Science of Computer Programming 233 (2024) 103057

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

OnTrack: Reflecting on domain specific formal methods for railway designs

Phillip James [*], Faron Moller, Filippos Pantekis

Department of Computer Science, Swansea University, Swansea, UK

ARTICLE INFO

Keywords:
Model driven engineering
Formal methods
Railway verification

ABSTRACT

OnTrack is a tool that supports workflows for railway verification that has been implemented using model driven engineering frameworks. Starting with graphical scheme plans and finishing with automatically generated formal models set-up for verification, OnTrack allows railway engineers to interact with verification procedures through encapsulating formal methods. OnTrack is grounded on a domain specification language (DSL) capturing scheme plans and supports generation of various formal models using model transformations. In this paper, we detail the role model driven engineering takes within OnTrack and reflect on the use of model driven engineering concepts for developing domain specific formal methods toolsets.

1. Introduction

In this paper, we reflect on the development of OnTrack, a tool that automates workflows for railway verification. Such workflows usually start with graphical scheme plans or scheme plan representations and finish with automatically generated formal models set up for verification. OnTrack has been implemented using a number of model driven frameworks and aims to overcome typical issues surrounding the uptake of formal methods by industry [38,18,17,21] by encapsulating such methods within a domain specific tooling environment. The paper reflects upon advances made in a number of previous papers [39,35,38,47].

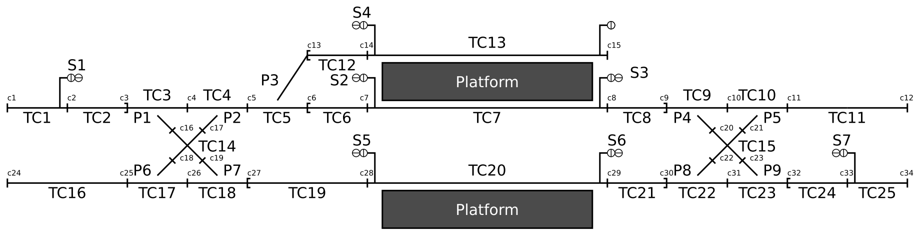
For many years, the application of verification processes such as model checking and interactive theorem proving to various industrial case studies has been successfully illustrated, e.g. see [52,9,59,62,48,31,34,58,19,50,15,27]. Even though these approaches have been successful from a Computer Science perspective, the adoption of formal methods within industry is still limited [10] due to questions around faithful modelling, scalability and accessibility [38,4,20]. Without experts in the field of formal verification, the modelling approaches presented are often in a form that is acceptable to computer scientists, but not to the engineer working within the domain. These presentations thus lead to doubts in the approach by the engineers and a low level of confidence towards the capabilities of the approach to correctly capture the systems being modelled. At the same time, many verification methods are prone to suffering from a scalability problem that makes their application to large industrial problems lengthy and often unfeasible. Finally, tool support for verification procedures is often aimed towards a Computer Science audience interested in verification, and hence is not easily accessible to engineers outside the field of formal methods. This work gives an experience report on how these problems can be overcome by using model driven engineering within the domain of railway signalling.

Within the railway industry, it is common practice to define graphical descriptions of railway networks. Such descriptions enable an engineer to visually represent the tracks and signals etc., of a railway network. Within OnTrack, we offer this graphical language

[*] Corresponding author.
E-mail addresses: P.D.James@swansea.ac.uk (P. James), F.G.Moller@swansea.ac.uk (F. Moller), Filippos.Pantekis@swansea.ac.uk (F. Pantekis).

https://doi.org/10.1016/j.scico.2023.103057
Received 19 March 2023; Received in revised form 7 November 2023; Accepted 8 November 2023

Available online 15 November 2023



Control Table

Route	Normal	Reverse	Clear
R1A	P1, P2	P3	TC2, TC3, TC4, TC5, TC12, TC13
R1B	P1, P2, P3		TC2, TC3, TC4, TC5, TC6, TC7, TC8
R1C		P1, P7	TC2, TC3, TC14, TC18, TC19, TC20, TC21
R2	P3	P2, P6	TC6, TC5, TC4, TC14, TC17
R3	P4, P5		TC8, TC9, TC10
R4		P3, P2, P6	TC12, TC5, TC4, TC14, TC17
R5	P7, P6		TC19, TC18, TC17
R6		P8, P5	TC21, TC22, TC15, TC10
R7A		P4, P9	TC24, TC23, TC15, TC9, TC8, TC7, TC6
R7B	P9, P8		TC24, TC23, TC22, TC21, TC20, TC19

Fig. 1. A simple scheme plan.

as one particular starting point. OnTrack is grounded on a domain specification language (DSL) for the railway. Around this DSL, we have created a graphical editor and implemented a number of model transformations. These transformations can be used to generate formal specifications in different languages. We distinguish between model transformations for abstraction (DSL meta-model to DSL meta-model), representation (DSL meta-model to a meta-model for a formal specification language) and generate for verification transformations (DSL/formal specification language meta-model to plain text). OnTrack also offers the ability to visualise failed verification attempts within the OnTrack editor, rather than at the mathematical level of the verification tool output.

In addition, both the DSL and the extensibility of OnTrack to allow the generation of formal models in any given language form part of the novelty of our work. The second key aspect of our approach is that we define abstractions on the DSL in order to yield optimised descriptions prior to formal analysis. Importantly, these abstractions allow for common benefits for verification across different formal languages. The effectiveness of the approach has been illustrated through its application to verify a number of different real world railway scheme plans. We note that OnTrack was developed through 2012 – 2018 and thus some of the languages and techniques we discuss in this paper have since advanced. However we feel the essence of the lessons from the development remain insightful.

Finally, OnTrack is designed for the railway domain, but the clear separation of an editor with support for abstractions from the chosen formal language is a principle that we feel is more widely applicable.

The remainder of this paper is structured as follows. In Section 2 we briefly introduce the field of railway signalling verification and discuss related contributions. In Section 3 we explore the core model driven frameworks that have been used in implementing OnTrack. Section 4 then explores the main structure of the OnTrack tool, highlighting the key areas where model driven frameworks have successfully played a role. In Section 5 we highlight challenges that were faced when implementing particular features of the tool within an MDE setting. Finally, Section 6 gives a reflection from the authors on the use of model driven frameworks for developing formal methods toolsets, before we conclude the paper in Section 7.

2. Verification of railway interlocking data

Railway control systems are a typical example of a safety-critical system, where their failure could lead to catastrophic consequences. A core system for ensuring safety is the Interlocking system. Interlockings operate functionally as a filter between inputs from railway signallers and the concrete railway infrastructure itself. Their aim is to ensure that requested changes from the operator, such as setting of a particular route, are only undertaken if it is safe to do so with respect to the current state of the railway.

Interlocking applications are developed according to processes prescribed by railway authorities, such as Network Rail's *Governance for Railway Investment Projects* (GRIP) process. The first four GRIP phases (Output definition, Feasibility, Option selection, Single option development) define the track plan and routes of the railway to be constructed. The next phase, is then contracted to signalling companies such as Siemens Rail Automation (who have supported the research we present here). Such companies choose appropriate track equipment and add control logic in the form of control tables that stipulate, among other things, when it is safe to set routes. The track plan in addition to the control tables is typically referred to as a scheme plan (see Fig. 1). The signalling company, based upon these design documents, then implements the interlocking system. Thus, correctness of this detailed design and in particular the control logic is fundamental to ensuring safety of the railway.

2.1. Track plans

A railway network consists of a number of track-side elements of different types, for instance linear sections, points, and physical signals (or marker boards in newer deployments). The track plan in Fig. 1 shows an example layout of a railway network having a number of track circuits (TC1, TC2, TC3 etc.), a number of points (P1, P2 etc.), and 7 signals (S1, S2, etc.).

Track circuits are detection sections, and are used by interlocking systems to detect the presence of trains in a railway network. A point can be switched between two positions known as Normal and Reverse. A signal is used to control entry to a particular route, which is a region of the scheme plan that depends on the direction of travel. For example Signal S4 controls whether or not a train can leave the top platform and exit the track plan via TC16. The rules on when a signal can be set to allow a train to proceed are captured within so-called control and release tables. Here we present only the former for simplicity.

2.2. Control tables

An interlocking system constantly monitors the status of track-side elements, and also sets them to appropriate states in order to allow trains to travel safely through the given railway network. A control table specifies the routes in the given network layout and the conditions for setting these routes. A route is a path from a source signal to a destination signal.

