Formal Development and Verification of a Distributed Railway Control System

Anne E. Haxthausen[1] and Jan Peleska[2]

[1] Dept. of Information Technology, Techn. University of Denmark, DK-2800 Lyngby, e-mail ah@it.dtu.dk

[2] BISS, Universität Bremen, P.O. Box 330440, D-28334 Bremen, e-mail jp@informatik.uni-bremen.de

Abstract. In this article we introduce the concept for a distributed railway control system and present the specification and verification of the main algorithm used for safe distributed control. Our design and verification approach is based on the RAISE method, starting with highly abstract algebraic specifications which are transformed into directly implementable distributed control processes by applying a series of refinement and verification steps. Concrete safety requirements are derived from an abstract version that can be easily validated with respect to soundness and completeness. Complexity is further reduced by separating the system model into a domain model describing the physical system in absence of control and a controller model introducing the safety-related control mechanisms as a separate entity monitoring observables of the physical system to decide whether it is safe for a train to move or for a point to be switched.

1 Introduction

The present modernisation of European railway networks raises a large variety of issues related to the design and verification of railway control systems. One of these problems is the question how to design control systems for small local networks that can only operate effectively if the costs for initial installation, operation and maintenance of the control system are low. Today's centralised interlocking systems – at least those which are available in Germany – are far too expensive for such small (possibly privatised) networks. A promising approach is to *distribute* the tasks of train control, train protection and interlocking over a network of cooperating components using the standard communication facilities offered by mobile telephone providers. On the other hand, a distributed control concept also introduces new safety issues that could be disregarded as long as centralised control was applied: First, the new communication medium requires security and reliability mechanisms that were unnecessary for centralised systems transmitting control commands to signals and points over wires. Second, the distribution of a control algorithm over several components raises new design and verification issues, since the concept of a global state space as available in a centralised interlocking system can no longer be implemented.

In this article, we will describe the concept of a distributed railway control system consisting of *switch boxes (SB)*, each one locally controlling a point, and *train control computers (TCC)* residing in the train engines and collecting the local state information from switch boxes along the track to derive the decision whether the train may enter the next track segment. The system concept does not require signals along the track, since the "go/no-go" decisions are performed and indicated in the train control computers. We give an overview over the formal specification and verification of the main control algorithm executed by the distributed cooperating control components. The system is designed to operate on *simple networks*, which means in our context that there are two distinguished destinations A and B, such that at each track segment of the network there is a uniquely defined direction to reach A and B, respectively. Typically, this definition applies to networks which are not highly frequented by trains and connect two main stations with small intermediate stations (Figure 1).

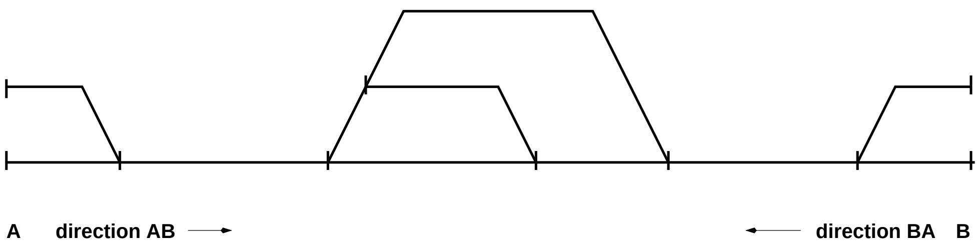

Fig. 1. Simple railway network.

Our specification and verification approach is based on the RAISE formal method and tool set [Rlg92, Rmg95] and follows the *invent-and-verify paradigm*. To address safety issues in a systematic way the standard procedure (see [Sto96]) separating the *equipment under control* – that is, the railway network with its trains – from the *control system* – in our case, the set of TCCs and SBs – is applied. To this end, we first develop abstract algebraic specifications for the *domain model*, i. e., the railway network and the trains to be controlled, and the *safety requirements* stating that the system must not perform a transition into a hazardous state where trains may collide or derailing might occur. These requirements are expressed as conditions about the *observables* of the domain model. Using stepwise refinement and accompanying verification steps, we introduce additional observables that may be monitored by a *controller* giving the "can move/cannot move" conditions for each train and the "can be switched/cannot be switched" conditions for each point. The completeness and consistency of these conditions is verified by proving refinement relations to the higher-level specifications which already have been proved to be consistent with the initial safety requirements. The first stage of the invent-and-verify development ends when the observables of the last refinement needed to control the safety of train movements and point switching are implementable in the sense that they can be transformed into a concrete state space that may be conveniently partitioned among a set of distributed cooperating processes. The second stage specifies and verifies the concrete – i.e., implementable – distributed *controller model* by introducing communicating processes which represent train control computers and switch boxes. The TCC processes collect state information from the SB processes to make the "can move/cannot move" decisions. The SB processes store the relevant state information to take the "can be switched/cannot be switched" decisions for their local points. The resulting controller is a nondeterministic distributed program. The nondeterministic process constructs indicate where application-dependent control decisions – like defining the order in which trains may pass along a single-track section – can be made without violating the safety requirements.

