A Domain-specific Language for Railway Interlocking Systems

Linh H. Vu[1], Anne E. Haxthausen[1], and Jan Peleska[2]

[1] DTU Compute, Technical University of Denmark, Denmark
[2] Department of Mathematics and Computer Science
University of Bremen, Germany

Abstract. This paper presents a domain-specific language (DSL) for describing route-based interlocking systems which are compatible with European Train Control System ETCS Level 2. The abstract syntax and static semantics of the language are formally defined using the RAISE Specification Language (RSL). Furthermore, the paper describes an interlocking table generator (ITG) that generates automatically a well-formed interlocking table from a well-formed railway network layout. Experiments with the DSL and ITG using the RAISE tools and the C++ implementation show that the use of the DSL and ITG can increase the productivity and significantly reduce errors in the specifications of railway interlocking systems.

Key words: domain-specific languages, interlocking tables, validation and verification, railway signalling systems, formal methods

1 Introduction

In the period 2009–2021 all Danish railway signalling systems will be replaced with the standardized European Train Control System (ETCS) Level 2 [3]. A vital part of these new systems is the interlocking systems that are responsible for setting safe train routes in the railway network. One of the goals of the RobustRailS research project[1] is to establish a holistic method supporting the development and verification of these new interlocking systems and in particular to provide *automated* construction and verification of the interlocking tables. This should overcome the time consuming, error prone process of manual construction and verification that was used in the past. As interlocking systems have the highest safety integrity level (SIL4), *formal* development and verification methods are strongly recommended by the CENELEC EN50128 standard [4]. Furthermore, the use of *domain-specific languages* for software development has shown to ease the communication between domain experts, software engineers, and certification authorities. Therefore, our RobustRailS method combines the use of formal methods with domain-specific languages.

The main contribution of this paper is a formal specification of: (i) the abstract syntax and static semantics of a domain-specific language (DSL) for

[1] http://www.robustrails.man.dtu.dk/

describing route-based interlocking systems which are compatible with ETCS Level 2, and (ii) an interlocking table generator (ITG) that, for a given well-formed railway network layout, generates a statically well-formed interlocking table. The specification of the DSL and the ITG are directly executable, but have also been translated into C++ and integrated as part of the interlocking toolbox in RT-Tester tool-chain [14] as shown in Fig. 1. The DSL and the ITG serve as a front-end for providing and validating configuration data for a generic model of the dynamic behaviour of the new Danish interlocking systems. After being instantiated with configuration data, the model can be verified against safety properties – such as non-collision and non-derailment – using model checking and inductive reasoning.

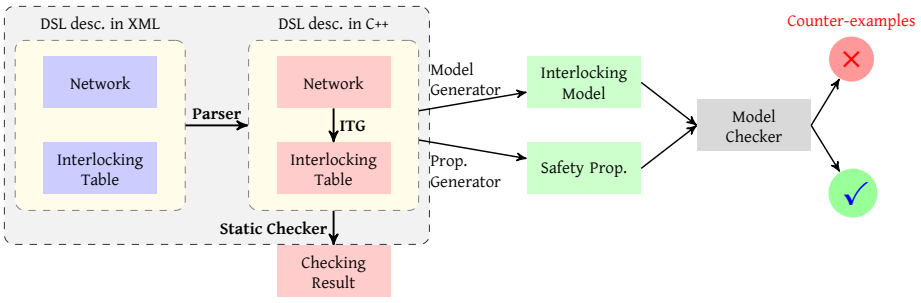

Fig. 1. Development process and tool-chain for railway interlocking software systems. The work presented in this paper is marked within the gray dashed box.

The paper is organized as follows: Sec. 2 presents the specification of DSL's abstract syntax in the RAISE Specification Language (RSL) [6], and Secs. 3 and 4 outline the principles of the static semantics of the language and the ITG. Sec. 5 discusses the implementation of the language and its applications. Finally related work and concluding remarks are presented in Sec. 6 and Sec. 7. All RSL specifications can be found at http://www2.compute.dtu.dk/~lvho/.

2 Abstract Syntax

A description of an interlocking system in the proposed DSL consists of a railway network layout and an interlocking table. In the following two subsections, the abstract syntaxes of network layouts and interlocking tables are specified as types in RSL.

2.1 Railway Network Layouts

A railway network layout[2] is a description of the topology of a railway network. Each element in a network layout is given a unique id. Therefore, we introduce types for ids of track sections, marker boards, and level crossings, respectively.

[2] In the remaining of this paper, the terms *network*, *network layout*, *railway network layout* are used interchangeably.

SecId = **Text**, MbId = **Text**, LcId = **Text**

A network layout is represented as a record consisting of four maps – one for each kind of element – mapping the ids of the elements into geographical information about these elements.