In railway signalling terminology, setting a route denotes the process of allocating resources. That is, sections, points and signals for the route are set and then locked for use exclusively for one train. The specification of a route and conditions for setting it includes the following information. Considering the control table shown in Fig. 1: the name of the route (e.g. R1A), a list of the detection sections in the route's path that need to be clear of trains (e.g. TC2, TC3, TC4, TC5, TC12 and TC13) and the required positions of points used by the route (e.g. P1 and P2 are Normal and P3 is Reverse). Release tables are similar, but instead dictate when resources along a route can be released after they have been used by a train.

2.3. Safety guarantees

In order to prevent collisions and derailment of trains, interlocking systems employ a simple logical principle: a route is locked exclusively for use by one train at a time. Thus, it is common that signalling companies wish to check various properties of their designs, including:

Collision-freedom: which excludes two trains occupying the same track.
Run-through-freedom: which states that whenever a train enters a point, the point is set to cater for this. For example, considering Fig. 1 e.g., when a train travels from track TC12 to track TC5, point P3 is set so that it connects TC12 and TC5 (and not TC5 and TC6).
No-derailment which prescribes that whenever a train occupies a point, the point does not move.

Here, correct design for the scheme plan is clearly safety-critical as mistakes can lead to a violation of any of the three safety properties above.

2.4. Related work on verification environments

It is still an open research question as how to perform safety checks on interlocking designs. Here the main research challenge is how to cope with the complexity of the problem as the state space to be verified grows exponentially in the size of the scheme plan. Over a sustained period, several research groups, see e.g. [32,3,29,24,26,25,57,38,37,16,30,61,60,56,11,44,58,19,50,15,27], have been addressing this challenge and have developed a number of different modelling and verification approaches. Indeed, recent approaches have also been shown to scale well to modern industrial systems [27,15,50] and even to railway systems yet to be realised within rail industry [14]. Also, effort has been undertaken to review the usability of railway verification tools, where, for example, Ferrari et al. have systematically assessed the usability of 13 different formal methods tools [17]. In spite of this, formal methods still lack widespread use within industry often due to questions surrounding the usability and expertise required for applying formal methods [18,17,21].

The modelling part of such approaches usually consists of "transformations" of how to derive a (formal) model from informal rail descriptions as used in rail industry such as a track plan (e.g., as a CAD drawing) enriched by various tables (e.g., a control table). Similarly, the verification part usually states a safety condition (e.g., no train collision) and expresses this as a (formal) property (e.g., as a logical formula). Finally, a suitable and often automated verification tool is utilised to provide an answer if the property holds for the given model.

With respect to modelling, historically, Dines Bjørner has notably developed an extensive DSL for the railway domain [6,8,7]. Bjørner's DSL – the DSL on which we build later – has been applied in the PRaCoSy (People's Republic of China Railway Computing System) project to model a 600 km line between Zhengzhou and Wuhan [8]. More recently, Vu et al. have developed a detailed DSL aimed particularly at interlocking modelling and verification [58].

There are several projects with a close relevance to this work. The first is the development environment for verification of railway control systems created by Haxthausen and Peleska [23,28]. This environment includes a DSL allowing modelling of control systems, and an automatic translation from models described in this DSL to executable control programs. At each level of production, various safety checking steps are taken. The difference between our approach and this is that we employ standard techniques from the field of MDE to support our toolset, rather than just implementing elements such as model transformations in a general purpose language. We highlight the benefits of this in Section 6. Similarly, work by Luteberget et al. [43] has explored automated graphical layout of scheme plans, which indeed was a challenge encountered within our work. Here their approach encodes the optimisation challenge as a SAT problem, where as we utilised simulated annealing [47].

Next, is the SafeCap project [51] which has the aim of improving railway capacity safely by integrating proof-based reasoning about time and state-based models. Part of the project aims to develop an intuitive graphical DSL for the railway domain with a tailored toolset [33,53] supporting verification of railway plans. Their approach is based on Event B [1] and the Rodin framework [2]. The approach taken in the development of this graphical language is inspired by the methodology we present in this work (in fact, the SafeCap DSL and toolset have been developed in co-operation with several of the authors of this paper).

Finally, work by Kanso [40] presents a framework that aids in the development of verified railway interlockings. The framework is built around the Agda theorem prover and has been applied to verify two existing railway control systems. The approach by Kanso also presents novel results on integrating model checking into Agda. Here we differ from Kanso, as we concentrate on designing verification processes with industrial applicability in mind, rather than applying new formal methods to industry in an exemplary fashion.

3. Model driven frameworks

In this section, we discuss the main Eclipse IDE components and plugins that we use for creating the OnTrack DSL and the associated tool support. To this end, we discuss the Eclipse Modelling Framework (EMF) [54], the Graphical Modelling Framework (GMF) [22] and Epsilon [42]. Each of these plugins are developed to support various aspects of model driven engineering and development [41,5] of DSLs.

3.1. Eclipse modelling framework

Many people consider the core of a language to be its abstract syntax. From an abstract syntax, one can develop artefacts such as a concrete syntax or model transformations to another abstract syntax. The Eclipse Modelling Framework [54] is a modelling framework and code generation facility for building tools and other applications based on a structured data model. Part of this framework includes Ecore [54] which is a UML class diagram like language for describing meta-models for DSLs. Models are stored using the XMI (XML based) file format and can be edited using a number of varying viewpoints. From such a XMI model specification, EMF provides tools and runtime support for producing various Java classes for the model, along with a set of adaptor classes that enable viewing and editing the model. Finally, such a model serves as the basis for creating a graphical syntax for a DSL using GMF. We have utilised EMF to define the elements of the underlying domain specific language within OnTrack. For further reading on EMF we refer the reader to [54].

3.2. Graphical modelling framework

The GMF or Graphical Modelling Framework project [22] provides the features allowing one to develop, from an Ecore meta-model, a graphical concrete syntax for a DSL. The result of applying the GMF process is a graphical editor encapsulating this graphical concrete syntax. Such an editor is shown in Fig. 2. This editor consists of a drawing canvas (in the centre) and a palette (right hand side). Graphical elements from the palette can be dragged and positioned onto the drawing canvas. Overall, the editor can be used to produce model instances of the DSL described by the underlying Ecore meta-model.

GMF uses the Graphical Editing Framework GEF for many of its features, but provides a useful development framework on top of GEF. The main features of GMF can be split into two components: a tooling framework for developing graphical editors and a runtime framework for running such editors. Here, we discuss the tooling component.

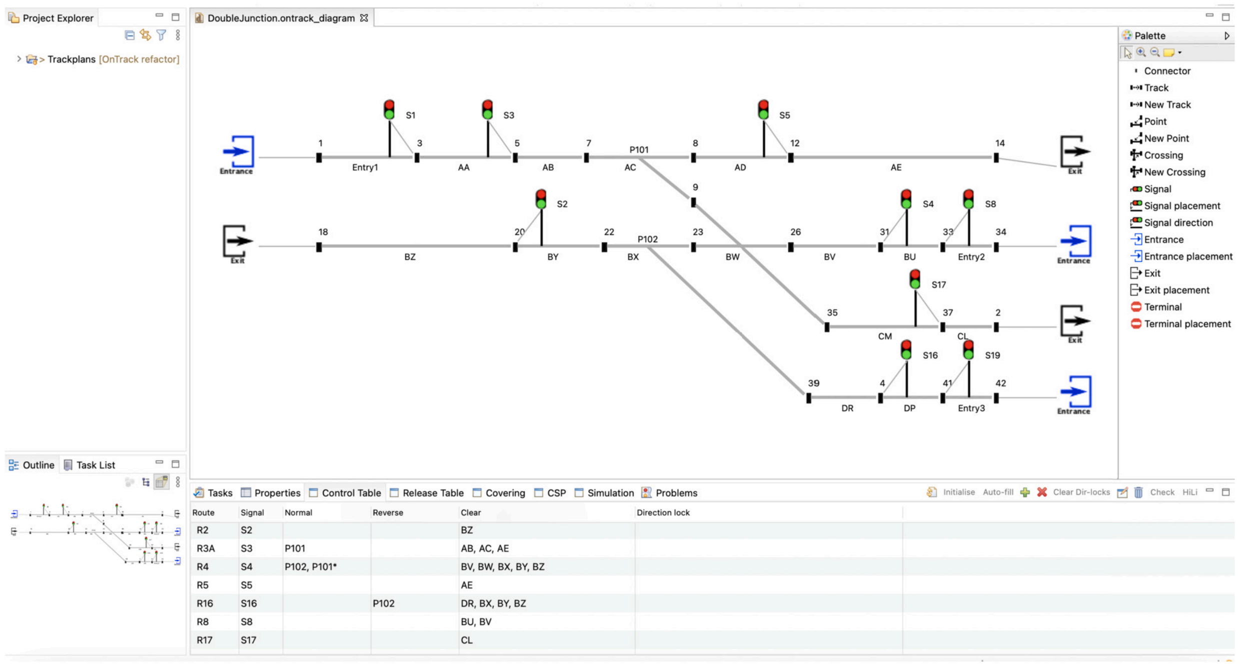
3.2.1. GMF tooling

The tooling component of GMF provides easy access and model driven editing to several models that are required to create a GMF editor plugin.

