The original publication is available at
http://doi.org/10.1007/978-3-319-68499-4_5
In Proceedings of RSSRail 2017: The 2nd International Conference on Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification

Formal modelling techniques for efficient development of railway control products

M. Butler[1], D. Dghaym[1], T. Fischer[2], T.S. Hoang[1], K. Reichl[2], C. Snook[1], and P. Tummeltshammer[2]

[1] ECS, University of Southampton, U.K.
{mjb,dd4g12,t.s.hoang,cfs}@ecs.soton.ac.uk
[2] Thales Austria GmbH
{tomas.fischer,klaus.reichl,peter.tummeltshammer}@thalesgroup.com

Abstract. We wish to model railway control systems in a formally precise way so that product lines can be adapted to specific customer requirements. Typically a customer is a railway operator with national conventions leading to different variation points based on a common core principle. A formal model of the core product must be precise and manipulatable so that different feature variations can be specified and verified without disrupting important properties that have already been established in the core product. Cyber-physical systems such as railway interlocking, are characterised by the combination of device behaviours resulting in an overall safe system behaviour. Hence there is a strong need for correct sequential operation with safety "interlocks" making up a process. We utilise diagrammatic modelling tools to make the core product more accessible to systems engineers. The *RailGround* example used to discuss these techniques is an open source model of a railway control system that has been made available by Thales Austria GmbH for research purpose, which demonstrates some fundamental modelling challenges.

Keywords: Event-B, iUML-B, ERS, Interlocking

1 Introduction

A railway control system is a safety-critical cyber-physical system where common principles are well established and adopted on a broadly generic infrastructure, but with an abundance of feature variations across national boundaries. In order to be able to offer a configurable, yet certified, product it is therefore essential to adopt an efficient product development process that allows a verified core product to be adapted to specific solutions. We propose a model-based approach that will support such a development process.

Motivation Our motivation is to model railway control systems in a formally precise way so that product lines can be adapted to specific customer requirements. Typically a customer is a railway operator with national conventions leading to

different variation points based on a common core principle such as *Interlocking* (IXL). A formal model of the core product therefore must be precise and manipulatable so that different feature variations can be specified and verified without disrupting important properties that have already been established in the core product. Such properties include the safety principles of a technology and we assume that they have already been proven to ensure safety. For example, in our IXL model, we assume that if conflicting paths are exclusively enabled, this is sufficient to ensure that trains do not collide. In future work, we envisage using various *domain-specific languages* (DSLs) so that customers can precisely specify their specific feature requirements. For now, we focus on modelling the core system.

We model the core system using notations that are accessible to systems engineers. These engineers have extensive domain knowledge, are skilled at specifiying systems in the railway domain and usually have experience in semi-formal modelling tools such as UML and SysML. They are generally less experienced at formal modelling and proof. For this reason we utilise graphical representations of the formal model. To fully understand the model, and to debug models when proofs do not discharge automatically, it is necessary to understand the formal notation. Formal methods specialists are needed to help with proof and specific modelling difficulties, but the main content of the models must be accessible to less specialised systems engineers and other stakeholders.

Event-B and extensions The Event-B modelling method [1] is suitable for this formal modelling task because it allows us to verify (core) properties while leaving certain features underspecified, and subsequently refine the model to fully specify those features in a consistent manner with respect to the abstract model. Event-B has strong tool support for verification and validation in the form of theorem provers and model-checkers. Diagrammatic modelling notations and tools are available which aid model accessibility. We use the iUML-B class diagrams and state-machines [19,18,15] in conjunction with *Event Refinement Structure* (ERS) [5,6] to visualise event refinement structures.

RailGround For research and illustration purposes we use an example railway interlocking specification called RailGround [14]. RailGround is provided as an open specification and model for this purpose. This is a simplified version of interlocking systems, built specifically for research on formal validation and verification of railway systems [14]. This example is used as part of the rail use case of the European project *Enable-S3* [4].

Contribution *Cyber-Physical Systems* (CPS) such as the railway interlocking, are made up of many disparate devices with interrelationships. They are characterised by the combination of device behaviours resulting in an overall safe system behaviour. Hence there is a strong need for correct sequential operation with safety "interlocks" making up a process. To model CPS we start by modelling the entity relationships of the devices using an iUML-B class diagram, we then model the individual behaviour of instances of these entities using iUML-B

state-machines. However, this is not sufficient to show the overall system process. To show this we add the ERS view which shows the process based on the sequences of events involved in the interaction of all devices.

Our contribution is an approach for modelling CPS using diagrammatic notations for the three views above: Entity-Relationships, Entity-Behaviour and System Process. Our approach utilises an integration of iUML-B and ERS.

Structure In Section 2 we describe the requirements of the RailGround system and introduce the modelling notations and tools that we use to model it. In Section 3 we describe our model of the RailGround system in order to illustrate the use of the formal modelling notations. Section 4 discusses related work to our approach and the case study. In Section 5 we reflect on the effectiveness and benefits of combining the modelling notations and indicate future work.

Dataset The Event-B model illustrating this paper is available as a dataset here: `https://doi.org/10.5258/SOTON/D0184`. The required Rodin and plug-in configuration is given in a 'readme' file within the dataset.

