University of Surrey

Department of Computing

Combining event-based and state-based modeling for railway verification

Faron Moller
Hoang Nga Nguyen
Markus Roggenbach
Steve Schneider
Helen Treharne

March 19th 2012

Computing Sciences Report

CS-12-02

Combining event-based and state-based modelling for railway verification

Faron Moller[1], Hoang Nga Nguyen[1], Markus Roggenbach[1], Steve Schneider[2], and Helen Treharne[2]

[1] Swansea University, Wales, UK
[2] University of Surrey, England, UK

Abstract. This paper is concerned with the formal modelling of signalling and point control in the domain of railway engineering. Rules for handling interlocking to ensure railway safety and liveness are often intricate and challenging to verify. We develop a CSP||B model taking a "natural modelling" approach, where the models are as close as possible to the domain model, providing traceability and ease of understanding to the domain expert. This leads to a natural separation between the global modelling of the tracks in B, and the CSP encapsulation of the local views of the individual trains following the driving rules. The approach is illustrated through a small case study (Mini-Alvey), and the model provides verification through formal proofs or informative counter example traces if verification fails.

1 Introduction

Formal verification of railway control software has been identified as one of the "Grand Challenges" of Computer Science [9]. But in respect of this challenge, a question has been asked by the community: "Where do the axioms come from?" [5]. Bluntly expressing a view common to the Formal Methods community, Paulson states, "I have seen many pieces of work spoilt by unrealistic models, incorrect axioms or proofs of irrelevant properties" [15]. The modelling of systems, as well as of proof obligations, needs to be *faithful*.

Faithfulness relates to a variety of concerns. Axioms must be traceable to the informal domain description. The model must be formulated in a way that the reader can maintain a clear overview. The model must provide the *right* level of abstraction for the properties to be shown: details that cannot destroy the property should be left out, it must be possible for the properties to hold or to be wrong. The proof steps should have a meaningful interpretation within the original domain (an invariant has a meaning in the real world). The formal model should be accessible to the domain experts (under minimal guidance). Finally, it should be easy to switch between the real world and the modelling world.

Our paper reports on a case study in cooperation with the company Invensys Rail, where railway engineers and computer scientists together undertake the exercise of domain engineering (Section 2) and formal modelling (Section 3). Here, we use CSP||B [16], which combines event-based with state-based modelling. This leads to a "natural modelling" approach, where the models are as close as possible to the domain model, providing traceability and ease of understanding to the domain expert. The verification tool ProB allows us to analyze the obtained model for safety and liveness and provides meaningful counter example traces if appropriate (Section 4). We relate our approach to previous studies (Section 5) carried out in either a purely event-based or in a purely state-based setting.

2 Informal description of the railway domain

Physically, a railway consists of (at least) four different components; see Figure 1. The *Controller* selects routes for trains. The *Interlocking* serves as a safety mechanism with regards to the Controller and, in addition, controls and monitors the Track equipment. The *Track equipment* consists of elements such as signals, points, and track circuits: signals can show green or red (the yellow aspect of a signal is not modelled since we are only interested in whether a train is authorized to enter a section); points can be in normal position (leading trains straight ahead) or in reverse position (leading trains to a different line) and track circuits detect if there is a train on a track. Finally, *Trains* have a driver who determines their behaviour.

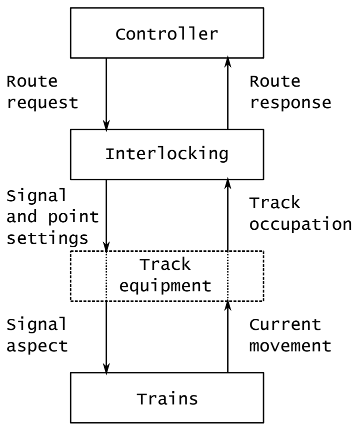


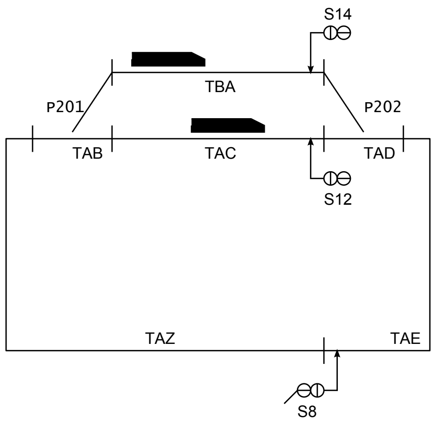
Fig. 1. Information flow.