Graphical Definition Model: The graphical definition model is where the user can define the various figures to be used for the concrete syntax.

Tooling Definition Model: As illustrated in Fig. 2 (right hand side), most editors created using GMF include a palette allowing users to create and work with constructs from the concrete syntax of the DSL. The tooling definition model is where users can define and design the elements to be included and displayed in the palette.

Mapping Model: The mapping model is one of the most important models used when generating a GMF editor. It is where one can define how elements from the graphical definition model and tooling definition model are linked to elements from the underlying Ecore meta-model.



Route	Signal	Normal	Reverse	Clear	Direction lock
R2	S2			BZ	
R3A	S3	P101		AB, AC, AE	
R4	S4	P102, P101*		BV, BW, BX, BY, BZ	
R5	S5			AE	
R16	S16		P102	DR, BX, BY, BZ	
R8	S8			BU, BV	
R17	S17			CL	

Fig. 2. OnTrack GMF editor for railway scheme plans.

Generator Model: Finally, the generator model combines the information of the previous models with details that are needed to generate code for the editor. The generation of this model from the mapping model is often an automatic step, however it is possible to customise this model to include features such as extension points [22]. The generator model is used to generate a DSL editor similar to the one shown in Fig. 2.

For further details on developing GMF editors we refer the reader to [54,22].

3.2.2. *Epsilon*

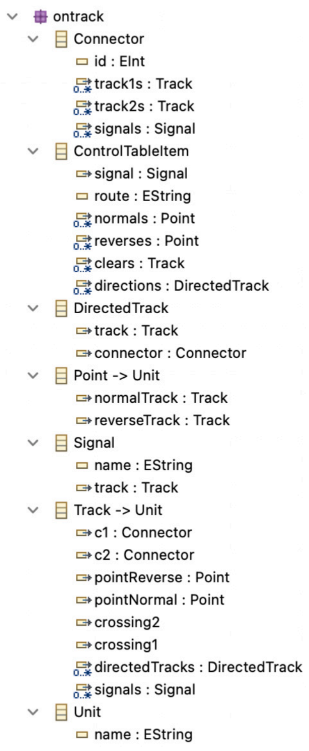
Often, the development of a GMF editor is motivated by the possibility of producing, from an instance model created by the editor, some sort of output usually in the form of text or program code. Similarly, many people wish to transform the model into a slightly different model, or compare it to another model that may be an instance of a different meta-model. To help with these tasks, users can make use of what are known as model transformations. Although there are several possible frameworks for defining model transformations we concentrate our review on the Epsilon framework [42] used in this work.

Epsilon, an extensible platform of integrated languages for model management [42], provides a family of languages and features for defining and applying model transformations, comparisons, validation and code generation. In the case of Ecore meta-models, the main types of model transformation which are of interest to us are:

1. Model-to-text transformation (M2T): Model-to-text transformations can be viewed as model-to-model transformations, where the output model is simply an instance of the (very general) meta-model defining sequences of characters. Such transformations are often used for code generation from a given model to a programming language. Later in Section 4.1.1, we will use this type of transformation to generate formal specifications from graphical models. Interestingly when generating text, one can opt to use a meta-model for the output text or to skip the meta-model and simply directly output text.
2. Model-to-model transformation (M2M): Model-to-model transformations define how a model instance of one Ecore meta-model can be transformed into a model instance of (optionally) another Ecore meta-model. Later in Section 4.1.2 we will use these to capture abstractions from the field of model checking of Interlockings.

To support the above model transformations Epsilon provides several languages [42] of which we consider and use:

EOL: The Epsilon object language that provides a common set of model management constructs. EOL forms the base language of which the other Epsilon languages are constructed.
ETL: The Epsilon transformation language for specifying model to model transformations.
EGL: The Epsilon generation language for model-to-text transformations. EGL provides a templating feature for code generation without requiring a meta-model for the output model.



```
ontrack
  Connector
    id : EInt
    track1s : Track
    track2s : Track
    signals : Signal
  ControlTableItem
    signal : Signal
    route : EString
    normals : Point
    reverses : Point
    clears : Track
    directions : DirectedTrack
  DirectedTrack
    track : Track
    connector : Connector
  Point -> Unit
    normalTrack : Track
    reverseTrack : Track
  Signal
    name : EString
    track : Track
  Track -> Unit
    c1 : Connector
    c2 : Connector
    pointReverse : Point
    pointNormal : Point
    crossing2
    crossing1
    directedTracks : DirectedTrack
    signals : Signal
  Unit
    name : EString
```

Fig. 3. A snippet of the OnTrack DSL.

EWL: Finally, the Epsilon wizard language for defining and executing transformation workflows, including activating transformations from a GMF editor.

More details on these languages and their formal definitions can be found in the Epsilon book by Kolovos et al. [42].

4. OnTrack architecture and functionality

OnTrack has been created using the EMF [54] and GMF frameworks [22] and multiple Epsilon [42] model transformations. Fig. 4 shows the workflows that we support in OnTrack. Initially, users can either import an existing rail plan in RailML [46] or they can draw a *track plan* using the graphical front end. They can then enrich this graphical plan by adding textual (table based) information for control and release tables. Together these form a scheme plan, which are models formulated relative to OnTrack's DSL meta-model. Fig. 3 provides a snippet of this DSL highlighting the elements required to capture a control table. However the full DSL contains 20 classes and around 60 associations. We note that the DSL only captures those elements necessary for verification of control tables against safety properties at ERTMS Level 2 and below, which means it does not capture many elements of railway systems. The DSL also took some time in development and was a result of many discussions with Railway experts. It was also built upon the elements visible in manually constructed formal models that had been initially, in a time-consuming manner, developed to explore verification techniques.

A scheme plan is then the basis for subsequent workflows that support its verification, simulation, analysis etc. Scheme plans can then be translated to formal specifications in various (specification) formalisms.

Once a formal model has been generated, it can be simulated or verified using the tools associated with the formal specification language that has been used as the generation target language. For example, ProB can be used for animating and verifying CSP||B, SPASS can be used for verifying CASL, TimedCSP Simulator can be used for simulating and visualising train runs.

OnTrack is extensible, that is, the editor, tooling and importantly, abstractions (see Section 4.1.2) can be reused to generate formal models from various contexts by providing a suitable model transformation to that context. These models can then be simulated and analysed with the respective tools. In principle, this workflow can be fully automatic. In a prototyping phase, one would support it only partially, i.e., OnTrack produces a file that then needs to be loaded into another tool, rather than having OnTrack opening this other tool directly.

4.1. The successful role of model transformations

The original goals of OnTrack were to support the generation of formal models from a graphical model. Here, it was somewhat planned that model transformations would be used to generate text based formal models. However, throughout development, it

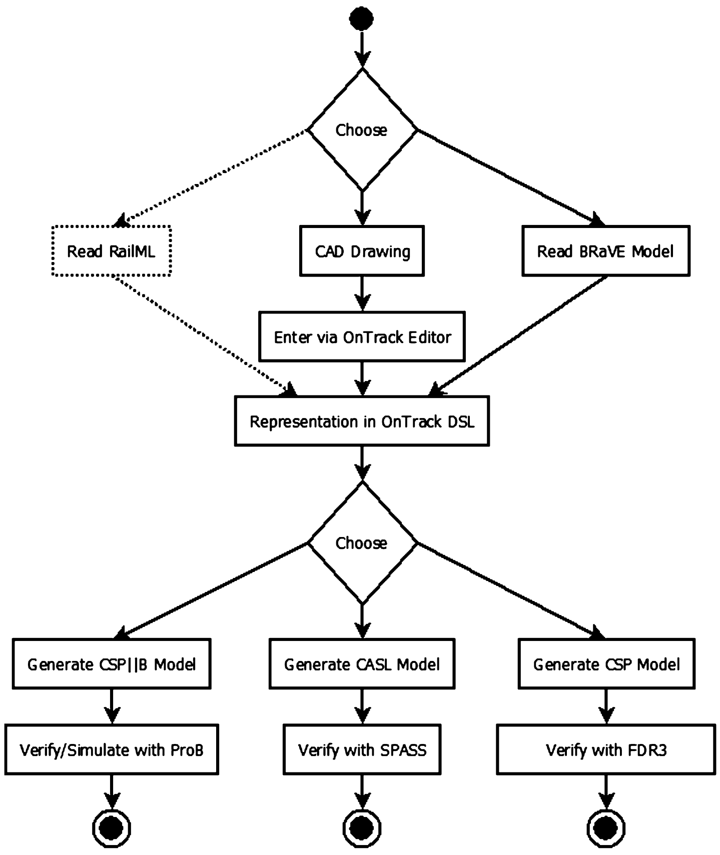


Fig. 4. The OnTrack workflow.