The work presented here originated from a collaboration of the authors with INSY GmbH Berlin, who developed the distributed systems design described in the next section for their railway control system RELIS 2000 designed for local railway networks. In this collaboration, the authors focus on the generalisation and verification of the control concepts used in RELIS 2000. Furthermore, the second author is cooperating with South African Railways in the field of development, verification, validation and test of safety-critical systems.

In Section 2, we introduce the general concept for the distributed railway control system discussed in this article. Section 3 presents the formal specification of the system's domain model. In Section 4, an abstract version of the safety requirements is introduced. The subsequent sections are concerned with the development of the control system as a series of refinement and verification steps. In the discussion (Section 7) we sketch the more general issues of our concept for the development, verification, validation and test of safety-critical systems.

2 Engineering Concept

In this section, we introduce the technical concept of the distributed railway control system to be formally specified and verified below. The technical concept is based on the RELIS 2000 system of INSY GmbH with generalisations and modifications performed by the authors.

Consider the system configuration depicted in Figure 2. The tasks of train control, train protection and interlocking are distributed on train control computers (TCC) residing in each train $\mathsf{T1}$, $\mathsf{T2}$ and switch boxes (SB) $\mathsf{SB1}$, $\mathsf{SB2}$, $\mathsf{SB3}$, each one controlling a single point, the boundary between two segments (e. g. blocks) of a single track or a railway crossing. The basic principle of the control algorithm is as follows:

- Each switch box stores the local safety-related information in its state space. For example, this information contains the actual state of the traffic lights guarding the railway crossing, whether a train is approaching the switch box or the track segments that are presently connected by the local point. The switch boxes use sensors to detect approaching trains and to decide whether a train has left the critical area close to a point or a crossing.

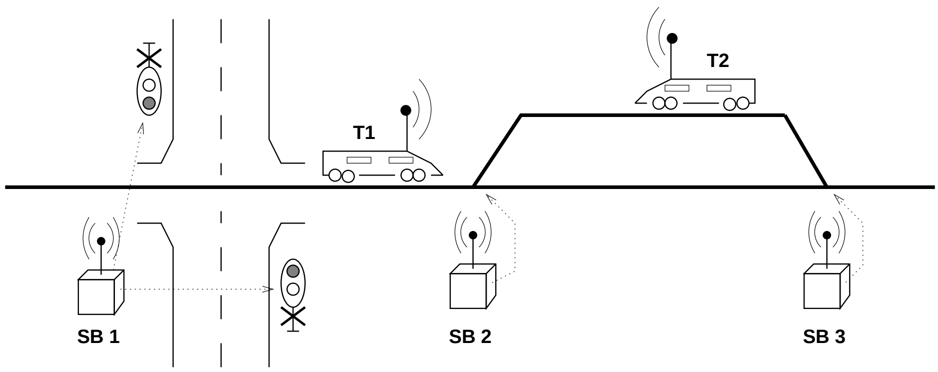

Fig. 2. Distributed railway control system – trains communicating with switch boxes.

- To pass a railway crossing or to enter a new track segment, a train's TCC communicates with the relevant switch boxes to make a request for blocking a crossing, switching a point or just reserving the relevant track segments at the SB for the train to pass. The decision which switch boxes to address is based on the location of the train which is determined by means of the Global Positioning System (GPS) or by using track components signalling their location to the passing train.
- Depending on their local state, the switch boxes may or may not comply with the request received from a TCC. In any case, each SB returns its (possibly updated) local state information to the requesting TCC. After having collected the response from each relevant SB, the TCC evaluates the SB states to decide whether it is safe to approach the crossing or to enter the next track segment.
- For train protection, each TCC blocks the train engine if it is not allowed to leave a station and triggers the emergency brake if the train approaches a railway crossing or enters a new track segment without permission from the associated switch boxes. Furthermore, each TCC monitors the speed of the train and gives warning messages or triggers the emergency brakes if the actual speed exceeds the maximum velocity admitted for the type of train at its actual location in the network.