NetworkLayout ::
  linears : SecId $\xrightarrow[m]{}$ Linear
  points : SecId $\xrightarrow[m]{}$ Point
  marker_boards : MbId $\xrightarrow[m]{}$ MarkerBoard
  level_crossings : LcId $\xrightarrow[m]{}$ LevelCrossing

The information recorded about a linear section is composed of information about its neighboring sections and its length. A linear section may have up to two neighbors: one at the *down* end and at the *up* end[3].

Linear ::
  neighbors : LinearEnd $\xrightarrow[m]{}$ SecId
  length : Distance,
Direction == DOWN | UP, LinearEnd = Direction, Distance = **Nat**

For each point section similar information is recorded. In this case there are up to three neighboring sections: one at the *stem* end, one at the *plus* end, and one at the *minus* end. The length of a point section is the distance from the stem tip to the plus (or minus) tip.

Point ::
  neighbors : PointEnd $\xrightarrow[m]{}$ SecId
  length : Distance,
PointEnd == STEM | PLUS | MINUS

The information recorded about a marker board includes: the id of the section along which it is placed, the travel direction (up or down) that it is intended for, and the distance from the location where it is placed to the tip of the section in the marker board's travel direction, as illustrated in Fig. 2.

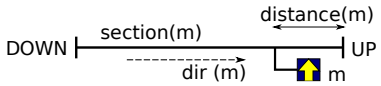

Fig. 2. A marker board m and its associated geographical information.

MarkerBoard ::
  section : SecId
  dir : Direction
  distance : Distance

[3] In Denmark *up* and *down* denote the directions in which the distance from a certain reference location is increasing and decreasing, respectively.

The information recorded about a level crossing is the (non-empty) set of the (parallel) sections that it covers.

LevelCrossing :: crossed_secs : SecId-**set**

2.2 Interlocking Tables

In order to guide trains safely through a railway network, the interlocking system reserves exclusively a fraction of the network, called a *route*, for a train at a time. Contrary to legacy systems, in ETCS Level 2, there are no physically signals (along the tracks), but *virtual signals*[4]. A virtual signal is associated with a marker board, and has the same geographical information as the marker board. The aspects of virtual signals are used to calculate the movement authorities determining how far forward trains are allowed to move [3]. A *route* is defined as a path from a *source* signal to (another) *destination* signal. Both signals are in the direction (up or down) of the route. A route is said to be *elementary* if there does not exist a signal which is placed between the source and the destination of the route and which has the same direction as the route.

An *interlocking table* specifies the elementary routes in a given network and the specification for setting these routes. A specification of a route includes: (i) the list of the sections in the *path* from the source to the destination, (ii) the list of the sections used as the *overlaps*, (iii) a map from *points* used by the route to their required positions (`PLUS` or `MINUS`), (iv) a set of *protecting signals*, (v) a set of *level crossings* covering the sections in the route's path and overlaps, and (vi) a set of routes that are in *conflict* with the route, and therefore must not be set at the same time as the given route.

An interlocking table is naturally represented as a map from the id of each route into a record containing the specification of the route.

InterlockingTable = RouteId $\overrightarrow{m}$ Route,
Route ::
  source : MbId
  dest : MbId
  path : SecId*
  overlaps : SecId*
  points : SecId $\overrightarrow{m}$ PointPos
  signals : MbId-**set**
  lcs : LcId-**set**
  conflicts : RouteId-**set**,
RouteId = **Text**, PointPos == PLUS | MINUS

3 Static Semantics

The static semantics of the DSL presented in Sec. 2 are specified by predicates in RSL. A DSL description of an interlocking system is well-formed if its network

[4] The term *virtual signal* is abbreviated to *signal* in the remaining of the paper.

and interlocking table both are well-formed. For example, the predicate for checking whether an interlocking table is well-formed w.r.t. a network layout is defined with the following signature in RSL

is_wf : InterlockingTable × NetworkLayout → **Bool**

This predicate is the conjunction of a number of sub-predicates that specify well-formedness conditions, e.g. a route should have proper protection provided by protecting signals and/or protecting points, or two routes that are physically in conflict in the network must be listed in the routes' specification as being in conflict. Note that the route's protection and conflicting routes are automatically computed from the network layout. Afterwards, they are compared with the route's specification in order to determine whether the specification is correct.

Details about all well-formedness conditions can be found in the published RSL specification of the DSL.