2 Background

We first present some background information on the case study including its requirements in Section 2.1. Subsequently, we give a brief overview of the Event-B method in Section 2.2, of iUML-B in Section 2.3, and of ERS in Section 2.4.

2.1 RailGround

The example used in this paper is based on RailGround, a formal model of a railway interlocking system using Event-B, which was developed by Thales Austria GmbH [14]. Railway systems, in general, aim at providing a timely, efficient and most importantly a safe train service. This requires a reliable command and control system that ensures a train can safely enter its specified path. In the system under consideration, the railway topology consists of a set of connected elements, which are protected by signals passing information to the trains. The safety of a train is ensured by allowing its path to be set, only if it does not conflict with the current available paths. The following requirements are extracted and simplified from [14]. For illustration, we will consider the network topology with one track and two points as in Figure 1. Note that we focus on modelling the system functional safety here. This is a subset of the overall system safety functionality. In particular, technical measures from other domains to achieve the desired *Safety Integrity Level* (SIL) are not considered.

Railway Topology The railway topology is formed by a set of *Rail Elements*. A rail element is a unit which provides a physical running path for the trains, i.e. rails (e.g. track, points, crossing). A *Rail Connector* is a port of a rail element used to define the element's connectivity via *Rail Segments* as well as to link

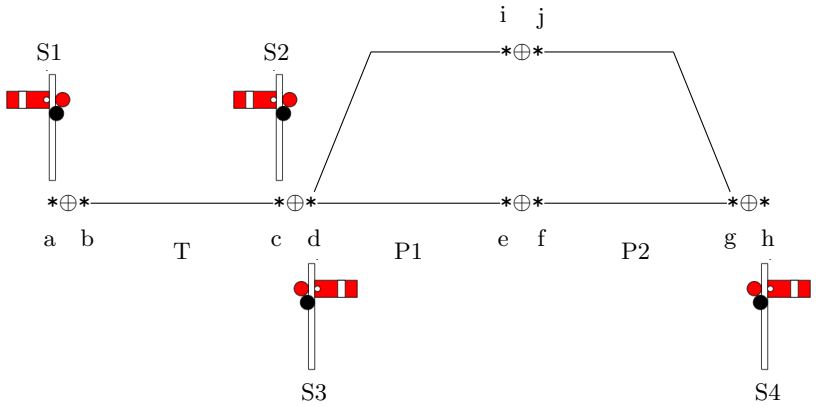


Fig. 1: An example railway topology [14]

this element with the adjacent ones via rail links. Depending on its type, each rail element usually has 2 to 4 rail connectors. Each rail connector belongs to exactly one rail element. Typically, a rail element is made up of one or more *Segments*. A rail segment is a connection from an element's rail connector to another connector of the same rail element.

REQ 1 The network topology is a set of rail elements.
REQ 2 A rail element has 2 to 4 rail connectors. Each rail connector belongs to exactly one rail element.
REQ 3 A rail segment is a connection from a rail connector of some rail element to another rail connector of the same rail element.

In Figure 1, there are three rail elements, namely T (a track), P1, P2 (points). The connectors are a, b, ..., h and associate with the rail elements as follows:

$$\mathsf{T} \mapsto \{\mathsf{b}, \mathsf{c}\},\ \mathsf{P1} \mapsto \{\mathsf{d}, \mathsf{e}, \mathsf{i}\},\ \mathsf{P2} \mapsto \{\mathsf{f}, \mathsf{j}, \mathsf{g}\}.$$

The segments are {bc, cb, di, id, de, ed, jg, gj, fg, gf}. The relationship between the rail elements and the segments are as follows:

$$\mathsf{T} \mapsto \{\mathsf{bc},\mathsf{cb}\},\ \mathsf{P1} \mapsto \{\mathsf{di}, \mathsf{id}, \mathsf{de}, \mathsf{ed}\},\ \mathsf{P2} \mapsto \{\mathsf{jg}, \mathsf{gj}, \mathsf{fg}, \mathsf{gf}\}.$$

Element Positions For each rail element, an *Element Position* is a distinct situation of that rail element. Furthermore, each element position defines the set of possible element connections (defined by segments) for that particular rail element.

REQ 4 For each rail element, there is a set of possible element positions
REQ 5 Each rail element and position correspond to a set of rail segments

For example, a set of points has three possible positions POS_X (in transition), POS_L (left), POS_R (right). Consider the points P1, position POS_X corresponds to an emptyset of segments, POS_L corresponds to segments {di, id}, and POS_R corresponds to segments {de, ed}.

Paths A path is a sequence of rail segments, with the constraint that two rail segments of the same rail element are not allowed within one path. A path can be activated so that trains are allowed to be on that path.

REQ 6 A path is a sequence of rail segments.
REQ 7 Two rail segments belonging to the same element are not allowed within one path.

Consider the example in Figure 1, a path could be the following sequence of segments [bc, di, jg], or [gf, ed, cb].

Path Life-Cycle A set of paths are pre-defined in the network. Before becoming active, a path must be requested. As soon as all conditions for the path (e.g., rail elements must be in the required position to establish its path), a requested path can be activated. As a train moves along a path, rail elements that are no longer in use can be released. An active path can be removed only after all its rail elements are released. A rail element position can only be changed if the rail element is not part of an active path.