In the subsequent sections we will focus on the formal specification and verification of the control algorithm concerned with "can move/cannot move" decisions for trains and "can be switched/cannot be switched" decisions for points. To introduce the principles of this algorithm, consider Figure 3 which shows the local state space of two switch boxes *SB1, SB2*.

In state component CONNECTED, the switch box stores which track segments are presently connected by the local point. (If the SB just separates two blocks on a single track, this information is static.) In the components DIR S1, DIR S2,... the directions associated with each track segment are stored: A segment can either be used only for trains going in direction $A \rightarrow B$, or for trains going in direction $B \rightarrow A$ or in both directions ($A \leftrightarrow B$). Typically, this information is fairly static and will only be changed if deviations from the ordinary train schedule occur, for example when constructions are going on or when a train arrives late. As explained below, the segment direction will be evaluated to decide whether a train may reserve a switch box. It is also stored in the TCCs to determine which switch boxes to address in a specific situation. The LOCKED_BY state component indicates whether a specific train has the right to pass the switch box. If such a train is registered in this component, it is impossible to switch the local point to another direction until the train has passed. For the detection of passing trains, a state component SENSOR is activated by a set of sensors attached to the track when a train approaches the point. The component is returned to state "passive" as soon as the sensors indicate that the last waggon of the train has passed the point. To decide whether a train may get a reservation for a segment approaching the switch box and whether a point may be locked for a train, additional state components RES S1, RES S2,... are maintained at each switch box for every track segment whose segment direction is approaching the SB. The ACTION component of the state space is used as a "transaction flag" for commands which have to be executed on several switch boxes in a synchronised manner: The switch box will refuse new commands, as long as the ACTION flag indicates such a transaction. Observe that this flag is unnecessary for the standard reservation commands described next.

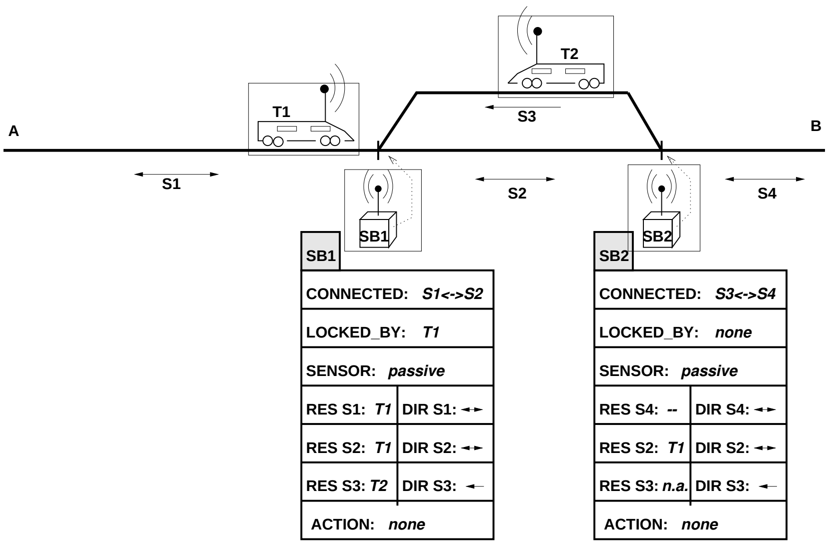

Fig. 3. Switch box state spaces.

To determine, whether a train T1 may enter a new segment S2 (c. f. Figure 3), the train control computer and the relevant switch boxes evaluate the state space described above as follows:

- To guarantee safety for the train at its local position, two conditions must be fulfilled:
  1. The train direction must be consistent with the direction associated with the local track segment. (Train T1 going in direction $A \rightarrow B$ cannot have its position on segment S3, since the latter has associated direction $B \rightarrow A$.)
  2. Each train must have a reservation for its local track segment at the next switch box to be approached by the train (S1 must be reserved for train T1 at switch box SB1).
- To enter the next segment (S2 for train T1), three safety conditions must be fulfilled:
  1. The train direction must be consistent with the direction of the segment to be entered. (S1 has direction $A \leftrightarrow B$, so this is consistent with T1's train direction $A \rightarrow B$.)
  2. The next SB must be locked for the train (SB1 is locked by T1, so this condition is fulfilled for T1).
  3. The train must have a reservation for the next segment S2 at every switch box where S2 is an approaching segment. (In Figure 3, S2 approaches both SB1 and SB2, so T1 must reserve S2 at both switch boxes. In contrast to that, T2 only needed to reserve S3 at SB1 before entering S3 from S4.)
- In order to fulfil these two conditions, the train signals its wish to enter the next segment to the associated switch boxes. Each switch box enters the train's reservation for the next segment if this is not already reserved for another train. If reservation is possible and the SB is not locked by another train, it will switch its point into the required direction if necessary and lock the point for the requesting train.
- If the two conditions are fulfilled the train may enter the next segment. The next SB will delete the lock and all reservations made by the train, as soon as it has passed the SB. (In Figure 3, SB1 will unlock its point and delete all references to T1, as soon as the train has passed the point and entered S2. Note that T1 is still completely safe at its new location, since each train wishing to enter S2 from either S1 or S4 also needs a reservation of S2 at SB2, and this is still blocked by T1.)

In the sections below, this informal system concept is described and verified in a formal way.

3 Domain model

In this section we show (parts of) a domain model capturing those physical objects and events of the uncontrolled railway system which are relevant for the development of the railway control system. We divide the model into a static part and a dynamic (state based) part. Other authors have established similar railway domain models [BGH+97].

3.1 Static part of the model

The static part of the model comprise definitions of data types for objects.

The physical objects we consider include the trains, the points (switch boxes) and the railway network.

Trains

Each train has a unique identification belonging to the following, not further specified type:

type TrainId

Points

Each point has a unique identification belonging to the following, not further specified type:

type PointId

Railway network

A railway network consists of segments connected according to the network topology.

In our model, the network topology is specified by a predicate (*are_neighbours*) which defines which segment ends are neighbours:

type
  Segment, End == a_end | b_end,
  SegmentEnd = Segment × End
value
  are_neighbours : SegmentEnd × SegmentEnd → **Bool**

The *are_neighbours* predicate must satisfy a number of axioms (not presented here) ensuring that the network is directed.

3.2 Dynamic part of the model

As trains move along the segments of the network and points are switched, the *state* of the railway may change over time. We use a discrete, event-based model to describe state transitions.

The state space
At this early phase of development, we do not yet know, what the exact state space is, but only that the state space should contain information about some dynamic properties of objects which we will explain below. Therefore, we just introduce a name for the type of states without giving any datatype representation:

type State

and characterise this type implicitly by specifying state observer functions of the form $obs : State \times \ldots \rightarrow T$ which can be used to capture information (of type T) about the state.

Dynamic properties of trains
Each train has a *position* and a *direction* which may change over time.

We assume that the length of segments is chosen such that any train has a position on one or two neighbouring segments, or it has passed and end point of the network:

type
Position ==
single(seg_of : Segment) | double(fst : Segment, snd : Segment) | error

Since the railway network is directed according to our *simple network* assumption described in the introduction, there are two possible train directions:

type Direction == dirAB | dirBA

We introduce the following functions to observe the mentioned properties:

value /* state observers */
position : State × TrainId → Position,
direction : State × TrainId → Direction

Dynamic properties of points
Points may be switched. Hence, the connections between segment ends of the railway network may change over time. We introduce the following function to observe this:

value /* state observer */
are_connected : State × SegmentEnd × SegmentEnd → **Bool**

Events
We consider the following events:

- trains move from one position to their next position
- points are switched

It should be noted that in this uncontrolled model, events may lead to unsafe states.

