MODELLING RAILWAY INTERLOCKING SYSTEMS

Morten P. Lindegaard, Peter Viuf, and
Anne E. Haxthausen

Dept. of IT, Techn. University of Denmark, DK-2800 Lyngby
fax: +45-45-884530, phone: +45-45-933332,
e-mail: {mpl,pv,ah}@it.dtu.dk

Abstract: In this paper we present a formal model of railway interlocking systems following a protocol based on train routes. The model is divided into one part describing the physical system and another part describing the control mechanisms monitoring observables of the physical system. The safety requirements are formalised at a high level of abstraction and it is then verified that the protocol (concrete safety requirements) ensures safety. *Copyright © 2000 IFAC*

Keywords: Formal methods, RAISE, verification, safety, railways, interlocking systems.

1. INTRODUCTION

The task of railway interlocking systems is to control train traffic in such a way that dangerous situations like train collisions and derailments do not occur. Failure to do so may have severe consequences, and it is therefore important that interlocking systems do not contain design flaws. Formal methods can be used to ensure correctness of the design.

In this paper we report on a project the goal of which is to formally prove that interlocking systems following a certain protocol satisfy the safety requirements. The considered interlocking systems resemble the computer-based interlocking systems used by the Danish State Railways (DSB). The approach we have taken is to (i) establish a model of the traffic and the interlocking system, (ii) formally state what is required for the traffic to be safe, and (iii) prove that these safety properties are invariants of the entire system consisting of the traffic and the interlocking system. We have used the RAISE formal method (RAISE Language Group, 1992; RAISE Method Group, 1995) to develop the model which is expressed in the RAISE Specification Language.

In Section 2 we introduce the concept of train route based interlocking systems treated in this paper. Then in Section 3 we present a formal model of the uncontrolled system and in Section 4 we specify the safety requirements. Next, in Section 5, we model the control system (the interlocking system and protocols to be followed) and in Section 6 we explain our verification strategy. Further development of the system is outlined in Section 7. Finally, in Section 8, we will discuss the work presented in this paper.

2. THE CONCEPT OF TRAIN ROUTE BASED INTERLOCKING SYSTEMS

In this section, we introduce the concepts of train route based interlocking systems for stations.

2.1 *Equipment at a Station*

Train route based interlocking systems use various track-side equipment to monitor and control trains.

Train isolations The railway tracks are divided into electrically isolated segments also known as isolations. The purpose of the isolations is train detection: It can be detected whether an isolation is occupied by a train or not.

Points Tracks are joined at points which can guide trains onto different tracks depending on the setting of the point.

Signals Signals are placed at borders between isolations and are only visible in one direction. The purpose of signals is to inform the train engineers whether they are allowed to proceed or not.

The interaction between the interlocking system and the track-side equipment at the station is illustrated in Figure 1. The interlocking system can read the state of all the devices, and can change the setting of signals and points.

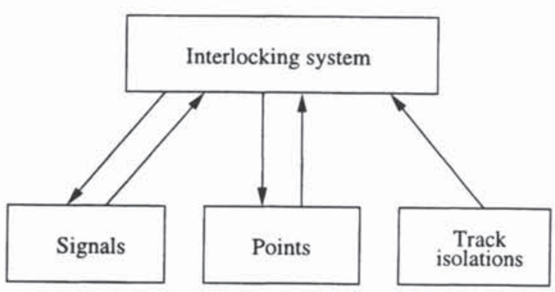

Fig. 1. The interaction between the interlocking system and track-side equipment

2.2 *Interlocking Systems*

The task of an interlocking system is to control traffic so that collisions and derailments are prevented. The interlocking systems considered in this paper are based on a concept of train routes (to be explained in section 2.3) and resemble the computer-based interlocking systems used by the Danish State Railways.

2.2.1. *Internal Image* An internal image of the state of the track-side equipment is kept in the memory of the interlocking system which uses it as basis for decisions. Trains are not explicitly represented in the internal image but are reflected implicitly by their occupation of isolations.

2.2.2. *Control Loop* The interlocking system has a control loop which may be divided into different phases:

(1) Updating the internal image
(2) Collecting requests for train routes
(3) Calculating whether requests for train routes can be served and how signals and points should be set
(4) Setting points and signals

2.3 *Train Routes*

To administrate the access to different areas of the station, the interlocking system uses the concept of train routes.