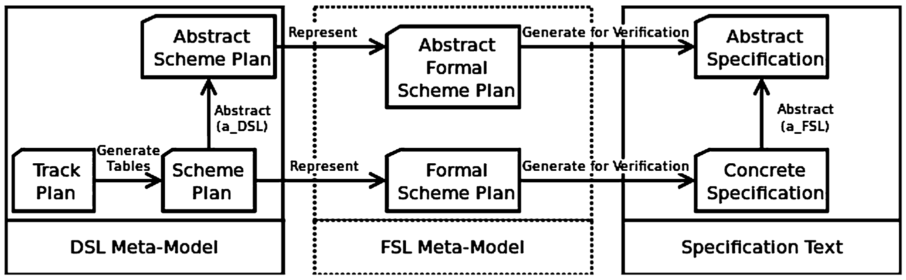


Fig. 5. Model transformations within OnTrack.

became apparent that model transformation could also be used as a powerful way of implementing common abstractions (see Section 4.1.2) that are needed to allow for successful verification. For example, see the abstractions explored in [37,45]. We now discuss the role of model transformations within these settings. The general scheme is captured in Fig. 5.

Initially, a user draws a *Track Plan* using the graphical front end. Then the first model transformation we have implemented, *Generate Tables* leads to a *Scheme Plan*, which is formed from a track plan and its associated control tables. This model transformation is included to aid signalling engineers by pre-populating control tables with data that can be automatically derived from the scheme plan. We note, that in general it is not possible to fully automatically generate control tables as there are often design choices to be made.

Next, there are two possible scenarios for the horizontal workflow that depend upon whether or not a meta-model is available for the formal specification language one would like to generate.

1. Using a meta-model for the formal specification language: The first option is to have a meta-model describing the formal specification language. A *Represent* transformation translates a *Scheme Plan* into an equivalent *Formal Scheme Plan* over the meta-model of the formal specification language. Then various *Generate for Verification* model-to-text transformations turn a *Formal Scheme Plan* into a *Formal Specification Text* ready for verification.

```
spec Pair [sort S] [sort T] =
    sort Pair[S,T]
    ops first: Pair[S,T] -> S;
        second: Pair[S,T] -> T;
...
spec StaticSignature =
     sorts Signal, Unit, Connector ...
     sorts Track, Point < Unit ...
...
```

Fig. 6. Example of static text generation for our DSL.

```
1.  [% var rail : OTDiagram  := OTDiagram.allInstances().at(0); %]
2.  ...
3.  [%if(rail.hasUnits.size > 0){%]
4.      free type Unit ::=
5.      [%  var i := 0;
6.          while (i < rail.hasUnits.size()-1){ %]
7.          [%  var unit : Unit := rail.hasUnits.at(i);
8.             i := i+1; %]
9.             [%=unit.name%] | [%}%]
10.     [% var unit : Unit := rail.hasUnits.at(i); %]
11.     [%=unit.id%] [%}%]
```

Fig. 7. Dynamic text generation for elements of Unit free type.

2. Direct generation of a formal specification: The second approach is to directly generate a formal specification. Thus only the *Generate for Verification* model-to-text transformations need to be implemented.

In both cases, the *Generate for Verification* transformations can enrich the models appropriately for verification, e.g. by including the various lemmas that may be known to help that formal method in proving safety.

This horizontal workflow provides a transformation yielding a formal specification that faithfully represents a scheme plan. In Section 4.1.1 we highlight the second approach that has been taken for the generation of CASL specifications 4.1.1.

Finally, the top level of the workflow shows the ability of OnTrack to include abstractions. We are interested in abstractions as these can ease verification. For currently implemented abstractions, the diagram commutes. Although we note that abstractions within the tool are implemented at the DSL meta-model level, whilst any abstractions on concrete specifications are part of the formal construction within the given formal language. Thus, when implementing new abstractions, one should check that the implementation meets the formal definition of the abstraction.

4.1.1. Formal model generation

Here we describe the direct implementation of the *Generate for Verification* transformations for CASL. The *Generate for Verification* transformation translates meta-model instances of OnTrack's DSL into formal specification text. This transformation is implemented using the Epsilon Generation Language (EGL) [42]. EGL allows template files to be written describing the text to be generated. These templates provide two main features for outputting text, namely the ability to output *static text* and to output *dynamic text*. *Static text* is considered text that is always generated independent of the model. Whilst *Dynamic text* is text that depends on the given model. For this reason, *Dynamic text* is sometimes referred to as configuration data. By default, any text written in an EGL template is considered to be static text. For example we know that the specification of data types for our DSL and similarly our extension of this with dynamical aspects is the same for all models. Hence this is rather straightforwardly encoded as static text to be output, see Fig. 6.

Within the same EGL template, we can then specify the output for a concrete scheme plan. For example, consider the free type of units that is used to capture track-like components. Here, such a free type is built from the concrete elements of linear tracks and points contained within the graphical model. Hence we can specify the template in Fig. 7 for the dynamic generation of the free type Unit. The result of applying this EGL fragment, to the concrete scheme plan in Fig. 1 is the following CASL specification fragment:

```
free type Unit::= TC1 | TC2 |... | P1 | P2... .
```

Considering Fig. 7, the first construct of EGL that we notice is `[%` and `%]`. Any text specified between such a set of brackets is interpreted as code. For example, the line `var rail: OTDiagram:=...` is a line of EGL code for declaring the variable `rail` and assigning to it the current scheme plan instance within the graphical editor. This variable, can then be used throughout the EGL template to refer to the current model instance. Next, we see an EGL if statement (line 3). This statement checks the number of elements in the `hasUnits` relation of the current rail diagram. If there are linear units or points that have been drawn in the

```
1.  rule abs transform rd: Input!OTDiagram to rd2 : Target!OTDiagram {
2.        rd.computeAbstractions();
3.        for(ut:Unit in rd.hasUnits){
4.            if(not (toDelete.contains(ut))){
5.                if(consToBeMapped.contains(ut.hasC1))  {
6.                    ut.hasC1 = ut.hasC1.getMapping();
7.                 }
8.                if(consToBeMapped.contains(ut.hasC2))  {
9.                    ut.hasC2 = ut.hasC2.getMapping();
10.                }
11.               rd2.hasUnits.add(ut);
12.           }
13         }
14.        //Omitted code: computation with connectors and signals//
15.        rd2.computeTables();
16.  }
```

Fig. 8. ETL rule for abstract model transformation.

diagram, the code inside the if statement is executed. The first line within the if statement is static text to be generated. That is, as long as the if statement is entered, the text "`free type Unit::=`" will be output. Lines 6 to 9 perform a loop through the units of the concrete scheme plan instance. For each unit up until the last but one, we can see that in line 9 the dynamic text generation "`[%=unit.name%]`" is executed. Here, the dynamic text generation also contains the "`=`" symbol. This indicates that the text following is a piece of code that returns a value. For example, "`unit.name`" is a field containing the name that has been given to the current unit element. This name will then be output by the generation process. This dynamic text generation is immediately followed by the static text generation "`|`". This produces the "`|`" symbol between elements of the free type. Finally, after the while statement there is another block of code (lines 10 and 11) that outputs the last unit identifier in the collection.

In a similar manner to the presented free type generation, it is possible to explore all the elements of the diagram generating the concrete scheme plan specification in Casl. After generation of the scheme plan, the safety property to be proven over the scheme plan can also be generated. As Casl supports quantification within property specifications, this property is the same for all scheme plans and is thus simply generated as static text. The result is a full Casl specification ready for verification of the current model instance.