For each kind of event we introduce a state constructor which can be used to make the associated state changes:

$$
\begin{array}{l}
\textbf{value} \ /* \text{ state constructors } */ \\
\quad \text{move} : \text{State} \times \text{TrainId} \rightarrow \text{State}, \\
\quad \text{switch} : \text{State} \times \text{PointId} \times \text{SegmentEnd} \rightarrow \text{State}
\end{array}
$$

Their behaviour is defined by observer axioms. For instance, the following axiom states that moving a train does not change how segment ends are connected

$$
\begin{array}{l}
\textbf{axiom} \ /* \text{ observer axioms } */ \\
\quad [\,\text{are\_connected\_move}\,] \\
\qquad \forall\, \sigma : \text{State}, \text{t} : \text{TrainId}, \text{se1}, \text{se2} : \text{SegmentEnd} \bullet \\
\qquad\quad \text{are\_connected}(\text{move}(\sigma, \text{t}), \text{se1}, \text{se2}) \equiv \text{are\_connected}(\sigma, \text{se1}, \text{se2})
\end{array}
$$

and the following axiom states that moving a train affects the position of the train itself:

$$
\begin{array}{l}
\quad [\,\text{position\_move}\,] \\
\qquad \forall\, \sigma : \text{State}, \text{t1}, \text{t2} : \text{TrainId} \bullet \\
\qquad\quad \text{position}(\text{move}(\sigma, \text{t1}), \text{t2}) \equiv \\
\qquad\qquad \textbf{if}\ \text{t2} = \text{t1}\ \textbf{then} \\
\qquad\qquad\quad \text{next\_position}(\sigma, \text{position}(\sigma, \text{t2}), \text{direction}(\sigma, \text{t2})) \\
\qquad\qquad \textbf{else}\ \ \text{position}(\sigma, \text{t2})\ \ \textbf{end} \\
\qquad\quad \textbf{pre}\ \text{safe}(\sigma)
\end{array}
$$

where *next_position(σ, pos, dir)* gives the next position after *pos* in direction *dir*, and *safe* is a function defined in next section.

There are similar observer axioms for switch.

4 Safety requirements

Our goal is to develop a *train control & interlocking system* satisfying the following two safety requirements:

No collision: Two trains must not reside on the same segment.
No derailing: Trains must not derail (by passing an end point of the network or by entering a point from a segment which is not connected with the next segment).

The notion of safety can be formalised by defining a predicate which can be used to test whether a state is safe:

value
safe : State $\rightarrow$ **Bool**
safe(σ) $\equiv$ no_collision(σ) $\wedge$ no_derailing(σ),

no_collision : State $\rightarrow$ **Bool**
no_collision(σ) $\equiv$
($\forall$ t1, t2 : TrainId • t1 $\neq$ t2 $\Rightarrow$
segments(position(σ, t1)) $\cap$ segments(position(σ, t2)) = {}
),

no_derailing : State $\rightarrow$ **Bool**
no_derailing(σ) $\equiv$ ...

5 Development of the railway control system: first stage

The purpose of the railway control system is to prevent events to happen when they may lead to an unsafe state. We develop an implementable controller model by stepwise refinement following the *invent-and-verify paradigm*. The development is divided into two major stages of which we describe the first in this section.

In the first major stage of development we design a full state space keeping information not only about the dynamic properties described in the domain model, but also about new dynamic data (observables) like segment reservations which may be monitored by the controller to evaluate the "can move/cannot move" and "can be switched/cannot be switched" conditions. New data like segment reservations also give rise to new state constructors modelling events like making a reservation.

Our strategy for fulfilling the safety requirements is to invent

1. a *state invariant consistent(σ)*, and
2. for each constructor *con*, a *guard* (condition) *can_con(σ, ...)* which can be used by the controller to decide whether it should allow events (corresponding to application of that constructor) to happen

such that the following *strong safety requirements* are fulfilled:

1. States satisfying the state invariant must also be safe.
2. Any state transition made by a state constructor must preserve the state invariant when the associated guard is true.
3. If the guards for two different events are both true in a state satisfying the state invariant, then a state change made by one of the events must not make the guard for the other event false.

These requirements ensure that if the initial state satisfies the state invariant, and the railway control system only allows events to happen when the corresponding guards are true then the system will stay safe.

The first strong safety requirement can be formalised by the following theory:

[consistent_is_safe]
$\forall\ \sigma : \mathrm{State} \bullet \mathrm{consistent}(\sigma) \Rightarrow \mathrm{safe}(\sigma)$

The second strong safety requirement can be formalised by a theory

[safe_con]
$\forall\ ... \bullet\ \mathrm{consistent}(\sigma) \wedge \mathrm{can\_con}(\sigma,\ ...\ ) \Rightarrow \mathrm{consistent}(\mathrm{con}(\sigma,\ ...))$

for each constructor *con*, and the third strong safety requirement can be formalised by a theory

[safe_con1_con2]
$\forall\ ... \bullet$
$\mathrm{consistent}(\sigma) \wedge \mathrm{can\_con1}(\sigma, \mathrm{x}) \wedge \mathrm{can\_con2}(\sigma, \mathrm{y})$
$\Rightarrow \mathrm{can\_con2}(\mathrm{con1}(\sigma, \mathrm{x}), \mathrm{y})$

for each pair of constructors, *con1* and *con2*.