The Controller can request that the Interlocking sets a specific route. If the requested route is safe to set, the Interlocking will set the points and signals accordingly and will give a positive response to the Controller; should the requested route be unsafe, the response will be negative. In order to determine if it is safe to set a route, the Interlocking considers point positions, semaphores[3] that can lock a point, and the track occupation. For simplicity, in this study we make the following (unrealistic) **Assumption 1:** *Track equipment reacts instantly and is free of defects.* With this assumption, the Interlocking has only to consider the track occupation as an input from the Track equipment. The train driver decides if the train moves forward or halts. Here, so-called *Driving Rules* (formulated by the railway authorities) say that, e.g., *the driver shall stop at a red signal, and may proceed at a green signal.* The driver can decide to halt at a green signal, e.g., because there are cattle on the tracks (not unheard

[3] A semaphore is a type of railway signal in this context.

of in Wales). The current movement of the train is observed by the track circuits. Assumption 1 allows us to abstract from the Track equipment layer, which therefore is depicted with a dashed box in Figure 1.

Railways are built according to a *Track plan.* Figure 2 depicts a prominent



Control table

Route	Normal	Reserve	Clear
8B	P201		TAZ TAB TAC
8A		P201	TAZ TAB TBA
12A	P202		TAD TAE
14A		P202	TAD TAE

Release tables

P201	Occupied
8B	TAC
8A	TBA

P202	Occupied
12	TAE
14	TAE

Fig. 2. Mini-Alvey.

example referred to in the literature as the Mini-Alvey track plan [19, 17]. This plan shows various tracks (TAB, TAC, TAD, ...), signals (S8, S12, S14), and points (p201, p202). Notice that points carry two names, e.g., p201 and TAB: as a point, they are in normal or reverse position, whereas as a track circuit, they detect if they are occupied by a train. Signal engineers define which routes shall be possible in a track plan. A *route* is a (directed) path in a track plan which leads from one signal to the next signal; e.g., referring to the tables in Figure 2, the tracks TAZ, TAB, TAC together form route 8B from signal S8 to signal S12. With regards to the interplay between tracks and routes, we make here **Assumption 2:** *The set of routes uses tracks in only one direction.* Many real world track plans have this property; an end station, however, violates it. In Figure 2 we assume that trains can move only in a clockwise direction and thus Assumption 2 holds. Control tables define when an interlocking is allowed to set a route. They define point positions and which tracks need to be free. For route 8B, for instance, it is required that point 201 is in normal position and that all of the tracks TAZ, TAB, and TAC are clear.

To avoid reductions in journey time and capacity, the points of a route are "released" behind a train. To this end, signalling engineers implement *release tables* in the interlocking. For instance, point p201 on route 8B can be released after a train has completely passed the point (TAB is free) and is on track TAC (TAC is occupied).

The interlocking performs an infinite control loop in three steps: read inputs, process data, and provide outputs. The cycle time of a real world interlocking is in the order of one second. As we can see from Figure 1, the inputs to an interlocking are route requests and track occupation, and its outputs are route responses as well as signal and point settings. We make the following realistic

Assumption 3: *There is at most one route request per cycle.* In its processing phase, an interlocking releases points according to Figure 3, deals with a potential

```
Input: Track occupation
Output: Release locks of points
foreach point p do
    foreach set route R do
        if all tracks are free/occupied as required by release table of p then
            Remove R from locks(p);
    end
end
```

Fig. 3. Release of points.

route request according to the algorithm in Figure 4 and sets all signals to red if

```
Input:  Route request for route R
Output: Route response
if all tracks of R are clear then
    if all points of R are normal/reverse as required by R or free to move then
        move all points of R which are not in the right position;
        for each point p of R: append R to lock(p);
        set the signal of R to "green";
        send "R granted" as route response;
    else
        send "R refused" as route response;
    end
else
    send "R refused" as route response;
end
```

Fig. 4. Route requests.

a condition of their control table is not true. Note that there can be more than one green signal, and a train might not move onto its set route from one cycle to the next. In the Mini-Alvey, however, this is not the case: the routes 8A and 8B as well as the routes 12A and 14A exclude each other; with two trains on different routes, there is as most one green signal possible.

With regards to an interlocking system, *Safety* means:

Collision-freedom There are no collisions.
No-moving-points Trains are not derailed by moving points under them.
No-incorrect-set-points Trains are not derailed by moving onto points that are not set for them.
Correct-speed Trains are not derailed due to too high speed, e.g., in a curve.

Liveness on the other hand means:

Progress It is always the case that some route can be set or a train can move.

3 Event-based and state-based modelling

The information flow shown in Figure 1 suggests that railways should be modelled in an *event-based* way: the controller sends a request message to the interlocking to which the interlocking responds; the interlocking sends signalling information to the trains; and the trains inform the interlocking about their movements. The interlocking serves as the system's clock: messages can be exchange once per cycle.

The control and release tables as shown in Figure 2 as well as their processing described in Figures 3 and 4, however, suggest that railways should be modelled in an *state-based* way: if points are in a certain state and tracks are in the state "clear," then the signal controlling a route can be set to green; if some tracks are "clear" and others are "occupied," then the lock can be removed from a point.

The language CSP||B [16] caters to this hybrid nature of railways and allows one to build a natural mode which is accessible to the domain experts.