A *train route* is a path between two signals. Figure 2 shows a small station with ten signals[1]. There is, for instance, a train route from signal A to signal D. There is no explicit train route from signal E to signal B. However, it may be composed from the train route from signal E to signal C and the train route from signal C to signal B.

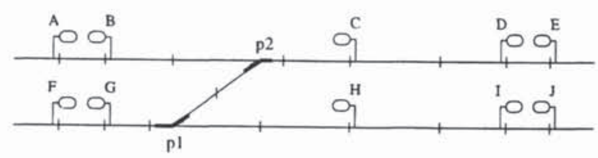

Fig. 2. A small station

2.3.1. *Using train routes* When a train approaches the station, a train route is requested. As an example, if a train is to enter the lower track of the station in Figure 2 from the left, the train route from signal F to signal D can be requested. If the train route is not already reserved by another train and does not overlap with another reserved train route, it is reserved. If points need to be switched for the train to move on the train route, they are switched. For instance, points *p1* and *p2* need to be set as indicated in Figure 2 for a train to move from signal F to signal D. When points are set accordingly, the entry signal of the train route (in this case signal F) is set to green. The signal is set to red when the train has passed it. The reservation of the train route is cancelled when the train leaves it.

A signal is only green when a train route behind it is set (i.e. the train route is reserved and the points and signals are set) and the train has not entered the train route yet. The default setting of signals is red.

2.3.2. *Trains on Train Routes* In this paper we assume that trains are always on train routes. Hence, the reservation of a train route is not cancelled until the train has left the train route. This scheme is different from the one used by the

[1] The way the signals are drawn shows in which direction they are visible. Signal A is for instance only visible when travelling from left to right.

Danish State Railways in which a train route may be cancelled before the train has left it and trains therefore can be outside reserved train routes.

2.3.3. *Safety through Train Routes* To prevent derailments, a train is not given permission to enter a train route before the points have been set accordingly. To prevent collisions, there must be no more than one train on a reserved train route at a time, and overlapping train routes must not be reserved at the same time.

3. MODELLING THE UNCONTROLLED DOMAIN

In this section we show (parts of) a model of the uncontrolled railway domain. We divide the model into a static part and a dynamic (state based) part. Other authors have established similar railway domain models (Montigel, 1992; Bjørner *et al.*, 1997; Hansen, 1996; Hansen, 1998; Haxthausen and Peleska, 1999).

3.1 *Static Part of the Model*

The static part of the model comprise definitions of data types for physical objects of the uncontrolled domain. The physical objects we consider include the railway network, the points, the signals, and the trains.

3.1.1. *The Railway Network* We have chosen a discrete model of the railway network according to which the network is divided into *isolations*. An isolation is either a *linear* piece of the railway track, or it is a *junction* having three ends called the *stem*, the *left* and the *right* branch. A type for representing isolations is declared:

type Isolation

It is assumed that the network is oriented in the sense that there is a global direction *dir*1 (with opposite direction *dir*2) such that on any isolation of the network one can decide what is direction *dir*1. A variant type representing directions is declared:

type Direction == dir1 | dir2

The static topology of the net is defined by the predicate *static_layout*

value
static_layout : Isolation × Isolation → **Bool**

where *static_layout*(i_1, i_2) is true if and only if i_2 is a neighbour of i_1 in the direction *dir1*. The predicate must satisfy a number of axioms ensuring that the network is well-formed.

3.1.2. *Points* Each junction has a *point* which connects the stem end of the junction with the left branch or the right branch. A type for identifiers of points is declared:

type Point

For each junction-isolation, the associated point is obtained by a function *point_of*.

3.1.3. *Signals* A type for identifiers of signals is declared:

type Signal

Signals are placed at borders between isolations and are only visible in one direction. The placement of signals is described by the following function:

value signal_at :
Isolation × Isolation → Signal_option
type Signal_option ==
none_s | some_s(signal : Signal)

where *signal_at(i,i')* is *some_s(s)*, if a signal *s* is visible when travelling from *i* to *i'*, and is *none_s* otherwise.

3.1.4. *Trains* A type for identifiers of trains on the considered railway net is declared:

type Train

3.2 *Dynamic Part of the Model*

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

3.2.1. *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 data type representation:

type State

and characterise this type implicitly by specifying state observer functions of the form *obs : State* × ... → *T* which can be used to capture information (of type *T*) about the state.