The state space, state invariant, guards etc. are found by stepwise refinement and verification.

5.1 First specification

The first specification is an abstract, algebraic specification extending the domain model with the following declarations:

value /* state invariant */
consistent : State $\rightarrow$ **Bool**
value /* guards for constructors */
can_move : State $\times$ TrainId $\rightarrow$ **Bool**,
can_switch : State $\times$ PointId $\times$ SegmentEnd $\rightarrow$ **Bool**

As the *State* is not yet explicit, and the set of observers is not complete, we cannot yet give complete explicit definitions of the state invariant and guards. Instead we specify requirements to the guards by implications of the form

axiom /* requirements to guard can_con */
[can_con_implication]
$\forall\ ...\ \mathrm{can\_con}(\sigma,\ ...) \wedge \mathrm{consistent}(\sigma) \Rightarrow\ ...$

and requirements to the state invariant by an implication of the form:

axiom /* requirements to consistent */
[consistent_implication]
$\forall\ \sigma : \mathrm{State} \bullet \mathrm{consistent}(\sigma) \Rightarrow \mathrm{p1}(\sigma)$

We use implications so that we can enrich the requirements in later steps with additional constraints (by changing the right-hand sides of the implications).

5.2 Second to fourth specification

Each of the next three specifications are algebraic and obtained from the previous specification by adding declarations of new observers, state constructors and guards, observer-constructor axioms for new observers and/or constructors and requirement axioms (in form of implications) for new guards. Furthermore, the requirements to the state invariant is enriched in specification number i by refining the *[consistent_implication]* axiom to

axiom /* requirements to consistent */
  [consistent_implication]
    $\forall\ \sigma$: State • consistent(σ) $\Rightarrow$ p1(σ) $\wedge$ ... $\wedge$ pi(σ)

(where $pi(\sigma)$ is a predicate), and the requirements to some of the previous guards *can_con* are refined by making the predicate of the right-hand side of the *[can_con_implication]* axioms stronger.

Below, we give a short survey of which concepts are added in the second to fourth specification.

Second specification
In the second specification, two new concepts are introduced:

- segment registrations for trains, and
- segment directions

The idea is, that a train must only be allowed to move to a segment if it is registered on that segment and if its direction is consistent with the direction of that segment.

Third specification
In the third specification, segment reservations at switch boxes is introduced and segment registrations is defined in terms of that. Furthermore, a concept of locking of points is introduced. The idea is that a train must lock a point in order to pass it, and when a train has locked a point, the point cannot be switched before the train has passed the point.

Fourth specification
In the forth specification, a notion of train routes is introduced, and sensors at the switch boxes sense when trains are passing.

5.3 Fifth specification

Finally, in the fifth specification we are able to define a concrete state space consisting of a state space for each train and a state space for each switch box:

type
  State = (TrainId $\overrightarrow{m}$ TrainState) $\times$ (SwitchboxId $\overrightarrow{m}$ SwitchboxState)

where *TrainState* and *SwitchboxState* are given explicit representations not shown here. The switch box state space was already indicated in figure 3.

With this explicit definition of *State*, it is now possible to replace all axioms with explicit function definitions in terms of functions defined for the two new types *TrainState* and *SwitchboxState*. For instance, the observer function *direction* can be defined as follows

direction : State × TrainId → Direction
direction((σ_t, σ_s), t) ≡ T.direction(σ_t(t)),

where *T.direction* is an observer function defined for train states (of type TrainState), and the state invariant can be given a definition of the form

consistent : State → **Bool**
consistent(σ) ≡ p1(σ) ∧ ... ∧ p5(σ)

5.4 Verification

Implementation relations

In each of the development steps (from specification number i to specification number $i+1$, $i = 1, ..., 4$) above, we have used the RAISE justification tools to prove that the new specification is a *refinement* of the previous specification, i.e. the new specification provides declarations of at least all the types and functions provided by the previous specification, and that all the axioms of the previous specification are consequences of the axioms of the new specification.