3.1 A short introduction to CSP||B

CSP||B [16] provides a way of combining the process algebra CSP [8] with the B-Method [1]. CSP describes *processes* in terms of the patterns of events that they can perform, and provides several semantic models for reasoning about them. The B-Method describes components as *B machines* which manage state and provide operations for querying and changing the state.

CSP||B provides a way of putting B machines and CSP processes in parallel. This is done by synchronising B operations of the form $y \leftarrow a(x)$ with CSP communication events of the form $a!x?y$, and allowing values x and y to pass between them on channels. In particular, the CSP process enforces the order in which operations of the B machine are allowed to occur. By considering B machines as CSP processes, CSP||B treats parallel combinations of CSP processes with B machines in a semantically well-founded way. This approach allows state information to be handled by the B machine, while event behaviour is captured by the CSP process.

3.2 Modelling aims

Our modelling principles aim to achieve:

- a model that is traceable to the informal domain description. This will enable us to review that essential properties of a railway system are not omitted;
- a model that can easily cater for more complex railway systems, e.g., for more complex driving rules;
- a generic architecture for modelling the railway domain that can be instantiated with particular track plans. This will enable us in future to verify general results about railway models;
- events (and operations) in the model that clearly reflect the algorithms of an interlocking cycle. This will ensure that it should be easy to switch between the real world and the model world.

```
$RW\_CTRL = \Box_{r \in ROUTE}(request!r \rightarrow RW\_CTRL)$

$TRAIN\_CTRL(t, currp)$
    $= nextSignal!t?s \rightarrow$
        if $(s == none \ or \ s == green)$
        then $(move.t.currp?newp \rightarrow TRAIN\_CTRL(t, newp)$
                $\Box$
              $stay.t.currp \rightarrow TRAIN\_CTRL(t, currp))$
        else $stay.t.currp \rightarrow TRAIN\_CTRL(t, currp)$
$ALL\_TRAINS$
    $= \Box_{t1 \in TRACK} \ enter.albert.t1 \rightarrow$
        $(\Box_{t2 \in diff(TRACK, exclude(t1))} \ enter.bertie.t2 \rightarrow$
            $(TRAIN\_CTRL(albert, t1) \ ||| \ TRAIN\_CTRL(bertie, t2)))$

$CTRL = (enter?t?p \rightarrow enter?t?p \rightarrow RW\_CTRL) \ _{\ldots}\|_{\ldots} \ ALL\_TRAINS$
```

Fig. 6. CSP control processes for Controller and Trains.

3.3 A CSP||B model

The architecture of our model[4] is depicted in Figure 5. CSP is used to describe the driving rules of trains in order to control their movement and enable the Controller to issue route requests. The B part models the impact of the current movement of trains on the track equipment and focuses on interlocking.

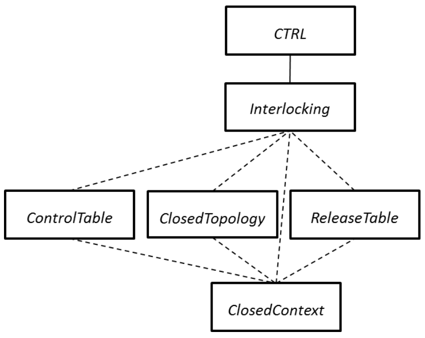


Fig. 5. CSP||B Architecture

Figure 6 illustrates the CSP that uses the *RW_CTRL* process to enable route requests to be made, whereas the *TRAIN_CTRL* process deals with the permissible movement of a train depending on the aspect of the signal that is protecting the track in front of it. It captures the driving rule that a train (and hence a driver) shall stop at a red signal. If there is no signal protecting the track in front, then the train is free to move to the next appropriate track position. As we shall see later, this position will depend on the B model. The process *ALL_TRAINS* enables us to initially place trains on any track in a way that preserves safety. This representation is entirely appropriate for a closed system of tracks. In this paper we focus on simply placing two trains, namely *albert* and *bertie*, after which we can model the servicing of route requests and explore their behaviour.

[4] CSP||B Mini-Alvey model download: http://www.csp-b.org/fm2012-mini-alvey.zip

In the B model the generic architecture of a track plan is represented in three static machines: *ControlTable*, *ClosedTopology* and *ReleaseTable*, which are instantiated for the Mini-Alvey. The associated datatypes for the elements within a track plan are defined in a fourth static machine, *ClosedContext*, e.g., tracks have set definitions as shown in Figure 7. We specifically include the

$$\boxed{\begin{aligned}
TRACKSTATUS &= \{occ, empty\} \quad \wedge \\
ALLTRACK &= \{TAZ, TAB, TAC, TAD, TAE, TBA, nullTrack\} \quad \wedge \\
TRACK &= ALLTRACK - \{nullTrack\} \quad \wedge \\
ASPECT &= \{red, green, none\} \quad \wedge \\
SIGNALSTATUS &= ASPECT - \{none\}
\end{aligned}}$$

Fig. 7. Datatype definitions for tracks defined in *ClosedContext*.

nullTrack in order to be able to identify the derailment of a train.