Overall, this generation means that OnTrack achieves the aim of automating the production of formal specifications from a graphical model. OnTrack is a toolset that is usable by engineers from the railway domain and allows them to produce formal Casl specifications ready for verification.

4.1.2. DSL level abstractions

Research groups have developed so-called domain level abstractions for the setting of railway verification [37,45,36,49,27,15,50]. These abstractions take advantage of domain specific features of railways in order to provide benefits to verification. Such abstractions often aim to reduce the scale and complexity of the railway plan that is needed to prove safety. They are typically expressed over a mathematically formulated DSL of the domain and often require a manual proof to be constructed within the setting of the formal specification approach being utilised. Within OnTrack, we have implemented a number of such abstractions ranging from simple removal of non essential elements (discussed below), to the much more complex construction of influencing regions (see the covering abstraction provided by James et al. [36]).

Within OnTrack one of the implemented abstractions is based on the *simplifying scheme plan* abstraction by Moller et al. [45], where various sequences of units are "collapsed" into single units. For example, considering Fig. 1, route R1A ends with track circuits $TC12$ and $TC13$ which appear in the clear column of the control table. As these are straight tracks, the abstraction allows them to be considered as a single logical unit. The abstraction has been shown correct, and to improve the feasibility of verification [45]. The abstraction is implemented using the Epsilon Transformation Language (ETL) [42] that is designed for model transformations. Fig. 8 gives an excerpt of our transformation. The algorithm uses the following list structures: `toDelete`: storing units to be removed and `consToBeMapped`: storing which connectors require renaming.

The `abs` rule performs as follows: line 1 states that the rule translates the given rail diagram `rd` to another `rd2`. The second line simply calls an operation `computeAbstraction()` on `rd` to compute which units can be collapsed and to populate the lists with appropriate values. Next, the algorithm will consider every unit `ut` within `rd` (line 3). If `ut` is not in the list `toDelete` (line 4), then the algorithm will perform analysis on the connectors of `ut`. If connector one of `ut` is within the set of connectors requiring renaming (line 5), then the first connector of `ut` is renamed using a call to the operation `getMapping()` (line 6). Lines 8 to 9 of the algorithm perform these steps for connector C2 of `ut`. After this computation, the modified unit `ut` is added as an element to `rd2` (line 11). The algorithm continues in a similar manner, computing which connectors and signals should be added to `rd2`. Finally, an operation `computeTables` is called to compute a new control table for `rd2`.

Importantly, we note, that any domain level abstractions implemented in OnTrack (such as the one presented) can be applied before the model transformations that generate formal models. This allows reuse of these abstractions for the currently supported formal models in OnTrack, as well as reuse for any languages that are added in the future. However, the ability to apply the transformation in OnTrack does not mean that the abstraction is semantically sound within the given formal model, and indeed the abstraction would need to be proven correct based upon the semantics of the formal language. As an example of such a proof we refer the reader to [36].

5. Challenges in input formats and counterexample visualisation

Throughout the development of OnTrack, there were a number of desired features from a railway verification perspective that did not fit naturally with the available MDE frameworks at the time. In particular, both involved interfacing OnTrack with other tools and formats that were not defined in the traditional sense of a meta-model or DSL. We now discuss these challenges.

5.1. Importing data

In the railway verification community, there are often issues around interoperability of tools due to a plethora of data formats that are used by the different organisation across the railway domain. Here, a limitation with many existing toolsets is that users are often required to re-draw and re-enter railway layouts directly into the verification toolset. Re-entering data is clearly cumbersome, open to errors and time consuming. Similarly, automated importation of verification data tends to be hard as geospatial information on positioning of language elements is often missing from the data.

Here, we had hoped to be able to develop a model transformation from existing data formats (for example Brave [12] and RailML [46]), however it was a challenge that after initial investigations looked to be too cumbersome. Hence the decision was taken to implement the import as an ad-hoc Java plugin.

5.1.1. Well defined data

Here, considering for example the Brave data format, the following aspects played a role in this decision:

- No defined language model: The lack of a well defined meta-model for the data meant that one would need to be developed; here the size of the Brave language is much larger than the OnTrack DSL and thus this endeavour seemed too large.
- Well Formed Models: Given the lack of a language definition, there were a number of concerns around importing incomplete complex models that may work fine with the Brave toolset, but would require the (likely manual task) of adding data to make a well formed OnTrack model. For example, a simple missing end of track marker at the edge of the scheme plan would work within simulations in Brave, but would not be a well-formed model within OnTrack.
- Granularity and Terminology: The Brave data format contains many added interconnected domain entities that were not needed in the setting of OnTrack. These differences in granularity often led to overlapping terminology that represented domain entities in a conflicting manner from varied viewpoints. For example, train detection units could cover multiple tracks and points in Brave, whereas within OnTrack this granularity is not present and is abstracted into a single unit of detection.

MDE Challenge 1 – Supporting less structured and non well-formed models: Overall, the developers of OnTrack felt that the well-structured nature of data often found within MDE tools was not matched by the less structured data that we would like to import. Thus dealing with abnormalities seemed much easier from a generic programming language perspective. Here we feel there could be an interesting area of research in dealing with non well-formed models or similar within MDE settings.

5.1.2. Model layout and geospatial information

In addition to being able to import data, the graphical nature of our models means that geospatial information on layout of model entities is essential for viewing and working with the models. However, we note that such geospatial information is not necessary for the verification of control tables, which is only concerned with the topological nature of the railway. Such geospatial information is often not available when importing models, or is very complex to deal with. Here we had hoped that GMF would offer a sensible graph layout option or algorithm that produced a somewhat workable result for railway engineers (even if not matching the real world scheme plan layout). However, essentially any imported model is displayed graphically in a hierarchical list of entities. This led to our imported models being completely unusable for domain experts, and is indeed one of the limitations of MDE frameworks such as GMF, where geospatial information is not considered as part of the models semantics.

To overcome this, we developed a best-fit approach to track layout that we then tested with end users [47]. This approach is based upon simulated annealing, which makes incrementally smaller changes to the layout over thousands of iterations, with an aim of optimising a defined score function. In terms of implementation, this feature took considerable effort for development of the simulated annealing algorithm. However, linking this to interact with the layout of elements within GMF was relatively simple once the initial hurdle of understanding the construction of GMF models had been overcome. We note that the implementation does not follow any systematic MDE principles, and is rather generic in terms of approach, which in turn may hinder maintainability of the toolset as MDE frameworks evolve. The approach is incorporated as a simple Java based import plug-in.

MDE Challenge 2 – Consider geospatial information as part of a models semantics: We feel that layout of entities in DSLs is something that would be fruitful for the MDE community to explore. Where perhaps the solution is in-fact a DSL that allows specification of how entities should be geospatially represented in relation to one-another. In addition, to aid in uptake of MDE tools, better documentation and examples (at the time of development of OnTrack) would have provided greater support for such ad-hoc plugins.

5.2. Counterexample visualisation

Finally, OnTrack aims to encapsulate formal methods for proving correctness of railway designs with regards to safety. When designing signalling systems, engineers often step through problems and safety issues like a mathematician would step through the lines of a proof. As OnTrack generates formal models from graphical specifications, it is highly desirable that any failed proofs, or counterexamples to a safety property be presented once again at the level of the graphical specification within the graphical DSL.

Here, there is once again a trade-off as failed proof counterexample traces are typically described at the low level of the proof tool involved (indeed sometimes lower than the generated formal models). Hence a reverse model transformation would be somewhat cumbersome to try to construct. Often such counterexample traces also contain not just a single “state”, but thousands of consecutive states, leading from initial conditions through to where the safety property is violated. Hence a graphical visualisation must not only be perceptually effective, it must also support the cognitive map with which railway engineers and formal methods researchers approach the problem.

To overcome this, we once again implemented an ad-hoc plugin that only supports CSP models (here a unique backwards transformation is needed for each formal method). We provide an interactive step-through approach via next/previous buttons, similar to a debugger. The various states of the system are displayed within the GMF editor (see Fig. 9, which highlights the dynamic state changes for a simple counterexample that ends with two trains occupying the same track highlighted in red). Again this approach was subjected to a user study [47] to help determine which states should be displayed (i.e. key frames), rather than displaying the possibly thousands of states for a given counterexample.

Overall here, the implementation effort was minimal, where familiarity with GMF aided in the development. However, we do note that some changes were made to the underlying DSL meta-model to support this functionality. For example to capture state information in a nice way, certain fields were added to domain elements.