REQ 8 A requested path can become an active path when all conditions for that path are met
REQ 9 An active path can be removed only after all its rail elements are released.
REQ 10 A rail element position can only be changed if it is not part of an active path.

In the example network topology, we can have the following paths R1–R4, with the following associations:

$$\mathsf{R1} \mapsto [\mathsf{bc}, \mathsf{de}, \mathsf{fg}], \mathsf{R2} \mapsto [\mathsf{bc}, \mathsf{di}, \mathsf{jg}], \mathsf{R3} \mapsto [\mathsf{gf}, \mathsf{ed}, \mathsf{cb}], \mathsf{R4} \mapsto [\mathsf{gj}, \mathsf{id}, \mathsf{cb}].$$

Vacancy Detection In order to detect trains on the network, the system is equipped with *Track Vacancy Detection* (TVD). Each segment belongs to exactly one TVD section. A TVD section is either vacant or occupied. A TVD section is occupied if there is some train on some segment belonging to that TVD section.

REQ 11 Each segment belongs to exactly one TVD section.
REQ 12 A TVD section can be either in vacant or occupied state.

Signals A *Signal* is an entity capable of passing information to trains. A signal is associated with a rail element for a particular traversal direction. A signal aspect is an (abstract) information conveyed by a signal. *Signal Default* is a predefined aspect of signals. Trains are assumed to obey the signals, in particular, stop at a signal containing default aspect.

REQ 13 A signal may be set to an aspect other than default, only if there is an active element after this signal.

REQ 14 A signal is associated to a connector and hence to a specific location within the topology, i.e., the information passed by the signal are only valid to a specific direction which in this case will be the segment starting at the signal connector.

In Figure 1, we have 4 signals, S1–S4 associated with different connectors as follows.

$$S1 \mapsto a,\ S2 \mapsto c,\ S3 \mapsto d,\ S4 \mapsto h$$

Safety Properties Safety in this model is ensured by the paths which are active. The paths can only be set if all its elements are in the right positions. Safety is ensured by preventing paths to be requested if there are other paths requiring the same elements.

REQ 15 Two active paths cannot overlap
REQ 16 An active path must have all its elements in the right positions
REQ 17 A path can be requested if it is disjoint from other active or requested paths.

2.2 Event-B

Event-B [1] is a formal method for system development. Main features of Event-B include the use of *refinement* to introduce system details gradually into the formal model. An Event-B model contains two parts: *contexts* and *machines*. Contexts contain *carrier sets*, *constants*, and *axioms* that constrain the carrier sets and constants. Machines contain *variables* v, *invariants* I(v) that constrain the variables, and *events*. An event comprises a guard denoting its enabling-condition and an action describing how the variables are modified when the event is executed. In general, an event e has the following form, where t are the event parameters, G(t, v) is the guard of the event, and v := E(t, v) is the action of the event[3].

$$e == \textbf{any}\ t\ \textbf{where}\ G(t,v)\ \textbf{then}\ v := E(t,v)\ \textbf{end}$$

A machine in Event-B corresponds to a transition system where *variables* represent the states and *events* specify the transitions. Contexts can be *extended* by adding new carrier sets, constants, axioms, and theorems. Machine M can be *refined* by machine N (we call M the abstract machine and N the concrete machine). The state of M and N are related by a *gluing invariant* J(v, w) where v, w are variables of M and N, respectively. The gluing invariant specifies the consistency between the abstract and concrete machines and must be maintained by the execution of both machines. Intuitively, any "behaviour" exhibited by N can be simulated by M, with respect to the gluing invariant J. Refinement in Event-B is reasoned event-wise. Consider an abstract event e and the corresponding concrete event f. Somewhat simplifying, we say that e is refined by f if

[3] Actions in Event-B are, in the most general cases, non-deterministic [8].

f's guard is stronger than that of e and f's action can be simulated by e's action, taking into account the gluing invariant J. More information about Event-B can be found in [8]. Event-B is supported by *Rodin platform* (Rodin) [2], an extensible toolkit which includes facilities for modelling, verifying the consistency of models using theorem proving and model checking techniques, and validating models with simulation-based approaches.

2.3 iUML-B

iUML-B [15,18,19] provides a diagrammatic modelling notation for Event-B in the form of state-machines and class-diagrams. The diagrammatic elements are contained within an Event-B model and generate or contribute to parts of it. For example a state-machine will automatically generate the Event-B data elements (sets, constants, axioms, variables, and invariants) to implement the states, and contribute additional guards and actions to existing events. iUML-B Class diagrams provide a way to visually model data relationships. Classes, attributes and associations are linked to Event-B data elements (carrier sets, constants, or variables) and generate constraints on those elements. In iUML-B class diagrams, a class represents some set of instances and the class may be used to show relationships with other classes. Usually the set of instances is given by an Event-B data element, but in some scenarios it is useful to construct a set using an expression as the class name.