The *ControlTable* machine splits up modelling the control table into two relations and a function, as shown in Figure 8; these are also instantiated for

$$\boxed{\begin{aligned}
normalTable &\in ROUTE \leftrightarrow POINTS \quad \wedge \\
reverseTable &\in ROUTE \leftrightarrow POINTS \quad \wedge \\
clearTable &\in ROUTE \rightarrow \mathbb{P}(TRACK) \quad \wedge \\
lockTable &\in ROUTE \leftrightarrow POINTS \quad \wedge \\
\ldots \quad \wedge & \\
clearTable &= \{B8 \mapsto \{TAZ, TAB, TAC\}, A8 \mapsto \{TAZ, TAB, TBA\}, \\
&\qquad A12 \mapsto \{TAD, TAE\}, A14 \mapsto \{TAD, TAE\}\}
\end{aligned}}$$

Fig. 8. Control table information defined in *ControlTable*.

a particular track plan, e.g., see the instantiation of clearTable[5]. Additionally, we relate which points would need to be locked for a particular route. The way we have modelled this allows us to distinguish between points being locked by more than one route, which will be useful when modelling railway systems that contain double junctions.

The *ReleaseTable* machine models a relation, shown in Figure 9, which represents the release tables, stating which locks can be released when a particular track is occupied.

$$\boxed{\begin{aligned}
releaseTable &\in TRACK \leftrightarrow (ROUTE * POINTS) \quad \wedge \\
releaseTable &= \{TBA \mapsto (A8, p201), TAC \mapsto (B8, p201), \\
&\qquad TAE \mapsto (A12, p202), TAE \mapsto (A14, p202)\}
\end{aligned}}$$

Fig. 9. Definitions from *ReleaseTable*.

The *ClosedTopology* machine models which signals are associated with a route, the track where a signal is situated, the track where points are situated and relations between tracks and possible successor tracks. For example, Figure 10 illustrates the relation *homeSignal* and instantiates it for the Mini-Alvey

[5] In ProB [12] *8A* is represented as *A8* since ProB does not allow names to begin with a numeral.

$$\begin{aligned}
\mathit{signal} &\in \mathit{ROUTE} \rightarrow \mathit{SIGNAL} \quad \wedge \\
\mathit{signal} &= \{(A8 \mapsto S8), (B8 \mapsto S8), (A12 \mapsto S12), (A14 \mapsto S14)\} \quad \wedge \\
\mathit{homeSignal} &\in \mathit{SIGNAL} \rightarrowtail \mathit{TRACK} \quad \wedge \\
\mathit{homeSignal} &= \{(S12 \mapsto \mathit{TAC}), (S14 \mapsto \mathit{TBA}), (S8 \mapsto \mathit{TAE})\} \\
\mathit{next} &\in \mathit{TRACK} \leftrightarrow \mathit{TRACK} \quad \wedge \\
\mathit{next} &= \{(\mathit{TAZ} \mapsto \mathit{TAB}), (\mathit{TAB} \mapsto \mathit{TAC}), (\mathit{TAB} \mapsto \mathit{TBA}), (\mathit{TAC} \mapsto \mathit{TAD}), \\
&\qquad (\mathit{TBA} \mapsto \mathit{TAD}), (\mathit{TAD} \mapsto \mathit{TAE}), (\mathit{TAE} \mapsto \mathit{TAZ})\}
\end{aligned}$$

Fig. 10. Definitions from *ClosedTopology.*

example stating the signal *S8* is associated with route *8B* and that it is situated on track *TAE*. The topology also provides all possible pairs of tracks and their successor tracks, thus capturing Assumption 2. In the model we provide further distinction by identifying those successor tracks that are static and the dynamic successor tracks that are dependent on the point positions (we omit the details here for brevity).

The *Interlocking* machine is the dynamic machine that captures information about track occupation using sets, and the *pos* function models the location of trains on tracks. This is an important function, as its specification as a partial injective function $\mathit{TRAIN} \rightarrowtail\!\!\!\!\!\!+\; \mathit{TRACK}$ captures the safety requirements: a collision (two trains on the same track) violates the injective nature of this function. Furthermore, a derailment violates the requirement that the range of *pos* is contained in *TRACK*, since a derailment results in a train being on the *nullTrack*, and $\mathit{nullTrack} \notin \mathit{TRACK}$.