MDE Challenge 3 – Animation of models: Overall, the authors feel that there could be a better solution to counterexample visualisation through some form of graphical DSL that supports animation. Indeed this perhaps could be achieved through a dedicated DSL/GMF editor combination that captures and supports a generic form of counterexample and interactions with it. From this, it would also be interesting to explore if bi-directional model transformations could be used to support a close link between DSL models and counterexamples from each formal method.

6. Reflections on model driven engineering for formal methods

The aims of the OnTrack toolset were to deal with shortcomings surrounding the uptake of formal methods by industry [38,4,20] due to questions around:

Faithful modelling: Do the proposed mathematical models faithfully represent the systems of concern? Modelling approaches offered from Computer Science are often in a form that is acceptable to computer scientists, but not to the engineer working within the domain. How can an engineer working within the domain come up with new models?

Scalability: Does the proposed technology scale up to industrially sized systems in a manner that is uniformly applicable? Often, formal methods have been applied in a pilot to specific systems, but require individual, hand-crafted adaptation and optimisation for each new system under consideration.

Usability: Are the methods accessible to practitioners in the domain of interest or is it just the developers of the approach who can apply them? Handling of tools for verification procedures is often aimed towards a Computer Science audience specialised in verification, however they are usually not manageable by engineers outside the field of formal methods.

On each of these fronts, utilising MDE frameworks to implement a tool environment has had a number of both expected and unforeseen benefits. In particular:

Takeaway 1 – DSLs and model transformations support faithful modelling well: Confidence in a formal model actually providing a *faithful model* of the domain has been increased. Here, both the use of OnTrack’s *DSL*, that is well understood by railway engineers, and the use of *model transformations*, that are highly systematic and easy to read (compared to an ad-hoc translation), have improved confidence of Siemens Rail Engineers in the formal models we have created. Interestingly, the tool has also allowed for rich discussions in a manner that overcomes language barriers between domain experts and formal methods researchers, supporting further refinement of the formal models and verification processes.

Takeaway 2 – Model transformations are a good approach to implementing abstractions: OnTrack has presented a new form of *scalability* for formal methods, in that it supports domain centred abstractions that can be utilised by any underlying formal method. Here, the development of domain level abstractions [36] were in fact inspired by the idea of DSL level model-to-model transformations. In addition, the fact that such abstractions are implemented as *model transformations* independent of the formal method, means that if a new formal method is added to OnTrack, it can benefit from these abstractions without any additional

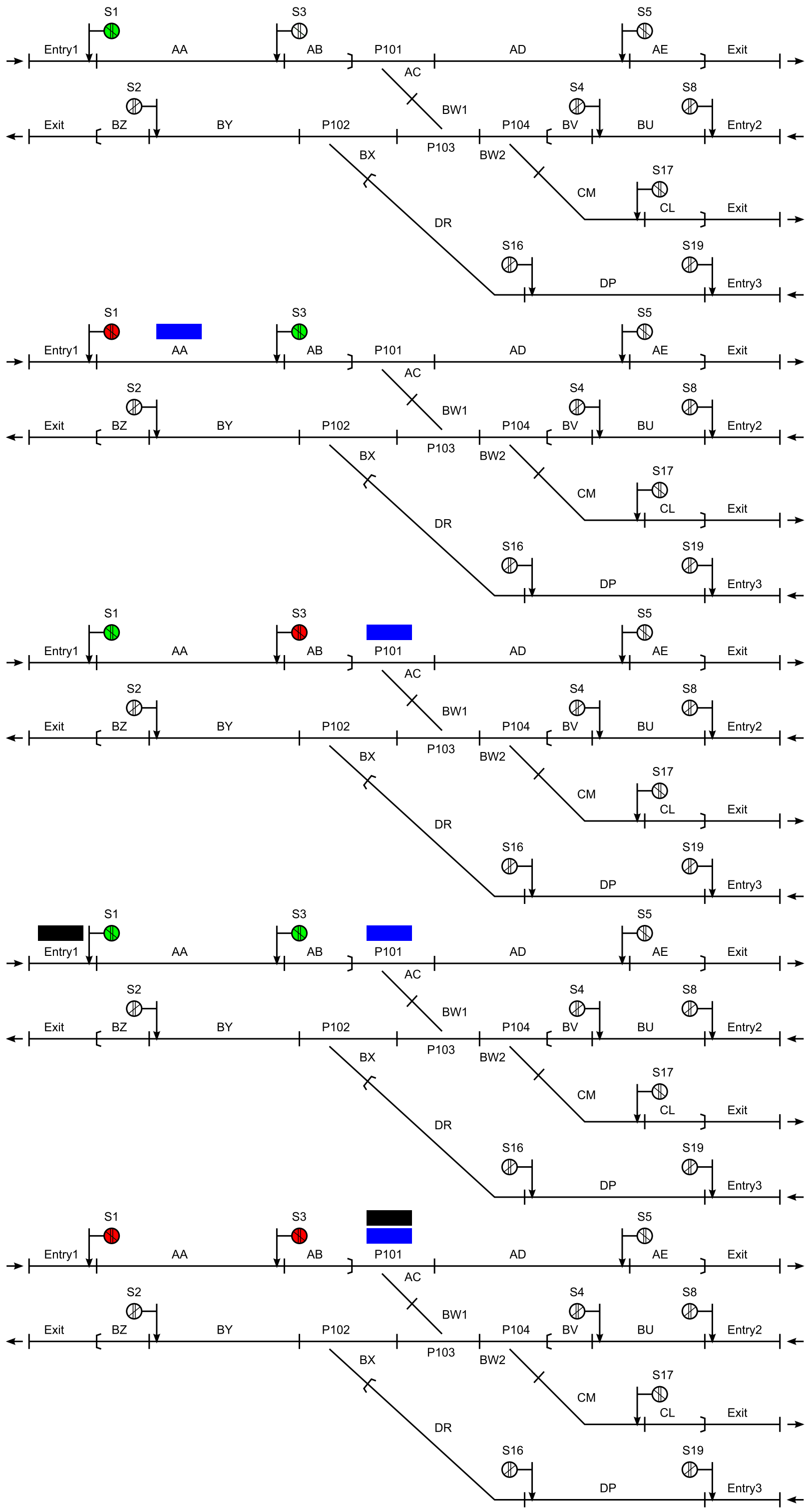


Fig. 9. Counterexample visualisation via key frames. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

implementation work.[1] As these abstractions are also written in a clear and concise way as *model transformations*, there is again increased confidence in the correctness of the implementation, which is especially important within the setting of formal methods.

[1] We note that there may be manual effort in constructing a semantics level proof, but this would have to be completed for any abstraction anyway.

Takeaway 3 – MDE tools were successful in allowing rapid development of a graphical tool to improve usability of formal methods: Thanks to OnTrack's *DSL* and its *GMF* based editor, OnTrack succeeds in being highly usable by railway engineers [47] by encapsulating formal model generation. Here, although not without some challenges (see Section 5), the use of *GMF* allowed for rapid development of a modelling environment that allows rail engineers to create, edit and fine-tune railway scheme plan designs. In addition, simple wizards have been implemented in Epsilon's Wizard Language that allow automation of several tasks, such as generation of control table data.

7. Conclusions and future ideas

In this paper, we have reviewed and reflected upon the implementation of OnTrack, a tool for automating workflows for railway verification. Overall, we believe the OnTrack project has been a success as a demonstrator of how model driven engineering can be applied to develop domain specific toolsets for formal methods. Indeed, OnTrack is still in use today by researchers developing formal models particularly at Swansea University, Coventry University and Siemens Rail Research. However, we note that there are barriers to the use of OnTrack within the setting of safety cases for the railway domain. In particular, many of the implementation techniques do not meet the relevant safety compliance checks required for safety critical systems (e.g. qualification of a T2 level tool according to EN50128 [13]) and thus any likely usage outside of industrial research departments would require re-development of the tool, in a likely more bespoke manner. However, the ideas explored in OnTrack have influenced the views on rich data models and also data transformations within Siemens Rail.

As experts in formal methods, the authors have developed a particular fondness towards the benefits of using DSLs and model transformations. Not only did GMF provide a low barrier to entry for developing a graphical specification tool, but formulating specification generation and abstractions as model transformations provides a real clarity both conceptually and in terms of both traceability and readability within the implementation. This level of traceability we feel is core to maintain faith in any formal methods tool. In addition, OnTrack has supported collaboration between academic groups benefiting from the tool without the need for a large implementation effort, but by simply developing translations to their formal models. For example, OnTrack was an integral part of the DITTO research project (see http://dittorailway.uk) allowing traffic engineers and transport operations researchers (from University of Leeds and University of Southampton), railway engineering researchers (from Birmingham University) and computer scientists (from Swansea University) to explore fundamental principles for the optimisation of rail operations.