2.4 Event Refinement Structures

In Event-B, behaviour can be decomposed during refinement into a combination of new and refining atomic events. However, the relationship between the events at different refinement levels is not explicit, for this we use ERS [6,5] diagrams. ERS, is a tree-like structure, inspired by Jackson Structure Diagrams (JSD) [10], that provides a graphical extension of Event-B to represent event decomposition explicitly. In addition to specifying event decomposition, an ERS diagram can explicitly represent control flow. Similar to JSD diagrams, the ordering of events is read from left-to-right. In addition to sequencing ERS provides different combinators that support iteration, choice and different forms of non-deterministic interleaving.

3 RailGround Model Using iUML-B and ERS

In this section we describe our version of the RailGround model which is modelled in iUML-B and ERS. For each refinement level we discuss the iUML-B class diagram and state-machine (where applicable) and then describe the behaviour of that refinement level, including the refinement relationships of events, using an ERS diagram.

The overall ERS diagram of the RailGround model is illustrated in Figure 2. The root of the diagram represents the name of the system and it is parametrised

by p of type PATH to show the possible interleaving of different paths. The different regions represent the different refinement levels. One of the refinement levels (**Rails**) does not change the event refinement structure, hence the second region represents two refinement levels. Events of the RailGround model are represented by the leaf nodes of the tree, where an event connected to its parent by a dashed line is a newly added event, while a solid line represents a refining event which is identified by the keyword **refines** in the Event-B model.

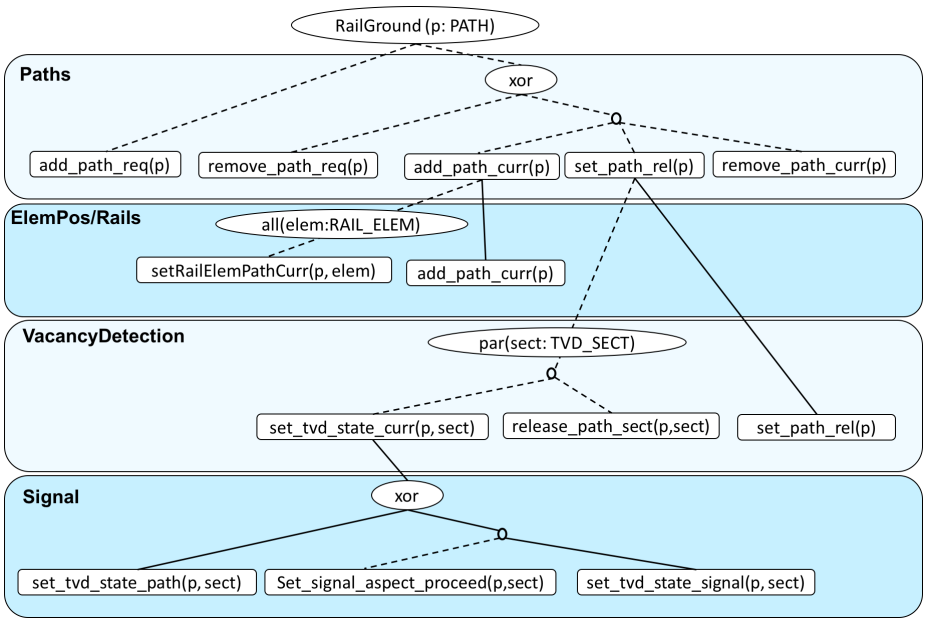


Fig. 2: Event Refinement of the RailGround model shown in ERS
(solid lines: refining events, dashed lines: new events)

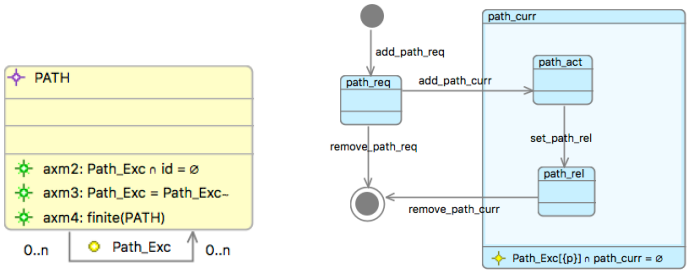
The refinement sequence adopted for the iUML-B model is as follows:

1. Paths - abstract representation of the path of a train through a rail network (REQ 15, REQ 17).
2. ElemPos - positioning of elements in the rail network to put a path in the right state (REQ 1, REQ 4, REQ 8, REQ 9, REQ 10, REQ 16).
3. Rails - connectivity of elements and their organisation into segments (REQ 5, REQ 6, REQ 7).
4. Vacancy detection - the ability of elements in the rail network to detect when they are occupied by a train (REQ 11, REQ 12).
5. Signal - signals that inform trains to stop or proceed through a path (REQ 2, REQ 3, REQ 13, REQ 14)

The reasons for choosing this sequence are

– The exclusive reservation of a path is the primary concept upon which interlocking safety is based. Therefore it is important to model this first when the model is simple, so that it is easier to validate.
– More detail about the operation of paths is introduced next (Element positioning, Rails)
– The occupation of a path is an important concept that can be introduced as soon as paths are sufficiently modelled.
– Signals are a design detail which can vary depending on customer. It is therefore more convenient to introduce this late.