The machine also captures information about signal settings using the function *signalStatus* and point settings using the sets: *normalPoints* and *reversePoints*. Additionally, the current locks on points are modelled using *currentLocks* and is a subset of all those that were identified as possible locks in *ControlTopology.* The machine also captures the current information about successor tracks through the function *nextd* which is dependent upon the position of the points and is a subset of *next* from *ControlTopology.* Figure 11 illustrates the invariant

$$\begin{aligned}
&\mathit{emptyTracks} \subseteq \mathit{TRACK} \quad \wedge \\
&\mathit{occupiedTracks} \subseteq \mathit{TRACK} \quad \wedge \\
&\mathit{emptyTracks} \cap \mathit{occupiedTracks} = \emptyset \quad \wedge \\
&\mathit{pos} : \mathit{TRAIN} \rightarrowtail\!\!\!\!\!\!+\; \mathit{TRACK} \quad \wedge \\
&\mathit{signalStatus} \in \mathit{SIGNAL} \rightarrow \mathit{SIGNALSTATUS} \quad \wedge \\
&\mathit{normalPoints} \subseteq \mathit{POINTS} \quad \wedge \\
&\mathit{reversePoints} \subseteq \mathit{POINTS} \quad \wedge \\
&\mathit{normalPoints} \cap \mathit{reversePoints} = \emptyset \quad \wedge \\
&\mathit{normalPoints} \cup \mathit{reversePoints} = \mathit{POINTS} \quad \wedge \\
&\mathit{currentLocks} \in \mathit{ROUTE} \leftrightarrow \mathit{POINTS} \quad \wedge \\
&\mathit{currentLocks} \subseteq \mathit{lockTable} \quad \wedge \\
&\mathit{nextd} \in \mathit{TRACK} \rightarrowtail\!\!\!\!\!\!+\; \mathit{TRACK} \quad \wedge \\
&\mathit{nextd} \subseteq \mathit{next}
\end{aligned}$$

Fig. 11. Invariant from *Interlocking.*

associated with this state. The initial state of the model sets all tracks to being empty, all signals to red, all points to the normal position and no locks are made on points.

This dynamic state and the constant information from all the other machines are then updated and queried, respectively, in the operations of the *Interlocking*. In this model the operations are precisely the events that were identified in the CSP control process *CTRL*. In CSP||B in general, this 1-1 relationship between operations and events need not exist. The algorithm for releasing points is reflected in the *move* operation. We focus on the *request* and *nextSignal* operations in Figures 12 and 13, respectively.

```
bb ⟵ request(route) =
 PRE route ∈ ROUTE
 THEN
   /* are the tracks for a route empty */
   IF ((clearTable(route) ⊆ emptyTracks))
   THEN
     LET unlockedPoints BE
       unlockedPoints = POINTS − ran(currentLocks)
     IN
       /* are all points in right position or unlocked */
       IF ((normalTable[route] ⊆ normalPoints ∪ unlockedPoints)∧
           (reverseTable[route] ⊆ reversePoints ∪ unlockedPoints))
       THEN
         LET np, rp BE
           np = (normalPointsnormalTable[route]) − reverseTable[route]∧
           rp = (reversePointsreverseTable[route]) − normalTable[route]
         IN
           /* move points on the route that need to be moved */
           normalPoints := np ||
           reversePoints := rp ||
           /* for each point p of route, lock p */
           currentLocks := currentLocks ∪ (route ◁ lockTable) ||
           /* set signal of route to green */
           signalStatus(signal(route)) := green ||
           /* grant the request */
           bb := true ||
           /* update topology, next track position might have changed because point changed*/
           nextd := staticNext ∪ dynamicNext[(np ∗ {normal} ∪ rp ∗ {reverse})]
         END
       ELSE
         bb := false   /* refuse request */
       END
     END
   ELSE
     bb := false   /* refuse request */
   END
 END
```

Fig. 12. *request* operation from *Interlocking*.

A route request r is made by the *RW_CTRL* and this becomes the input *route* to the corresponding B operation. The operation is responsible for providing the appropriate boolean route response, *bb*, adopting the algorithm in Figure 4. For example, in a scenario where a request for route $8B$ was being made in the state $\{TAZ, TAB, TAC\} \in emptyTracks$, the conditional would be true and the operation would proceed with the algorithm and signal $S8$ would be set to green. In addition to the route request algorithm itself, the B operation must also keep the current topology of the network up-to-date by updating the *nextd* state.

```
$s \longleftarrow nextSignal(t) =$
 PRE $t \in TRAIN \wedge t \in \mathrm{dom}(pos)$
 THEN
   IF $pos(t) \notin \mathrm{ran}(homeSignal)$
   THEN
     $s := none$
   ELSE
     $s := signalStatus(homeSignal^{-1}(pos(t)))$
   END
 END
```

Fig. 13. *nextSignal* operation from *Interlocking*

The *nextSignal* operation enables the signal aspect controlling the next move of a train, t, which is on a track, to be output as the parameter s, which is provided as an input to the *TRAIN_CTRL* process. For example, suppose *bertie* was on track *TAE* then evaluating $signalStatus(homeSignal^{-1}(TAE))$ returns its current value which will be either *red* or *green*.