Satisfaction of safety requirements

For each of the first four specifications we prove that it is consistent with the strong safety requirements stated in the beginning of this section, and finally for the fifth specification we prove that it fully satisfies these requirements.

The *[consistent_is_safe]* theory is verified to hold already for the first specification. Then, since refinements preserve theories, we know that it also holds for the second to fifth specification.

Verification of the *[can_con]* theories is done stepwise: For specification number i we prove

∀ ... • consistent(σ) ∧ can_con(σ, ...) ⇒ pi(con(σ, ...))

Then, since refinements preserve theories, the fifth specification satisfies

∀ ... • consistent(σ) ∧ can_con(σ, ...) ⇒
    (p1(con(σ, ...)) ∧ ... ∧ p5(con(σ, ...)))

which is equivalent to the *[can_con]* theory, cf. the definition of *consistent* in the fifth specification.

Verification of the *[can_con1_con2]* theories is done similarly.

6 Development of the railway control system: second stage

In the second stage, we develop an implementable model of a distributed railway controller consisting of concurrent communicating processes

value
  controller : State $\rightarrow$ **in any out any Unit**
  controller(σ_t, σ_s) $\equiv$
    (|| { TCC[t].main(σ_t(t)) | t : TrainId})
    ||
    (|| { SB[s].main(σ_s(s)) | s : SwitchboxId})

where $TCC[t].main(\sigma_t(t))$ is a process representing the train control computer in train t, and $SB[s].main(\sigma_s(s))$ is a process representing switch box s. These processes are defined in terms of the guards, state constructors and observers defined in the first major stage, and follow the protocol described in section 2.

7 Discussion

In this article, we have presented the engineering concept and the design and verification of a control algorithm for a distributed railway control system. We consider the following aspects of our work to be the main advantages in comparison to other work that has been performed in the field of design and verification of similar systems:

- Our refinement approach starting with highly abstract algebraic specifications and ending with concrete distributed programs helps to separate general aspects of train control mechanisms and their safety from concrete application-specific design decisions.
- Our verification concept is independent on the size of the underlying network topology. In contrast to that, experiments with model checking have led to unmanageable explosions of the state space, as soon as more complex networks were involved or a larger number of trains had to be controlled.
- Within the restrictions of the *simple network* definition given above, the network topologies covered by our algorithm are fairly general: There are no limits regarding the size of the network, the number tracks involved or the places where points may occur. In contrast to that, approaches using compositional reasoning and structural induction over the underlying network topologies only seem to work for unrealistically simplified networks.
- Starting with a most abstract version of safety requirements, our approach allows to verify their completeness and trace their "implementation" in the more concrete refinements of the abstract control algorithm in a straightforward manner. For approaches defining only implementation-specific safety requirements without reference to a more abstract safety concept, it is nearly infeasible to check safety requirements with respect to completeness.

We would like to emphasise that the control algorithm presented here represents just a building block in a more general approach for the development, verification, validation and test (VVT) of safety-critical systems which is investigated by the authors' research groups at DTU and the Bremen Institute of Safe Systems (BISS). In this wider context, our research work covers

- A systems engineering approach for safety-critical systems which is driven by hazard analysis, risk analysis and a design approach taking VVT issues into consideration right from the beginning of the development life cycle,
- Software-architectures for safety controllers,
- Automated real-time testing for embedded hardware/software components,
- An integrated standardised concept for verification, validation and test of safety-critical embedded controllers, applying combinations of VVT methods, each one optimised for a specific step in the system development life cycle.

References

[BGH+97] D. Bjørner, C.W. George, B. Stig Hansen, H. Laustrup, and S. Prehn. A railway system, coordination'97, case study workshop example. Technical Report 93, UNU/IIST, P.O.Box 3058, Macau, 1997.

[Han96] K. Mark Hansen. *Linking Safety Analysis to Safety Requirements — exemplified by Railway Interlocking Systems.* PhD thesis, Department of Information Technology, Technical University of Danmark, Lyngby, 1996.

[Rlg92] The RAISE Language Group. *The RAISE Specification Language.* The BCS Practitioners Series. Prentice Hall Int., 1992.

[Rmg95] The RAISE Method Group. *The RAISE Development Method.* The BCS Practitioners Series. Prentice Hall Int., 1995.

[Sto96] N. Storey. *Safety-Critical Computer Systems.* Addison Wesley, 1996.