3.2.2. *Dynamic Properties of Trains* Each train has a position and a direction which may change over time. Hence, we introduce two observer functions:

value
  position : State × Train → Position,
  direction : State × Train → Direction

The position of a train is modelled as a list of isolations which are occupied by the train:

type Position = Isolation*

3.2.3. *Dynamic Properties of Points* A point has control in either *left* or *right*, or it may be in the process of *switching* to one of the two sides. The setting of a point may change over time and is observed by a function:

value
  pointsetting : State × Point → PointSetting

where *PointSetting* is defined as follows:

type PointSetting ==
    left | right | s_t_left | s_t_right

3.2.4. *Dynamic Properties of Network* An isolation may be occupied or not. The occupation status in a given state can be derived from the position status of the trains in that state:

value
  occupied : State × Isolation → **Bool**
  occupied(σ, i) ≡
    (∃ t : Train • i ∈ **elems** position(σ,t))

Further functions are introduced to calculate from a given point setting how the junction-isolations are connected to their neighbour isolations.

3.2.5. *Dynamic Properties of Signals* A signal is either red or green. The setting of a signal may change over time, and is observed by a function:

value
  signalsetting : State × Signal → SignalSetting

where *SignalSetting* is defined as follows:

type SignalSetting == red | green

3.2.6. *Events* We consider the following events:

- A train enters an isolation.
- A train leaves an isolation.
- A point in control goes into a switching state.
- A switching point reaches control.
- A signal is set to either red or green.

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

value
  move_front : State × Train $\tilde{\rightarrow}$ State,
  move_rear : State × Train $\tilde{\rightarrow}$ State,
  set_point :
    State × Point × PointRequest → State,
  tau_point : State × Point $\tilde{\rightarrow}$ State,
  set_signal :
    State × Signal × SignalRequest → State
type
  PointRequest == left_req | right_req,
  SignalRequest == red_req | green_req

The behaviour of the generators is defined by observer axioms. For each pair of generator and observer, there is an axiom. For instance, the following axiom states that a train leaving an isolation does not affect the pointsetting:

axiom [pointsetting_move_front]
  ∀ σ : State, t : Train, p : Point •
   pointsetting(move_rear(σ, t), p) ≡
    pointsetting(σ, p)

and the following axiom states how the signal setting is changed when a signal is changed:

axiom [signalsetting_set_signal]
  ∀ σ : State, s1, s2 : Signal,
   sreq : SignalRequest •
   signalsetting(set_signal(σ, s1, sreq), s2) ≡
    if s1 = s2 **then**
      case sreq **of**
        red_req → red, green_req → green
      end
    else
      signalsetting(σ, s2)
    end

4. SAFETY REQUIREMENTS

Our goal is to develop a *system* satisfying the following two safety requirements:

No collisions: Two trains must not reside on the same isolation.
No derailments: Trains must not derail.

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 → **Bool**
  safe(σ) ≡ no_collisions(σ) ∧ no_derailments(σ)

The predicate *no_collisions* is defined as

value
  no_collisions : State → **Bool**
  no_collisions(σ) ≡
   (∀ t1, t2 : Train • t1 ≠ t2 ⇒
   elems position(σ,t1) ∩ position(σ,t2) = {})

The predicate *no_derailments* is also defined in terms of state observers so that it is true if and only if it holds for every train that (i) all pairs of adjacent isolations in the position are connected, and (ii) if there is a junction-isolation in the position, the associated point has control in either left or right.

5. MODELLING THE CONTROL SYSTEM

In this section we model the interlocking system and the protocols to be followed.

The protocols prescribe that certain events must only take place when certain conditions are fulfilled. The conditions for an event is modelled as a *guard* of the form *can_gen : State* $\times$ ... $\rightarrow$ *Bool*, where *gen* is the name of the generator modelling the considered event.

5.1 *Protocol for Trains*

The trains are expected to follow a protocol according to which they do not enter a new isolation if there is a red signal at the entrance of that isolation. This condition is formalised as a guard for the *move_front* function. The guard is explicitly defined in terms of various observer functions. Here we just state the signature:

value can_move_front : State $\times$ Train $\rightarrow$ **Bool**

5.2 *The Interlocking System and its Protocol*

The interlocking system serves requests for train routes and set points and signals. To do so, static information about the train routes as well as dynamic information about reserved train routes and the state of track-side equipment is needed.