Paths Our first abstract model introduces the notion of paths through a rail network. Paths are reserved for exclusive use ensuring that trains cannot collide. Paths are a conceptual device used by the control system, which are related to a set of physical elements in the railway system. The iUML-B class diagram, Figure 3a, defines a finite given set PATH of paths and an association, Path_Exc of paths that conflict with each other. Axioms constrain this association so that paths do not conflict with themselves and the association is symmetric. The iUML-B state-machine, Figure 3b defines the behaviour of paths. A path is initially requested (add_path_req) and can then be made active (add_path_curr), followed by released (add_path_rel) and then removed (remove_path_curr). Paths that have been made active but not yet removed are called current. This is represented by superstate path_curr which allows us to specify the state invariant that for all current paths, none of their conflicting paths are also current. This is the safety principle of interlocking systems. It is ensured by a guard on add_path_curr. There is also the possibility remove_path_req of un-requesting a path without it ever becoming current.



(a) iUML-B Class Diagram for path properties

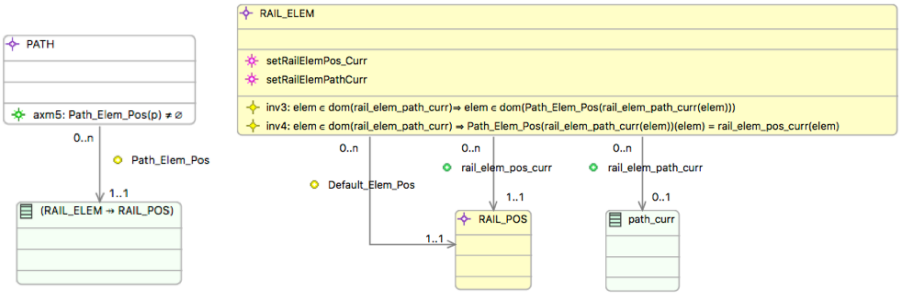
(b) iUML-B State-machine for path behaviour

Fig. 3: Abstract model of paths through a rail network

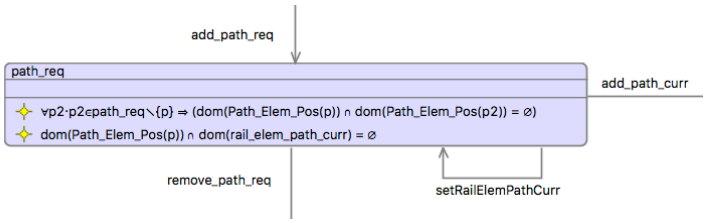
The process involved at this abstract level of the model is represented by the **Paths** region in Figure 2 on page 8. The RailGround model starts by adding a required path, then there is a choice (indicated by *xor*) between either removing the path or making it the current path. If the path p is added as current, then it can be followed by releasing the path, after which the path can be removed from the current paths. At this level, since there is only one (conceptual) device and no refinement, the ERS diagram has a close correspondence with the iUML-B state-machine diagram.

Element position In the first refinement we introduce the idea that elements need to be in a particular position for a path. This corresponds to physical devices such as railway "points". We define a constant function Path_Elem_Pos (Figure 4a) which, for each path, gives a functional mapping from elements to positions. That is, the position that each element of the path needs to be in for that path to be ready. We also define a default position Default_Elem_Pos for each element. Variable functional associations are defined for the current position rail_elem_pos_curr and current path rail_elem_path_curr of each rail element. Two RAIL_ELEM class methods are provided to set the position, rail_elem_pos_curr, of a particular element. Method setRailElemPos_Curr sets the position when the element is not involved in a path, and method setRailElemPath_Curr sets the element to the appropriate position for a given path. The current path of an element is set and reset when the path is made current and released respectively (i.e. these actions are added to the relevant state-machine transitions). Two class invariants are added to class RAIL_ELEM. The first states that, if an element belongs to a current path, then that element must have a defined position for that path according to Path_Elem_Pos. The second ensures that the element is currently in the correct position according to Path_Elem_Pos. Two state invariants in state path_req (Figure 4b) require that the requested path has no elements in common with another requested path and no elements in common with a current path.

The ElemPos region (Figure 2 on page 8), illustrates how the atomicity of adding a current path is broken into two events, the first sets the rail elements position of the current path to the required position, followed by adding the path as current, which is in this case the refining event (solid line). However in order to add the path as current, there is a requirement that all elements of the required path should be in the right position, that is why we apply the *all* combinator adding an additional dimension, elem of type RAIL_ELEM, to the ERS model. The other new event (setRailElemPos_Curr) is not associated with a path and therefore does not appear in the process described by the ERS diagram. The ERS diagram visualises the system level process requirement that the positioning of elements must be completed before the path becomes *current*. This is not apparent in the iUML-B model which focusses on the behaviour of individual devices (PATH and RAIL_ELEM). The fact that setRailElemPath_Curr is a preliminary (stuttering) event leading to add_path_curr is made clear in the ERS diagram. Arguably, this is shown in the iUML-B state-machine by adding setRailElemPath_Curr as a transition on the source state, path_req of add_path_curr



(a) iUML-B Class Diagram for element position



(b) State-invariants for element position

Fig. 4: First refinement introducing element positioning

but the event refinement relationship is not as explicit as in the ERS diagram. On the other hand, the ERS diagram does not illustrate state constraint information such as the requirement that setRailElemPath_Curr is only performed while the associated path is in the state path_req.