4 Verification of CSP||B model

Our CSP||B models can be verified using ProB [12] as it supports B models that are controlled by CSP controllers. In this section we illustrate the use of ProB to verify invariants in order to ensure safety (collision-freedom and no derailment). Invariant violations will enable us to identify errors in control tables and provide meaningful traces to demonstrate this violation. Additionally, we outline how model checking can be used in order to verify liveness. This approach to verification enables us to identify errors in release tables.

4.1 Safety Verification

In the *Interlocking* machine we captured the notion of safety using the *pos* function in an invariant. The following four scenarios illustrate how we may check the model with respect to this invariant.
B model only: Using ProB we performed an invariant check, which means that all states of the *Interlocking* machine are explored. ProB provided the following counter-example trace which leads to violation of the invariant (collision):

$$\langle enter.albert.TAZ, enter.bertie.TAB, move.albert.TAZ.TAB \rangle$$

The B model alone does not place any constraints on train moves and therefore it is no surprise that a collision occurs. In this case, *albert* moved through a red light to collide with *bertie* on *TAZ*.

CSP||B model: The same invariant check was performed on the *Interlocking* as controlled by the *CTRL* resulting in *all* nodes being checked; no invariant violations were found. In this case the train driving rules ensure that collisions do not occur.

CSP||B model with faulty clear tracks for a route in control table: Suppose the control table is adjusted to contain the following mistake (i.e. TAB is omitted from the tracks that should be clear to grant route $8B$):

$$\begin{aligned} \mathit{clearTable} = \{&B8 \mapsto \{TAZ, TAC\}, A8 \mapsto \{TAZ, TAB, TBA\},\\ &A12 \mapsto \{TAD, TAE\}, A14 \mapsto \{TAD, TAE\}\} \end{aligned}$$

Then the following trace is produced automatically as a counter-example:

$$\begin{aligned} \langle &\mathit{enter.albert.TAB}, \mathit{enter.bertie.TAE}, \mathit{request.B8.true},\\ &\mathit{nextSignal.bertie.green}, \mathit{move.bertie.TAE.TAZ},\\ &\mathit{nextSignal.bertie.none}, \mathit{move.bertie.TAZ.TAB}\rangle \end{aligned}$$

This leads to a collision of the two trains on TAB. Since $\mathit{clearTable}(B8)$ does not take TAB into account, the fact that *albert* is already on TAB is ignored. Consequently, as both TAZ and TAC are not occupied, the route is granted, then signal $S8$ is set to green and this enables *bertie* to make the moves which lead to the collision.

CSP||B model with faulty points in control table: If the control table contains a mistake on the directions of points, then this may also impact on safety. For example, suppose $p202$ is the wrong way around in the control table:

$$\begin{aligned} &\mathit{normalTable} = \{B8 \mapsto p201, A14 \mapsto p202\} \quad \wedge\\ &\mathit{reverseTable} = \{A8 \mapsto p201, A12 \mapsto p202\} \end{aligned}$$

Then the check yields the following counter-example trace showing the derailment of *bertie*:

$$\begin{aligned} \langle &\mathit{enter.albert.TAB}, \mathit{enter.bertie.TBA}, \mathit{request.A14.true},\\ &\mathit{nextSignal.bertie.green}, \mathit{move.bertie.TBA.nullTrack}\rangle \end{aligned}$$

This demonstrates a violation of the safety requirement no-incorrect-set-points.

4.2 Towards Liveness Verification

In addition to checking safety, ProB supports the analysis of models with respect to liveness properties expressed either as LTL formulae or as deadlock-freedom checks: that some progress can be made from any reachable state.

In our model, deadlock-freedom is not sufficient to ensure real progress, since the models have some transitions that correspond to no progress: rejected route requests $\mathit{request.r.false}$ or red signals and stationary trains $\mathit{nextSignal.t.red}$ and $\mathit{stay.t.p}$. ProB will report that the model is deadlock-free even when only those events are possible.

To check for the progress property, we must consider the model without those events. To do this, we first block the occurrence of the non-progressing events in the model, and then check for deadlock-freedom. This restriction on the model is captured as a parallel combination on the controller which is expressed as follows:

$$CTRL \;{}_{\ldots}\|_{\{|request.r.false, stay, nextSignal.t.red|\}}\; STOP$$

We check two scenarios with respect to deadlock-freedom under this controller:

CSP||B restricted model: ProB checks *all* states and confirms that the model is deadlock-free. This means that at any stage either a route can be allocated, or a train can progress, which gives the progress property.

