vee \lceil\rceil); & \\ & (\lceil \mathit{Gates} = \mathit{closed} \rceil \vee \lceil\rceil); & \\ & (\lceil \mathit{Gates} = \mathit{opening} \rceil \vee \lceil\rceil) & \{\text{RTAsm2}\} \\ & & \{\text{DAsm1}\} \\ \Rightarrow & (\ell \leq \mathit{Tgateclose} \vee \lceil\rceil); & \\ & (\lceil \mathit{Gates} = \mathit{closed} \rceil \vee \lceil\rceil); & \\ & (\lceil \mathit{Gates} = \mathit{opening} \rceil \vee \lceil\rceil) & \{\text{Lemma}\} \\ \Rightarrow & (\ell \leq \mathit{Tgateclose} \vee \lceil\rceil); & \\ & (\ell \leq \mathit{Tclosed} \vee \lceil\rceil); & \\ & (\lceil \mathit{Gates} = \mathit{opening} \rceil \vee \lceil\rceil) & \{\text{DAsm2}\} \\ \Rightarrow & ((\ell \leq \mathit{Tgateclose} \vee \lceil\rceil); & \\ & (\ell \leq \mathit{Tclosed} \vee \lceil\rceil); & \\ & (\ell \leq \mathit{Tgateopen} \vee \lceil\rceil)) & \{\text{DurCalc}\} \\ \Rightarrow & \ell \leq \mathit{Tgateclose} & \\ & \quad + \mathit{Tclosed} + \mathit{Tgateopen} & \end{array}$$

where we have used the following lemma:

The time that the gates are closed is limited by a sum of the time the gates are closed before the signal indicates "go", the active time with the signal on "go", the time the signal is "go" after the trains have left the crossing, and finally the time the gates can be closed while the system is inactive

$$\mathit{Lemma} \mathrel{\widehat{=}} \Box(\lceil \mathit{Gates} = \mathit{closed} \rceil \Rightarrow \ell \leq \mathit{Tclosed})$$

where $\mathit{Tclosed} = \mathit{Tnts} + \mathit{Tsched} + \mathit{Tinactive} + \mathit{Twait}$

Thus, we should choose *Tstopped* such that:

$$\mathit{Tgateclose} + \mathit{Tclosed} + \mathit{Tgateopen} \leq \mathit{Tstopped}$$

The second and third functional requirements are proved in a similar manner.

5 Refining the control model

The control model presented above (from now on referred to as RCS_0) specified a set of properties of the control system. This control model can be *refined* to introduce a less abstract control model (RCS_1), a *design*. In this model, the control system is defined by describing the communication between a controlling processor and some interface units.

5.1 Informal description

The refined control model comprises:

- A controlling processor
- A unit monitoring the positions of the trains
- A unit controlling the train signal
- A unit controlling the gates
- A unit monitoring the gates

The communication *history* between the processor and the units is captured by extending RCS_0 with a state variable

$$tr : \mathit{Event}^*$$

which contains a finite *trace* (i.e. list) of events. An event occurring at time t is appended to the previous value of tr. Hence, events occur at the points in time where the value of tr changes, and tr holds the history of events occurring prior to or at time t.

A detailed description of the refinement can be found in [15].

5.2 Correctness of refinement

The control strategy RCS_1 is a correct refinement of the original control strategy RCS_0 if

$$RCS_1 \Rightarrow RCS_0$$

Since RCS_0 together with the assumptions has been shown to satisfy the requirements, it follows that

$$RCS_1 \wedge ASM \Rightarrow Req$$

RCS_0 is a conjunction of 7 formulas: $RCS1, \ldots, RCS7$. Verifying that RCS_0 holds whenever RCS_1 holds, is therefore done by separately verifying each of the conjuncts. The complete calculations can be found in [15].

6 Conclusion

We have illustrated an approach to requirements engineering for real-time systems using a mathematical specification of system requirements and system design, where a design is verified by calculations. In the first stage a system model is built. Requirements for the system together with assumptions for the system environment are stated. A control model is then constructed, and it is proved that the control model satisfies the requirements whenever the assumptions hold. Design for a particular control system is then introduced and verified to preserve the properties of the control model. The end result is a set of *verified* specifications, which clearly expresses the responsibilities of component implementors.

In further refinement steps towards concrete programs, we need to know the relationship between programming language semantics and duration formulas. In [3] the Duration Calculus is used to give a real-time semantics to communicating sequential processes and also gives various specifications of schedulers for shared processors. Furthermore, the Duration Calculus has in [7] been used to give semantics to circuits, to prove the correctness of a circuit transformation, and to give a precise definition of delay-insensitive circuits. Current work is going on to extend Duration Calculus to a version with probabilities [8] as well as to mechanise the calculations using existing theorem provers.

References

[1] Dines Bjørner: A ProCoS Project Description, ESPRIT BRA 3104, *EATCS Bull., No 39, October 1989.*

[2] Zhou Chaochen, C.A.R. Hoare, A.P. Ravn: A Calculus of Durations, *Information Processing Letters*, 40(5), 1992.

[3] Zhou Chaochen, Michael R. Hansen, A. P. Ravn: Duration Specifications for Shared Processors. In: *Proceedings of Symposium on Formal Techniques in Real-time and Fault Tolerant Systems*, Nijmegen, January 6-10, 1992. (To appear in the LNCS series.)

[4] A.M. Davis and P.A. Freeman: Guest Editors' Introduction: Requirements Engineering, *IEEE Trans. Software Eng., 17*, March 1991, p. 210-211.

[5] John Dawes: *The VDM-SL reference guide*, Pitmann, 1991.

[6] A.G. Hamilton: *Logic for Mathematicians*, Cambridge University Press, Revised Edition, 1988.

[7] M. R. Hansen, Zhou Chaochen and Jørgen Staunstrup: *A Real-Time Duration Semantics for Circuits.* ProCoS Rep. ID/DTH MRH 7/1, September 1991.

[8] Zhiming Liu: *A Probabilistic Duration Calculus.* Working Paper, Department of Computer Science, Technical University of Denmark, December 1991.

[9] Ben Moszkowski: A Temporal Logic for Multilevel Reasoning about Hardware, *IEEE Computer*, vol. 18, no. 2, 1985, pp. 10-19.

[10] J. Nordahl: *Requirements Specification for a Railway Level Crossing*, ProCoS Note ID/DTH JNO 2, Feb. 1990.

[11] K.M. Hansen, A.P. Ravn, H. Rischel: Specifying and Verifying Requirements of Real-Time Systems, Proceedings of the ACM SIGSOFT '91 Conference on Software for Critical Systems, New Orleans, December 4-6, 1991, *ACM Software Engineering Notes*, vol. 15, no. 5, pp. 44-54, 1991.

[12] David G. Luenberger: *Introduction to Dynamic Systems. Theory, Models & Applications*, Wiley, 1979.

[13] A.P. Ravn, H. Rischel and V. Stavridou: Provably Correct Safety Critical Software, in *Proceedings of IFAC SafeComp'90*, London, England, Oct. 1990.

[14] A.P. Ravn, H. Rischel: Requirements Capture for Embedded Real-Time Systems, *Proceedings of IMACS-MCTS Symp.*, Villeneuve-d'Asq, France, May 1991, Vol. 2, p. 147-152.

[15] J. U. Skakkebæk: *Development of Provably Correct Systems*, Master's thesis, Dept. of Computer Science, Technical University of Denmark, 1991.