Rails In the second refinement we introduce a stronger relationship describing the physical construction of paths using rail segments. To do this we introduce a given set RAIL_SGMT (Figure 5) with a functional association Rail_Sgmt_Elem to RAIL_ELEM. An association Path_Segmt gives the subset of RAIL_SGMT that makes up each path. Class axioms specify various constraints to ensure the new segment representation is consistent with other configuration data.

In this refinement, we extended the context to introduce details about the rails connectivity using segments, which only resulted in changing the models behaviour by adding some invariants and guards to the existing events relating connectivity to the element position. Consequently, there were no changes to the structure and ordering of events, which remains the same as the ElemPos region.

Vacancy Detection In the third refinement (Figure 6) we introduce the detection of trains as they occupy rail segments. A new given set TVD_SECT is introduced to represent TVD sections that can detect when they are occupied. A

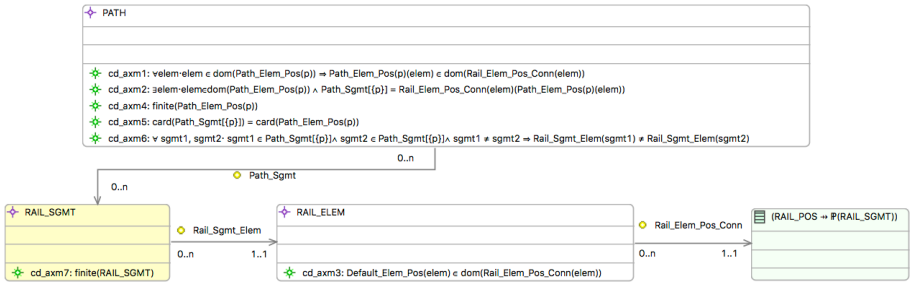


Fig. 5: Second refinement introducing rail segments

many-to-one relationship TVD_Seg_Sect from RAIL_SGMT to TVD_SECT specifies the TVD section of each segment. The TVD sections own an attribute tvd_state_curr which represents the current occupancy state: *Vacant* or *Occupied*. Class methods are provided for setting the state of this attribute: event set_tvd_state_curr sets it to *Occupied* while event release_path_sect sets it to Vacant. These events are only enabled when the section belongs to a segment of an active path, *i.e., it is assumed that trains only move over active paths*. (This will be ensured by signals in the next refinement.)

At the **Vacancy Detection** level (Figure 2 on page 8), we break the atomicity of set_path_rel, which releases the current active path. Here we introduce the *par* combinator which allows the interleaving of its instance values zero or more times before its follow-on event executes. In this case, the *par* shows the possibility of occupying a TVD section (set_tvd_state_curr) then leaving it (release_path_sect) before releasing an active path. This ensures that all TVD sections are vacant before releasing the path. The ERS diagram visualises the system level process requirement leading to releasing a path which is not so explicit in the iUML-B.

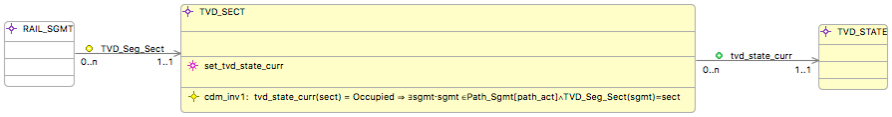


Fig. 6: Third refinement introducing detection of trains

Signals In the final refinement (Figure 7) we introduce signals that control the entry of trains wanting to use a path. The given set (class) SIGNAL has a variable attribute signal_aspect_curr which represents the current aspect and a constant attribute Signal_Aspect_Avail that provides the set of available aspects

for that signal. Note that the only specific signal aspect defined at this level is Signal_Aspect_Default which represents the signal's stop aspect. Other aspects may by introduced at later stages when tailoring the specification to a particular product. Class method set_signal_aspect_proceed sets the aspect of the signal to proceed (i.e, not default) while method set_tvd_state_signal sets the signal back to default as the corresponding connected section becomes occupied. Signals are related to paths via connectors. This is modelled by class (Elem_Ctor) and the associations Signal_Ctor and Path_Ctor_Beg. Signals are also related to TVD sections via their connectors and the association Sgmt_Ctor. A class invariant cdm_inv2 ensures that a signal is only set to a non-default value when there is an active path at the rear of the signal.