CSP||B restricted model with faulty release table: If the release table contains the mistake that $TBA \mapsto (A8, p201)$ is omitted, so the lock $(A8, p201)$ will not be released, then we might expect some additional blocking behaviour. The erroneous released table is as follows:

$$\begin{aligned} releaseTable = \{ & TAC \mapsto (B8, p201), TAE \mapsto (A12, p202), \\ & TAE \mapsto (A14, p202)\} \end{aligned}$$

In this case ProB automatically finds the trace that leads to a deadlock:

$$\begin{aligned} \langle & enter.albert.TAC,\ enter.bertie.TAE,\ request.A8.true, \\ & nextSignal.bertie.green,\ move.bertie.TAE.TAZ,\ request.A12.true, \\ & nextSignal.bertie.none,\ move.bertie.TAZ.TAB,\ nextSignal.Albert.green, \\ & nextSignal.bertie.none,\ move.bertie.TAB.TBA,\ move.albert.TAC.TAD, \\ & nextSignal.albert.none,\ move.albert.TAD.TAE\rangle \end{aligned}$$

This leads to a situation where *albert* is on TAE, *bertie* is on TBA, but no route is available, and both trains are at red lights. In particular, route $B8$ is not available (even though the tracks are clear) as $p201$ is locked with $(A8, p201)$.

4.3 Evaluation

We evaluate our model with respect to the aims listed in Section 3.2. With respect to traceability, the responsibility of the CSP and B is clear: the CSP is responsible for the controller and for the train driving rules, and the B is responsible for the interlocking. The particular track plan is explicit in the four B static machines. The two safety properties checked in this paper are directly represented within B invariants in the model. Verification of the liveness property is achieved as a deadlock-freedom check. It is clear where these checks are performed, providing traceability. The structure of the model is sufficiently generic to allow more complexity in the railway system. More complex driving rules can be captured within the CSP. More complex interlocking rules and track plans can be captured simply by changing the static B machines. The model can also be generalised to handle open track plans (with entry and exit tracks), and crossings. Finally, we see that the events and the operations of the model clearly reflect the information flow of Figure 1 and the interlocking cycle.

5 Related Work

Our work builds upon prior approaches to the modelling and verification of railways. [3, 13] are prominent examples from the B community, [17, 14] are classical contributions from process algebra, [7, 11] use techniques from Algebraic Specification. On a lower abstraction layer [6, 10] verify the safety of interlocking programs with logical approaches. Our modelling is most related to Winter's approach in CSP [18] and Abrial's modelling in Event-B [2]. In the following we briefly discuss their respective approaches and the manner in which we consider our approach to succeed in combining the successful aspects of these whilst avoiding their perceived deficiencies.

5.1 Event-based modelling in CSP

In cooperation with Queensland Rail, Australia, as industrial partner, Winter [18] presents a generic, event-based railway model in CSP as well as generic formulations of the two safety properties **Collision Freedom** and **NoMovingPoints**. Parts of the modelling are demonstrated by taking the track plan of the Mini-Alvey from Figure 2 as an example. If a safety property is violated, the model-checker FDR [4] provides counter-example traces in a format accessible to railway engineers.

Winter distinguishes between the static and dynamic parts of the system. The static part consists of data describing all model-specific elements, namely of a representation of the track plan and of several relations derived from the control tables. The dynamic part consists of CSP processes which encode the general behaviour of trains, points, signals, routes etc. This behaviour is encoded in events which – to a certain extent – appear in the information flow depicted in Figure 1. Control is decentralized. Consequently, testing if conditions hold are encoded via synchronizations: checking if a route can be set is encoded as a synchronization of trains; the driving rule stipulating that a train stops at a red signal is encoded as a synchronization between a train and a signal; etc.

Overall, this leads to a generic architecture and a natural representation of two safety properties. Thanks to the event-based approach it is possible to provide informative counter-example traces. Traceability, however, is limited. There are relations in the model *derived* from the control table. Thus, for example, the driving rule "trains stop at a red signal" is distributed over different parts of the model: it is a consequence of (1) the fact that the event "move to the first track protected by a signal" belongs to a specific synchronziation set and (2) a red signal does not offer this event.The model has no interlocking cycle.

In a follow-up publication [19], railways are modelled with Abstract State Machines (ASM) and the CSP modelling from [18] is dismissed. It is stated that the formal ASM model "is easier to read and understand" than the corresponding CSP model [19]. The counter-examples presented in [19], however, lack the elegance of the previous approach.

5.2 State based modelling in Event-B

Chapter 17 of the book by Abrial [2] gives an excellent detailed description and analysis of the railway domain, deriving a total of 39 different requirements. One basic example of this is SAF-1: “A block be can be reserved for at most one route” (page 519). The presentation – though tailored towards the Event-B method – is in line with our Industrial partner’s view. There are, however, differences when it comes to details of how interlockings are implemented.

The book presents a tower of four refinements of an initial Event-B specification; further refinements needed in order to finalize the model are left out. This tower builds up the railway domain in a step-by-step fashion; e.g., the second refinement introduces readiness for a route, the third speaks about green signals, the fourth introduces points. On each refinement level, various invariants are proven to hold. For example, it is shown that the initial model has the invariant **inv0_3**: $rsrtbl \in resbl \to resrt$, that is, that the controller variable *rsrtbl* representing a reserved route of reserved blocks is a mapping from the set of reserved blocks *resbl* to the set of reserved routes *resrt*. It is then argued that the fact that this mapping is total implies the safety property SAF-1 above.