Finally, the authors would like to further explore the use of Model driven approaches within OnTrack. In particular, we feel there is scope for improving upon the argument of faithful modelling by utilising validation languages such as EVL [42] to show correctness of the implemented model transformations. In addition, we are keen to explore the development of a separate tool that encompasses a family of DSLs to capture various forms of counterexamples as generated by various proof tools. Here, we feel a generic graphical framework that can be specialised to a particular domain and that supports presentation and interaction with failed proofs would provide many benefits to both the formal methods community and domain experts wishing to use formal methods. We envisage that perhaps bi-directional model transformations [55] would play a key role in such an endeavour.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank Siemens Rail Automation UK and the Rail Safety and Standards Board, RSSB, for supporting this work. We also wish to highlight the rich insights provided by Simon Chadwick, Mark Thomas, Thomas Werner and Andrew Lawrence from Siemens Rail Automation UK. We extend a special thanks to our academic collaborators, in particular, Prof. Markus Roggenbach, Dr. Hoang Nga Nguyen, Prof. Helen Treharne and Dr. Monika Seisenberger for their valued insights and contributions to the development of OnTrack.

References

[1] J.-R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.
[2] J.-R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, L. Voisin, Rodin: an open toolset for modelling and reasoning in Event-B, Int. J. Softw. Tools Technol. Transf. 12 (6) (2010) 447–466.
[3] M. Banci, A. Fantechi, S. Gnesi, Some experiences on formal specification of railway interlocking systems using statecharts, Technical report, CNR-ISTI, Pisa, Italy, 2005, TRain Workshop at SEFM 2005 (Software Engineering and Formal Methods), Koblenz, Germany, September 5–9, 2005.
[4] D. Basile, M.H. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, A. Piattino, D. Trentini, A. Ferrari, On the industrial uptake of formal methods in the railway domain, in: C.A. Furia, K. Winter (Eds.), Integrated Formal Methods, Springer, 2018, pp. 20–29.
[5] S. Beydeda, M. Book, V. Gruhn (Eds.), Model-Driven Software Development, Springer, 2005.
[6] D. Bjørner, Formal software techniques for railway systems, in: CTS2000: 9th IFAC Symposium on Control in Transportation Systems, 2000, pp. 1–12.
[7] D. Bjørner, Dynamics of railway nets: on an interface between automatic control and software engineering, in: CTS2003: 10th IFAC Symposium on Control in Transportation Systems, 2003.
[8] D. Bjørner, C. George, S. Prehn, Scheduling and rescheduling of trains, in: Industrial Strength Formal Methods in Practice, Springer, 1999, pp. 157–184, chapter 8.

[9] J. Boulanger, M. Gallardo, Validation and verification of METEOR safety software, in: J. Allen, R.J. Hill, C.A. Brebbia, G. Sciutto, S. Sone (Eds.), Computers in Railways VII, vol. 7, WIT Press, 2000, pp. 189–200.
[10] J.P. Bowen, M.G. Hinchey, Ten commandments of formal methods... ten years later, IEEE Comput. 39 (1) (2006) 40–48.
[11] Y. Cao, T. Xu, T. Tang, H. Wang, L. Zhao, Automatic generation and verification of interlocking tables based on domain specific language for computer based interlocking systems, in: Proceedings of the IEEE International Conference on Computer Science and Automation Engineering, CSAE 2011, vol. 2, IEEE, 2011, pp. 511–515.
[12] L. Chen, P. James, D. Kirkwood, H.N. Nguyen, G.L. Nicholson, M. Roggenbach, Towards integrated simulation and formal verification of rail yard designs-an experience report based on the UK East Coast Main Line, in: 2016 IEEE International Conference on Intelligent Rail Transportation (ICIRT), IEEE, 2016, pp. 347–355.
[13] European Union, Railway applications-communication, signalling and processing systems: Software for railway control and protection systems, En-50128, CENELEC, 2011.
[14] A. Fantechi, S. Gnesi, A.E. Haxthausen, Formal methods for distributed control systems of future railways, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Practice - 11th International Symposium, ISoLA 2022, Proceedings, Part IV, Rhodes, Greece, October 22–30, 2022, in: Lecture Notes in Computer Science (LNCS), vol. 13704, Springer, 2022, pp. 243–245.
[15] A. Fantechi, G. Gori, A.E. Haxthausen, C. Limbrée, Compositional verification of railway interlockings: comparison of two methods, in: S.C. Dutilleul, A.E. Haxthausen, T. Lecomte (Eds.), Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - 4th International Conference, Proceedings, RSSRail 2022, Paris, France, June 1–2, 2022, in: Lecture Notes in Computer Science (LNCS), vol. 13294, Springer, 2022, pp. 3–19.
[16] A. Ferrari, G. Magnani, D. Grasso, A. Fantechi, Model checking interlocking control tables, in: FORMS/FORMAT 2010, Springer, 2011, pp. 107–115.
[17] A. Ferrari, F. Mazzanti, D. Basile, M.H. ter Beek, Systematic evaluation and usability analysis of formal methods tools for railway signaling system design, IEEE Trans. Softw. Eng. 48 (11) (2022) 4675–4691.
[18] A. Ferrari, F. Mazzanti, D. Basile, M.H. ter Beek, A. Fantechi, Comparing formal tools for system design: a judgment study, in: Proceedings of the 42nd International Conference on Software Engineering (ICSE 2020), ACM, 2020, pp. 62–74.
[19] A. Ferrari, M.H. ter Beek, F. Mazzanti, D. Basile, A. Fantechi, S. Gnesi, A. Piattino, D. Trentini, Survey on formal methods and tools in railways: the ASTRail approach, in: S.C. Dutilleul, T. Lecomte, A.B. Romanovsky (Eds.), Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - Third International Conference, Proceedings, RSSRail 2019, Lille, France, June 4–6, 2019, in: Lecture Notes in Computer Science (LNCS), vol. 11495, Springer, 2019, pp. 226–241.
[20] H. Garavel, M.H. ter Beek, J.v.d. Pol, The 2020 expert survey on formal methods, in: M.H. ter Beek, D. Ničković (Eds.), Formal Methods for Industrial Critical Systems, Springer, 2020, pp. 3–69.
[21] M. Gleirscher, D. Marmsoler, Formal methods in dependable systems engineering: a survey of professionals from Europe and North America, Empir. Softw. Eng. 25 (6) (2020) 4473–4546.
[22] R.C. Gronback, Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit, Addison-Wesley Professional, 2009.
[23] A. Haxthausen, J. Peleska, A domain-oriented, model-based approach for construction and verification of railway control systems, in: C. Jones, Z. Liu, J. Woodcock (Eds.), Formal Methods and Hybrid Real-Time Systems, in: Lecture Notes in Computer Science (LNCS), vol. 4700, Springer, 2007, pp. 320–348.
[24] A.E. Haxthausen, Towards a framework for modelling and verification of relay interlocking systems, in: 16th Monterey Workshop: Modelling, Development and Verification of Adaptive Systems: the Grand Challenge for Robust Software, in: Lecture Notes in Computer Science (LNCS), vol. 6662, Springer, 2011, pp. 176–192.
[25] A.E. Haxthausen, Automated generation of formal safety conditions from railway interlocking tables, Int. J. Softw. Tools Technol. Transf. 16 (6) (2014) 713–726, Special Issue on Formal Methods for Railway Control Systems.
[26] A.E. Haxthausen, M.L. Bliguet, A.A. Kjær, Modelling and verification of relay interlocking systems, in: Foundations of Computer Software, Future Trends and Techniques for Development, 15th Monterey Workshop, in: Lecture Notes in Computer Science (LNCS), vol. 6028, Springer, 2010, pp. 141–153.
[27] A.E. Haxthausen, A. Fantechi, Compositional verification of railway interlocking systems, Form. Asp. Comput. 35 (1) (2023) 4.
[28] A.E. Haxthausen, A.A. Kjær, M.L. Bliguet, Formal development of a tool for automated modelling and verification of relay interlocking systems, in: M.J. Butler, W. Schulte (Eds.), FM 2011, in: LNCS, vol. 6664, Springer, 2011, pp. 118–132.
[29] A.E. Haxthausen, J. Peleska, S. Kinder, A formal approach for the construction and verification of railway control systems, Form. Asp. Comput. 23 (2) (2011) 191–219.
[30] A.E. Haxthausen, J. Peleska, R. Pinger, Applied bounded model checking for interlocking system designs, in: Software Engineering and Formal Methods, in: Lecture Notes in Computer Science (LNCS), vol. 8368, Springer, 2014, pp. 205–220.
[31] G. Holland, T. Kahsai, M. Roggenbach, B.H. Schlingloff, Towards formal testing of jet engine Rolls-Royce BR725, in: L. Czaja, M. Szczuka (Eds.), Proceedings 18th International Conference on Concurrency, Specification and Programming, Springer, 2009.
[32] A. Iliasov, I. Lopatkin, A. Romanovsky, Practical formal methods in railways – the SafeCap approach, in: Reliable Software Technologies, Ada-Europe 2014, Proceedings, 2014, pp. 177–192.
[33] A. Iliasov, D. Taylor, L. Laibinis, A.B. Romanovsky, Formal verification of signalling programs with SafeCap, in: B. Gallina, A. Skavhaug, F. Bitsch (Eds.), Computer Safety, Reliability, and Security - 37th International Conference, SAFECOMP 2018, Västerås, Sweden, September 19–21, 2018, in: Lecture Notes in Computer Science (LNCS), vol. 11093, 2018, pp. 91–106.
[34] P. James, SAT-based Model Checking and its Applications to Train Control Software, Master's thesis, Swansea University, 2010.
[35] P. James, Designing Domain Specific Languages for Verification and Applications to the Railway Domain, PhD thesis, Swansea University, 2014.
[36] P. James, F. Moller, H.N. Nguyen, M. Roggenbach, S. Schneider, H. Treharne, Techniques for modelling and verifying railway interlockings, Int. J. Softw. Tools Technol. Transf. 16 (6) (Nov 2014) 685–711.
[37] P. James, F. Moller, H.N. Nguyen, M. Roggenbach, S.A. Schneider, H. Treharne, On modelling and verifying railway interlockings: tracking train lengths, Sci. Comput. Program. 96 (2014) 315–336.
[38] P. James, M. Roggenbach, Encapsulating formal methods within domain specific languages: a solution for verifying railway scheme plans, Math. Comput. Sci. 8 (1) (2014).
[39] P. James, M. Trumble, H. Treharne, M. Roggenbach, S. Schneider. Ontrack, An open tooling environment for railway verification, in: G. Brat, N. Rungta, A. Venet (Eds.), NASA Formal Methods, in: Lecture Notes in Computer Science (LNCS), vol. 7871, Springer, 2013, pp. 435–440.
[40] K. Kanso, Agda as a Platform for the Development of Verified Railway Interlocking Systems, PhD thesis, Department of Computer Science, Swansea University, UK, August 2013.
[41] S. Kent, Model driven engineering, in: M. Butler, L. Petre, K. Sere (Eds.), Integrated Formal Methods, in: Lecture Notes in Computer Science (LNCS), vol. 2335, Springer, 2002, pp. 286–298.
[42] D. Kolovos, L. Rose, R. Paige, F. Polack, The Epsilon Book, 2013.
[43] B. Luteberget, K. Claessen, C. Johansen, Automated drawing of railway schematics using numerical optimization in SAT, in: W. Ahrendt, S.L. Tapia Tarifa (Eds.), Integrated Formal Methods, Springer, 2019, pp. 341–359.
[44] A. Mirabadi, M.B. Yazdi, Automatic generation and verification of railway interlocking control tables using FSM and NuSMV, Transp. Probl. 4 (2009) 103–110.
[45] F. Moller, H.N. Nguyen, M. Roggenbach, S. Schneider, H. Treharne, Defining and model checking abstractions of complex railway models using CSP||B, in: A. Biere, A. Nahir, T. Vos (Eds.), Hardware and Software: Verification and Testing, in: Lecture Notes in Computer Science (LNCS), vol. 7857, Springer, 2013.