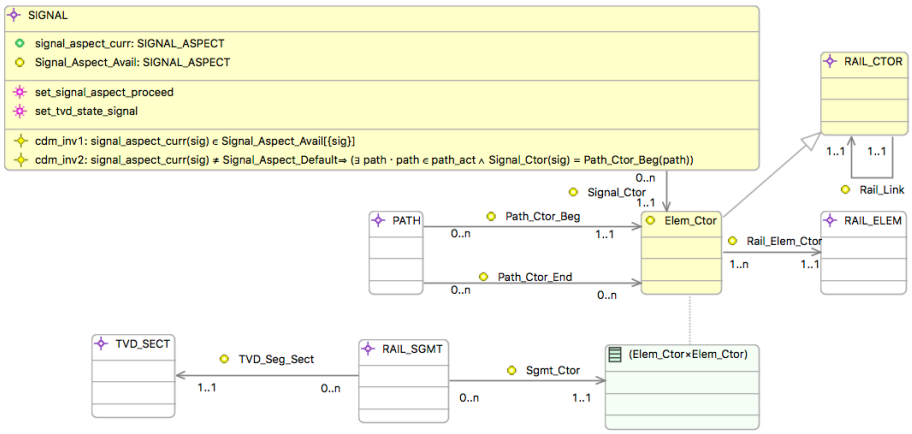


Fig. 7: Fourth refinement introducing signalling

In the **Signals** region of Figure 2, we split set_tvd_state_curr into two cases using the *xor* combinator. In the first case the TVD section is part of a path but is not protected by a signal (set_tvd_state_path). In the second case the section is protected by a signal. In this latter case we need to set the signal's aspect to proceed first. Then it is possible to occupy that section (set_tvd_state_signal), in which case we also set the signal's aspect back to default to indicate that the section is now occupied. Again the ERS diagram compliments the iUML-B model by visualising the system level process details of signal setting and how it interacts with TVD occupation.

4 Related Work

Our approach combines a state-based modelling notation (iUML-B) with a process-based notation (ERS). Essentially, iUML-B diagrams captures the complex data aspect of the system and their evolution, while ERS diagrams represent

the behavioural aspect of the system, in particular sequencing of events. In this sense, this is similar to various existing approaches combining state-based and process-based notations, e.g., CSP and Z [21], CSP and B [16,3], CSP and Event-B [17], etc.. In particular, these approaches also support development of systems via (separate) refinement of the state-based model and the process-based model. In our work, both iUML-B and ERS get there semantics by transforming them to Event-B and can contribute to the underlying Event-B model, hence their meanings are given entirely using Event-B. This is in constrast with the above mentioned approaches where essentially combining the different formalisms. As a result, the semantics of these approaches are given using more expressive notation such as *Unifying Theories of Programming* (UTP), for example [13].

We illustrate our approach on a case study based on the RailGround model [14]. The RailGround model is atypical in that it begins by modelling the established principals of interlocking systems without modelling the safety properties that those systems are designed to achieve. The reason for this is that the principles of interlocking are a proven design mechanism for controlling trains in a safe way. The model focusses instead on providing a precise and accurate specification of the interlocking product-line. The same case study has been used in [9] for illustrating the use of iUML-B class-diagrams to visualise domain-specific *Abstract Data Types* (ADTs). The RailGround case study is similar to the one tackled by Abrial [1, Chapter 17], however, our focus here is on the complementary usage of iUML-B and ERS diagrams for modelling. In [7], the authors present the development of a train control system using Event-B with the focus is on the use of ADTs to simplify the modelling task. In particular, the system is based on *Commnications-based Train Control* (CBTC) and hence the focus is on train tracking using moving blocks. In [11], the authors use `CSP||B` [16] to model an interlocking system and verifying the system using ProB model checker [12]. In [20], the authors present an approach for formal development of interlocking systems using a DSL to specify the configuration data of the interlocking system. The data is then used to generate a concrete behaviour model of the interlocking system from a generic behavioural model and concrete system properties from generic properties. The concrete properties and the conrete system are then verified using SMT-based bounded model checking (BMC) and inductive reasoning. In both [11] and [20], refinement is not considered.

5 Conclusion

In this paper we present an alternative development to the RailGround interlocking system, which was originally developed by Thales Austria GmbH using plain Event-B. Our approach is based on a combination of a state-based (iUML-B) and a process-based (ERS) approach. The RailGround model contains several complex entity relationships to represent the railway topology. By separating the entity relationship model, illustrated by the class diagrams, from the behavioural model, we are making the modelling process more efficient. The class diagram allows us to explore different abstractions efficiently compared with

specifying these data relationships textually in Event-B. The behavioural model of the system is represented by both the state-oriented state-machines and the process-oriented ERS diagrams. The statemachines give a view of the behaviour which is local to a particular type of entity. ERS makes the system level ordering of the transitions more visible. ERS also explicitly represents the event decomposition and their refinement relationships. In our developments, we used the iUML-B graphical tool which also automatically generates part of the Event-B model, making the modelling process more intuitive and efficient for engineers. Here, we only used the ERS as a visualisation to avoid the duplication of control variables generated by both ERS and state-machines. However, the ERS and state-machine views of behaviour complement each other and facilitate the modelling process.

The presented approach, which is based on different visualisations from different perspectives makes the model more understandable and easier to communicate, but also simplifies the model and is thus cheaper to verify and validate, which is a primary goal of the Enable-S3 project. This is a huge benefit for any further model adaptation and modifications, which are inevitable due to long product life time (25+ years). In future work we will do some trials to assess the complexity of making changes to the model with and without visualisations.

Future work In this paper, our focus is on building an approach for the generic modelling of the core system. Currently iUMLB and ERS specify control flow from different perspectives. We are looking at a tool integration of ERS with iUML-B to have a common generation mechanism of Event-B.