5.2.1. *Static Information* A type for identifiers of the train routes at a station is declared:

type TrainRoute

The static information of a train route can be retrieved by the following static observers:

value
isolations_of : TrainRoute $\rightarrow$ Isolation*,
direction_of : TrainRoute $\rightarrow$ Direction,
entry : TrainRoute $\rightarrow$ Signal,
exit : TrainRoute $\rightarrow$ Signal,
left : TrainRoute $\rightarrow$ Point-**set**,
right : TrainRoute $\rightarrow$ Point-**set**,
exclusions : TrainRoute $\rightarrow$ TrainRoute-**set**

These observers give the isolations, the direction, the entry signal, the exit signal, and the points which must be set to left, respectively right, control. The last observer gives the set of those train routes which must not be reserved when the considered train route is to be reserved.

A number of well-formedness constraints are imposed on train routes, e.g. the list of isolations given by *isolations_of* must be connected in all the states in which the points in the *left*-set have control in left and the points in the *right*-set have control in right.

5.2.2. *Dynamic Information* Besides the internal image of signal settings, point settings, and occupation status, an interlocking system contains dynamic data about which train routes have been reserved and which train routes have been *opened* for a train by setting the *entry* signal to green. This leads to the following observers:

value
signalimage : State $\times$ Signal $\rightarrow$ SignalImage,
pointimage : State $\times$ Point $\rightarrow$ PointImage,
isolation_occupied : State $\times$ Isolation $\rightarrow$ **Bool**,
reserved : State $\times$ TrainRoute $\rightarrow$ **Bool**,
open : State $\times$ TrainRoute $\rightarrow$ **Bool**

The types *SignalImage* and *PointImage* correspond to the types *SignalSetting* and *PointSetting*.

5.2.3. *Events* In the following, we will describe the events caused by the interlocking system, i.e. the events triggered by statements in the control loop described in Section 2.2.2.

The generators *set_point* and *set_signal* already described in Section 3.2.6 are used by the interlocking system. More generators for updating the internal image and reserving a train route are declared:

value
input_signal : State $\times$ Signal $\rightarrow$ State,
input_point : State $\times$ Point $\rightarrow$ State,
input_isolation : State $\times$ Isolation $\rightarrow$ State,
reserve : State $\times$ TrainRoute $\rightarrow$ State

A number of observer-generator axioms are added so that there is an axiom for each combination of observer and generator.

5.2.4. *Protocol* The interlocking system may always update the internal images of the track-side equipment, so there is no need for guards for the update functions *input_isolation*, *input_signal* and *input_point*. However, the interlocking system is expected to follow a protocol according to which it is only allowed to reserve train routes and set signals and points when certain conditions are met. These conditions are:

- It is only allowed to reserve a train route, if none of the train routes in its exclusion set are reserved.
- It is only allowed to request a signal to be set to red, if it is the entry signal of an open train route. Furthermore, the first isolation of that train route must be occupied.
- It is only allowed to request a signal to be set to green, if it is the entry signal of a reserved train route. Furthermore, all isolations of that train route must be unoccupied, all points within that train route must be in the required setting (as given by the *left* and *right* observers), and the exit signal of that train route must be red.
- It is only allowed to request a point to be set in its left/right position, if the point is in the set obtained by the *left/right* observer of a reserved train route and the setting of the point in the internal image is right/left.

These conditions are formalised as guards for the functions *reserve*, *set_signal*, *set_point*. The guards are explicitly defined in terms of observer functions. Here we just state their signatures:

value
  can_reserve : State × TrainRoute → **Bool**
  can_set_signal :
    State × Signal × SignalRequest → **Bool**
  can_set_point :
    State × Point × PointRequest → **Bool**

6. VERIFICATION STRATEGY

The purpose of the railway control system is to prevent events from happening when they may lead to an unsafe state. In order to verify that our model satisfies the safety requirements described in section 4, we follow the strategy of (Haxthausen and Peleska, 1999) to invent a *state invariant* *consistent*(σ) 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 generator must preserve the state invariant when the associated guard is true.
(3) If the guards for two events for two different system components 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]
  $\forall\ \sigma$: State • consistent(σ) $\Rightarrow$ safe(σ)

The second strong safety requirement can be formalised by a theory

[safe_gen] $\forall$ ... • consistent(σ) $\wedge$ can_gen(σ, ...)
  $\Rightarrow$ consistent(gen(σ, ...))

for each generator *gen*, and the third strong safety requirement can be formalised by a theory typically of the form

[safe_gen1_gen2] $\forall$ ... •
  consistent(σ) $\wedge$ can_gen1(σ,x) $\wedge$ can_gen2(σ,y)
  $\Rightarrow$ can_gen2(gen1(σ,x),y)

for each pair of generators, *gen1* and *gen2*, which are used by different system components (e.g. trains and the interlocking system).