4 Interlocking Table Generator

This section describes an interlocking table generator (ITG) that, for a given well-formed network, generates a statically well-formed interlocking table. The generator is formally specified in RSL as a function with the following signature:

mk_table : NetworkLayout $\xrightarrow{\sim}$ InterlockingTable

The function basically collects the specification of all elementary routes in the given network. For each marker board s, the generator constructs the specification of routes starting from s by traversing the network starting from the first section right after s. The following information is collected during the traversal: (i) the destination signal, (ii) the route's path and overlaps, (iii) level crossings covering the sections in the path and overlaps, (iv) signals for front and flank protection of the path and overlaps, and (v) required positions of points in the path and overlaps, and protecting points and required positions of these.

Repeating the procedure with all marker boards, we obtain a set of route specifications. For each route specification, we try to find its alternatives by replacing a subset of the route's protecting points with substitute protecting signals if it is possible. This results in a complete set of all elementary routes for the given network. Afterwards, we assign a unique id to each route, and for each pair of routes determine whether they are physically in conflict in the given network layout. Eventually, a statically well-formed interlocking table for the given network layout is produced.

5 Applications

The RSL specifications of the DSL and the ITG are executable, thus they can be tested directly. Let us consider an example of a typical network with two tracks as shown in Fig. 3. This network can be specified in RSL as a value n of type $NetworkLayout$. Executing the RSL term $mk\_table(n)$ results in a value of type $InterlockingTable$

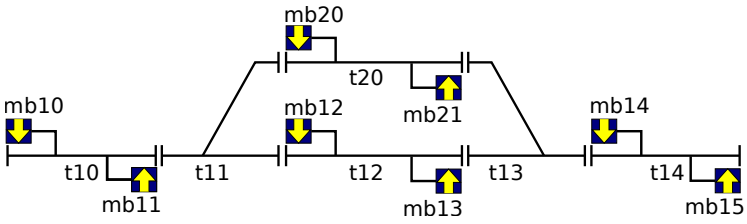

Fig. 3. An example railway network layout

["01a" ↦ mk_Route("mb11", "mb13", ⟨"t11", "t12"⟩, ⟨⟩,
["t13" ↦ MINUS, "t11" ↦ PLUS],
{"mb20", "mb12"}, {},
{"01b", "02a", "02b", "03", "04", "05a", "05b", "06b", "07"}),
... content skipped ...]

As it can be seen one of the generated routes has id `01a`, goes from `mb11` to `mb13` via two sections `t11,t12`, and has no overlap. It requires point `t11` (on its path) to be in `PLUS` position and point `t13` (outside its path) to be in `MINUS` position (as a protecting point). The route has also `mb20, mb12` as protecting signals, and is in conflict with the routes `01b, 02a, 02b, 03, 04, 05a, 05b, 06b, 07`.

Furthermore, the implementation of the DSL and the ITG front-end integrated in RT-Tester [14] can be used to construct and validate descriptions of networks and interlocking tables. Errors are reported together with suggestions how to fix them, e.g. missing protecting signals, points, or conflicting routes can be suggested to be added to the table. The tool can also generate interlocking tables from network descriptions.

6 Related Work

Applications of formal methods to the railway domain have been investigated by numerous research groups. The ultimate goal is to produce methods for developing railway control systems efficiently while ensuring safety. A general overview of the trends can be found in [5].

DSLs for the railway domain have been proved to be efficient for describing interlocking data for other kinds of interlocking systems [8, 12, 7, 11]. Our work goes along the same lines, but we have special focus on route-based interlocking systems which are compatible with ETCS Level 2, e.g. we include notions such as marker boards and virtual signals instead of physical signals.

Several other research groups [15, 1, 9, 2, 13, 10] have also investigated interlocking systems having interlocking tables as design specifications. They also translate the interlocking tables into execution/design models which are then formally verified to satisfy high-level safety requirements. In some cases [2, 13] that verification step is also used for data validation. However, like [9], we follow a *two-step* approach for verification and validation (V&V). In the first step, the data validation is performed by the static semantics checker (described in this paper) in order to ensure the well-formedness of the model that is generated from the data. In the second step (which is outside the scope of this paper),

high-level safety properties of the model are verified using a bounded model checking approach in combination with inductive reasoning.

A few ITGs have been proposed in previous research, e.g. in [13, 2], but they have not been formally specified as our ITG. These ITGs generate tables having data similar to a subset of our data, also by traversing the network layout. However, the ITG in [2] does not generate any data concerning flank and front protection. In [13], the ITG does not generate the collection of route conflicts and items for flank and front protection, but instead in a second phase (after the table generation) they employ model checking to derive these data which are then added manually. Our ITG is – to our best knowledge – the only ITG that is able provide completely automated generation of protecting points and signals directly from the network layout.

7 Conclusion and Ongoing Work

This paper has presented a formal specification of the abstract syntax and static semantics of a DSL for describing interlocking systems, and it has formalized an ITG. The DSL can be used to construct and validate the configuration data for interlocking systems, hence facilitating the automated development and verification. Experiments show that statically well-formed interlocking tables can be produced by "pressing a button" using the ITG. This is much more efficient than manual construction that would have taken many man hours and might have resulted in an interlocking table that was not statically correct and complete. Also, unlike for manual construction, if the network layout changes, the table can easily be re-generated and re-validated.