We have started to complement the approach with DSLs for specifying customer-specific variations in feature requirements. For example, a DSL for specifying signalling has been developed. The DSL is precise enough to perform a certain amount of static-checking, but easily understood by the customer's domain experts. The signalling specification is translated into an Event-B machine which is proven to refine the generic signalling required by the core product model. We are developing composition and instantiation mechanisms so that we can isolate the generic signalling requirements as a separate component in the core product model. This component structuring of the model helps to address scalability and re-use as well as facilitating customer specific feature variants in order to efficiently obtain a complete and verified model of a customer specific product.

Acknowledgement

This work has been conducted within the ENABLE-S3 project that has received funding from the ECSEL Joint Undertaking under Grant Agreement no. 692455. This Joint Undertaking receives support from the European Union's HORIZON 2020 research and innovation programme and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, Norway.

References

1. Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: An open toolset for modelling and reasoning in Event-B. *Software Tools for Technology Transfer*, 12(6):447–466, November 2010.
3. Michael J. Butler and Michael Leuschel. Combining CSP and B for specification and property verification. In John S. Fitzgerald, Ian J. Hayes, and Andrzej Tarlecki, editors, *FM 2005: Formal Methods, International Symposium of Formal Methods Europe, Newcastle, UK, July 18-22, 2005, Proceedings*, volume 3582 of *Lecture Notes in Computer Science*, pages 221–236. Springer, 2005.
4. The Enable-S3 Consortium. Enable-S3 European project, 2016. www.enable-s3.eu.
5. D. Dghaym, M.G. Trindade, M.J. Butler, and A.S. Fathabadi. A Graphical Tool for Event Refinement Structures in Event-B. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 269–274. Springer, 2016.
6. Asieh Salehi Fathabadi, Michael Butler, and Abdolbaghi Rezazadeh. Language and tool support for event refinement structures in Event-B. *Formal Aspects of Computing*, 27(3):499–523, May 2015.
7. Andreas Fürst, Thai Son Hoang, David A. Basin, Naoto Sato, and Kunihiko Miyazaki. Large-scale system development using abstract data types and refinement. *Sci. Comput. Program.*, 131:59–75, 2016.
8. Thai Son Hoang. An introduction to the Event-B modelling method. In *Industrial Deployment of System Engineering Methods*, pages 211–236. Springer-Verlag, 2013.
9. Thai Son Hoang, Colin Snook, Dana Dghaym, and Michael Butler. Class-diagrams for abstract data types. In Van Hung Dang and Deepak Kapur, editors, *ICTAC 2017*. Springer, 2017. To appear.
10. M. A. Jackson. *System Development*. Englewood Cliffs, N.J. : Prentice-Hall, 1983.
11. Phillip James, Faron Moller, Hoang Nga Nguyen, Markus Roggenbach, Steve A. Schneider, and Helen Treharne. On modelling and verifying railway interlockings: Tracking train lengths. *Sci. Comput. Program.*, 96:315–336, 2014.
12. Michael Leuschel and Michael Butler. ProB: An automated analysis toolset for the B method. *Software Tools for Technology Transfer (STTT)*, 10(2):185–203, 2008.
13. Marcel Oliveira, Ana Cavalcanti, and Jim Woodcock. A UTP semantics for *Circus*. *Formal Asp. Comput.*, 21(1-2):3–32, 2009.
14. Klaus Reichl. RailGround model on github, 2016. https://github.com/klar42/railground/ (Accessed 20/04/2017).
15. Mar Yah Said, Michael Butler, and Colin Snook. A method of refinement in UML-B. *Softw. Syst. Model.*, 14(4):1557–1580, October 2015.
16. Steve Schneider and Helen Treharne. CSP theorems for communicating B machines. *Formal Asp. Comput.*, 17(4):390–422, 2005.
17. Steve Schneider, Helen Treharne, and Heike Wehrheim. A CSP approach to control in event-b. In Dominique Méry and Stephan Merz, editors, *Integrated Formal Methods - 8th International Conference, IFM 2010, Nancy, France, October 11-14, 2010. Proceedings*, volume 6396 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2010.
18. Colin Snook. iUML-B statemachines. In *Proceedings of the Rodin Workshop 2014*, pages 29–30, Toulouse, France, 2014. http://eprints.soton.ac.uk/365301/.

19. Colin Snook and Michael Butler. UML-B: Formal modeling and design aided by UML. *ACM Trans. Softw. Eng. Methodol.*, 15(1):92–122, January 2006.
20. Linh Hong Vu, Anne Elisabeth Haxthausen, and Jan Peleska. Formal modelling and verification of interlocking systems featuring sequential release. *Sci. Comput. Program.*, 133:91–115, 2017.
21. Jim Woodcock and Ana Cavalcanti. The semantics of Circus. In Didier Bert, Jonathan P. Bowen, Martin C. Henson, and Ken Robinson, editors, *ZB 2002: Formal Specification and Development in Z and B, 2nd International Conference of B and Z Users, Grenoble, France, January 23-25, 2002, Proceedings*, volume 2272 of *Lecture Notes in Computer Science*, pages 184–203. Springer, 2002.