7. FURTHER DEVELOPMENT OF THE SYSTEM

The model presented above is in the form of an abstract, algebraic specification. It does not describe a specific station and interlocking system but a class of systems. The paradigm of stepwise refinement used in the RAISE method is followed to make the specification suited for implementation of prototypes and visualising the model.

7.1 *Explicit, Applicative Specification*

The first specification is refined by giving the type *State* a concrete type definition:

type State =
  (Train $\xrightarrow[m]{}$ TrainState) ×
  (Signal $\xrightarrow[m]{}$ SignalSetting) ×
  (Point $\xrightarrow[m]{}$ PointSetting) ×
  ISState

where *TrainState* and *ISState* denote the state of a train and the interlocking system, respectively.

Furthermore, the observer axioms are replaced with explicit function definitions. For instance, the function *signalsetting* can be defined as follows

value
  signalsetting : State × Signal → SignalSetting
  signalsetting((tm, sm, pm, iss), s) ≡ sm(s)

It is verified that the new specification is an *implementation* of the previous specification, i.e. the definitions of observer and generator functions in the new specification satisfy the axioms of the previous specification.

7.2 *Specification of the Control Loop*

When the applicative functions have been transformed into imperative functions, we describe the behaviour of system components in terms of imperative versions of guards and generators. Generators are only applied when their guards are true. I.e. they occur in expressions of the form:

if *can_gen* **then** *gen* **else skip end**

The control loop of the interlocking system consists of a sequence of the expressions of the above form so that the following is done in each *sweep*:

(1) The internal image is updated for all isolations/signals/points by the generators *input_isolation*/*input_signal*/*input_point*
(2) If a train route is requested and the guard *can_reserve* is true, the train route is reserved by the generator *reserve*
(3) All signals that can be set to green/red are set to green/red
(4) All points that can switch control from left/right to right/left, are set to do so

7.3 *Instantiation of the Specification*

To instantiate the specification for a concrete station, concrete types for *Isolation*, *Point*, *Signal*, and *Train* must be given, and the functions describing the network must be explicitly defined. The types must contain identifiers for the physical objects at the station; no more, no less.

As an example, an instantiation of the specification for the small station in Figure 2, Section 2, would among other declarations contain the following:

type Point == p1 | p2

8. DISCUSSION

In this paper we have presented a formal model of train route based railway interlocking systems resembling those used by the Danish State Railways. We divided the model into one part describing the uncontrolled, physical domain consisting of the railway network, points, signals, and trains, and another part describing the control mechanisms including the protocol used by the trains and the interlocking system. We formalised the safety requirements in terms of observables of the physical domain at a high level of abstraction so that they could easily be validated with respect to soundness and completeness. In fact, the safety requirements were formulated without even mentioning signals. It can then be verified that the protocol satisfies the safety requirements.

The interlocking systems used by the Danish State Railways have previously been modelled in (Hansen, 1996; Hansen, 1998). Our work extends that work by including details about the interlocking system, events and protocols.

We used the verification approach of (Haxthausen and Peleska, 1999). However, the railway control system considered in that paper was based on a totally different engineering concept using a distributed protocol.

Acknowledgements The authors would like to thank Kirsten Mark Hansen for providing material about interlocking systems used by the Danish State Railways, and for commenting on our model.

9. REFERENCES

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

Hansen, K. Mark (1996). Linking Safety Analysis to Safety Requirements — Exemplified by Railway Interlocking Systems. PhD thesis. Department of Information Technology, Technical University of Danmark. Lyngby.

Hansen, K. Mark (1998). Formalising railway interlocking systems. In: *Proceedings of Second FMERail Workshop.*

Haxthausen, A. and J. Peleska (1999). Formal development and verification of a distributed railway control system. In: *Proceedings of Formal Methods World Congress FM'99.* Vol. 1709 of *Lecture Notes in Computer Science.* Springer-Verlag. pp. 1546 – 1563.

Montigel, M. (1992). Formal representation of track topologies by double vertex graphs. In: *Proceedings of COMPRAIL'92.* Washington.

RAISE Language Group, The (1992). *The RAISE Specification Language.* The BCS Practitioners Series. Prentice Hall Int.

RAISE Method Group, The (1995). *The RAISE Development Method.* The BCS Practitioners Series. Prentice Hall Int.