We have experienced that it is more efficient to develop RSL specifications of the DSL and ITG and then translate these into C++, than to code directly in C++. The RSL specifications are much more readable, and this ensures that nothing is overlooked in the specification. Additionally, the fact that specifications are executable allows us to test the specifications before translating them.

Extensions need to be added in the future to cover other situations encountered in practice such as complex sections with more than three neighbors. Currently, we are working on the formal specification of the operational semantics of the language in the form of a state transition system and on automated verification of the safety properties.

Acknowledgments. The authors would like to thank Jan Bertelsen from Thales and Ross Edwin Gammon and Nikhil Mohan Pande from Railnet Denmark for helping us with their expertise about Danish interlocking systems and always being helpful when we had questions; Dr.-Ing. Uwe Schulze and Florian Lapschies from University of Bremen for their help with the implementation of the DSL in the RT-Tester tool-chain. The first two authors' research has been funded by the RobustRailS project granted by the Danish Council for Strategic Research. The third author's work has been partially funded by ITEA2 project openETCS under grant agreement 11025.

References

1. Michele Banci, Alessandro Fantechi, and Stefania Gnesi. Some Experiences on Formal Specification of Railway Interlocking Systems Using Statecharts. 2005.
2. Yan Cao, Tianhua Xu, Tao Tang, Haifeng Wang, and Lin Zhao. Automatic Generation and Verification of Interlocking Tables Based on Domain Specific Language for Computer Based Interlocking Systems (DSL-CBI). In *Proceedings of the IEEE International Conference on Computer Science and Automation Engineering (CSAE 2011)*, pages 511 – 515. IEEE, 2011.
3. ERTMS. Annex a for ETCS Baseline 3 and GSM-R Baseline 0, April 2012.
4. European Committee for Electrotechnical Standardization. *EN 50128:2011 – Railway Applications – Communications, Signalling and Processing Systems – Software for Railway Control and Protection Systems.* CENELEC, Brussels, 2011.
5. Alessandro Fantechi. Twenty-Five Years of Formal Methods and Railways: What Next? In Steve Counsell and Manuel Núñez, editors, *Software Engineering and Formal Methods*, volume 8368 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 2014.
6. RAISE Language Group. *The RAISE Specification Language.* The BCS Practitioners Series. Prentice Hall, 1992.
7. Anne E. Haxthausen. An Automated Generation of Formal Safety Conditions from Railway Interlocking Tables. *International Journal on Software Tools for Technology Transfer (STTT)*, 2013.
8. Anne E. Haxthausen, Nikolaj Christensen, and Rasmus Dyhrberg. From Domain Model to Domain-specific Language for Railway Control Systems. In *Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2004), Braunschweig*, 2004.
9. Anne E. Haxthausen, Jan Peleska, and Sebastian Kinder. A Formal Approach for the Construction and Verification of Railway Control Systems. *Formal Aspects of Computing*, 23(2):191–219, 2011.
10. Philip James, Faron Moller, HoangNga Nguyen, Markus Roggenbach, Steve Schneider, Helen Treharne, Matthew Trumble, and David Williams. Verification of Scheme Plans Using CSP||B. In Steve Counsell and Manuel Núñez, editors, *Software Engineering and Formal Methods*, volume 8368 of *Lecture Notes in Computer Science*, pages 189–204. Springer, 2014.
11. Phillip James and Markus Roggenbach. Encapsulating Formal Methods Within Domain Specific Languages: a Solution for Verifying Railway Scheme Plans. *Mathematics in Computer Science*, pages 1–28, 2014.
12. Kirsten Mewes. *Domain-specific Modelling of Railway Control Systems with Integrated Verification and Validation.* Verlag Dr. Hut, 2010.
13. Ahmad Mirabadi and Mohammad Bemani Yazdi. Automatic Generation and Verification of Railway Interlocking Control Tables Using FSM and NuSMV. *Transportation Problems*, pages 103–110, 2009.
14. Jan Peleska. Industrial-Strength Model-Based Testing - State of the Art and Current Challenges. In Alexander K. Petrenko and Holger Schlingloff, editors, *8th Workshop on Model-Based Testing*, volume 111 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–28. Open Publishing Association, 2013.
15. K. Winter, W. Johnston, P. Robinson, P. Strooper, and L. van den Berg. Tool Support for Checking Railway Interlocking Designs. In *Proceedings of the 10th Australian workshop on Safety Critical Systems and Software - Volume 55*, SCS '05, pages 101–107, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.