[46] A. Nash, D. Huerlimann, J. Schütte, V.P. Krauss, RailML – a standard data interface for railroad applications, in: WIT Transactions on the Built Environment, 2004, p. 74.

[47] F. Pantekis, P. James, L. O'Reilly, D. Archambault, F. Moller, Visualising railway safety verification, in: Formal Techniques for Safety-Critical Systems: 7th International Workshop, FTSCS 2019, Shenzhen, China, November 9, 2019, in: Communications in Computer and Information Science, vol. 1165, Springer, 2020, pp. 95–105, Revised Selected Papers.

[48] J. Peleska, D. Große, A.E. Haxthausen, R. Drechsler, Automated verification for train control systems, in: E. Schnieder, G. Tarnai (Eds.), Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems, Technical University of Braunschweig, 2004.

[49] J. Peleska, N. Krafczyk, A.E. Haxthausen, R. Pinger, Efficient data validation for geographical interlocking systems, in: S.C. Dutilleul, T. Lecomte, A.B. Romanovsky (Eds.), Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - Third International Conference, Proceedings, RSSRail 2019, Lille, France, June 4–6, 2019, in: Lecture Notes in Computer Science (LNCS), vol. 11495, Springer, 2019, pp. 142–158.

[50] J. Peleska, N. Krafczyk, A.E. Haxthausen, R. Pinger, Efficient data validation for geographical interlocking systems, Form. Asp. Comput. 33 (6) (2021) 925–955.

[51] A. Romanovsky, F. Moller, M. Roggenbach, Overcoming the railway capacity challenges without undermining rail network safety (SafeCap), UKRI (EPSRC) Project EP/I010807/1, 2011-2013.

[52] A. Simpson, A formal specification of an automatic train protection system, in: G. Goos, J. Hartmanis, J. van Leeuwen (Eds.), FME'94: Proceedings of the Second International Symposium of Formal Methods Europe on Industrial Benefit of Formal Methods, in: Lecture Notes in Computer Science (LNCS), vol. 873, Springer, 1994.

[53] P. Stankaitis, A. Iliasov, Safety verification of modern railway signalling with the SafeCap platform, in: 2017 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, IEEE Computer Society, Toulouse, France, October 23–26, 2017, pp. 153–156, 2017.

[54] D. Steinberg, F. Budinsky, E. Merks, M. Paternostro, EMF: Eclipse Modeling Framework, Pearson, 2008.

[55] P. Stevens, A landscape of bidirectional model transformations, in: R. Lämmel, J. Visser, J. Saraiva (Eds.), Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2–7, 2007, in: Lecture Notes in Computer Science (LNCS), vol. 5235, Springer, 2008, pp. 408–424, Revised Papers.

[56] D. Tombs, N. Robinson, G. Nikandros, Signalling control table generation and verification, in: Proceedings of Cost Efficient Railways Through Engineering, CORE 2002, Railway Technical Society of Australasia, 2002, pp. 415–425.

[57] L.H. Vu, A.E. Haxthausen, J. Peleska, Formal modeling and verification of interlocking systems featuring sequential release, in: Formal Techniques for Safety-Critical Systems, in: Communications in Computer and Information Science, vol. 476, Springer International Publishing, Switzerland, 2015, pp. 223–238.

[58] L.H. Vu, A.E. Haxthausen, J. Peleska, A domain-specific language for generic interlocking models and their properties, in: A. Fantechi, T. Lecomte, A.B. Romanovsky (Eds.), Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - Second International Conference, Proceedings, RSSRail 2017, Pistoia, Italy, November 14–16, 2017, in: Lecture Notes in Computer Science (LNCS), vol. 10598, Springer, 2017, pp. 99–115.

[59] K. Winter, Model checking railway interlocking systems, Aust. Comput. Sci. Commun. 24 (1) (2002) 303–310.

[60] K. Winter, Optimising ordering strategies for symbolic model checking of railway interlockings, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, in: Lecture Notes in Computer Science (LNCS), vol. 7610, Springer, 2012, pp. 246–260.

[61] K. Winter, W. Johnston, P. Robinson, P. Strooper, L. van den Berg, Tool support for checking railway interlocking designs, in: 10th Australian Workshop on Safety Critical Systems and Software, SCS'05, Proceedings, vol. 55, Australian Computer Society, Inc., 2006, pp. 101–107.

[62] K. Winter, N.J. Robinson, Modelling large railway interlockings and model checking small ones, in: M.J. Oudshoorn (Ed.), ACSC'03: Proceedings of the 26th Australasian Computer Science Conference, Australian Computer Society, 2003.