The modelling approach is generic, even though no concrete model is proven to be correct. Traceability in a tower of specifications can be complex for various reasons. For instance, a requirement can be the consequence of invariants from different levels. The relation between intended properties such as SAF-1 and the model remains an informal one. This is in contrast to other approaches (including Winter’s and ours) which directly represent the intended property in the formal world and then prove that the modelled property is a mathematical consequence of the formal model. Furthermore, the approach is monolithic: behaviour is not attached to the different entities to which they relate. For example, the driving rule “Trains are supposed to stop at red signals” (ENV-13) is modelled together with the rule “A green signal turns back to red automatically as soon as the first block is made occupied” (ENV-15). It is not evident that a control cycle can be identified in this approach. In particular, it appears that train movements and changes of signals are synchronized (see event FRONT_MOVE_1, page 546). This is not the case with interlockings: (1) several trains can move during one cycle; (2) a particularly fast train might actually pass the first track protected by a signal before the interlocking is able to react.

6 Conclusions

Through our association with Invensys Rail, we are working towards deriving railway models which are formal and analysable by current verification technologies, yet are fully faithful; we do not want to hide the engineering understandings held by our industrial partners in clever abstract encodings. Despite being expressed in the mathematical language of formal methods, our models must be immediately understandable – and verifiable – by our industrial partners.

This has proven to be a challenge, as we find that the extant approaches to railway modelling have been hindered in this respect by the framework in which they have been carried out. As explained above, modelling in the railway domain involves event-based components as well as state-based components. Using a solely-event-based framework or a solely-state-based framework succeeds in faithfully representing the relevant components, yet suffers in representing other components through encodings which – whilst clever feats of abstract modelling – are not easily appreciated by the working railway engineer.

In our future work we will develop and verify models based on open track plans and more complex interlocking rules. We will also consider refining the interlocking cycle so that train movement and the releasing of points can be modelled separately; this will involve interface refinement and require an extension to our existing CSP||B theory.

Acknowledgement: The authors would like to thank S. Chadwick and D. Taylor from the company Invensys Rail for their support and encouraging feedback.

References

1. J.-R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.
2. J.-R. Abrial. *Modeling in Event-B*, chapter 17 – Train System. Cambridge University Press, 2010.
3. J. Boulanger and M. Gallardo. Validation and verification of METEOR safety software. In *Advances in Transport Vol 7.* WIT Press, 2000.
4. FDR2. `http://www.fsel.com/software.html`.
5. J. Fiadeiro, 2012. Personal communication.
6. W. Fokkink and P. Hollingshead. Verification of interlockings: from control tables to ladder logic diagrams. In *Proceedings of FMICS'98*, pages 171–185, 1998.
7. A. E. Haxthausen and J. Peleska. Formal development and verification of a distributed railway control system. *IEEE Trans. Software Eng.*, 26(8):687–701, 2000.
8. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
9. R. Jacquart, editor. *IFIP 18th World Computer Congress, Topical Sessions*, chapter TRain: The Railway Domain - A Grand Challenge. Kluwer, 2004.
10. P. James and M. Roggenbach. Automatically Verifying Railway Interlockings using SAT-based Model Checking. In *AVoCS'10.* EASST, 2011.
11. P. James and M. Roggenbach. Designing domain specific languages for verification: First steps. In *ATE-2011.* CEUR-WS.org, 2011.
12. M. Leuschel and M. Butler. ProB: an automated analysis toolset for the B method. *Int. J. Softw. Tools Technol. Transf.*, 10(2):185–203, Feb. 2008.
13. M. Leuschel, J. Falampin, F. Fritz, and D. Plagge. Automated property verification for large scale b models with prob. *Formal Asp. Comput.*, 23(6):683–709, 2011.
14. M. J. Morley. Safety in railway signalling data: A behavioural analysis. In *Proceedings of the 6th International Workshop on Higher Order Logic Theorem Proving and its Applications*, pages 464–474. Springer, 1993.
15. L. Paulson, 2012. Isabelle user-list.
16. S. Schneider and H. Treharne. CSP theorems for communicating B machines. *Formal Asp. Comput.*, 17(4):390–422, 2005.
17. A. Simpson, J. Woodcock, and J. Davies. The mechanical verification of solid-state interlocking geographic data. In *Formal Methods Pacific 97.* Springer, 1997.

18. K. Winter. Model checking railway interlocking systems. *Australian Computer Science Communications*, 24(1), 2002.
19. K. Winter and N. Robinson. Modelling large railway interlockings and model checking small ones. In *Proceedings of the 26th Australasian computer science conference*. Australian Computer Society, Inc., 2003.