This document is a postprint version of a paper presented at RSSRail-2022, Paris 1-2/06/2022. DOI https://doi.org/10.1007/978-3-031-05814-1_11

The 4SECURail Formal Methods Demonstrator

Franco Mazzanti[1][0000−0003−4562−8777] and Dimitri Belli[1][0000−0003−1491−6450]

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Via G. Moruzzi 1, 56124 Pisa, Italy, {franco.mazzanti,dimitri.belli}@isti.cnr.it

Abstract. The need for high-quality standard interfaces is widely recognized as a mandatory step to reduce procurement costs and create safely operating complex railway infrastructures. That is why European initiatives like EULYNX have been set up precisely with the purpose of supporting standard interfaces development. The exploitation of formal methods during the phase of standardization plays an essential role in raising the quality of the generated specifications. 4SECURail is a recent project that aims to precisely show, with a structured evaluation (known as the formal methods demonstrator), how formal methods might help to improve the quality of a specific signalling interface selected as case study. This paper describes the experience gained with the experiment.

Keywords: Formal Verification · Formal Methods · System Requirements · Standard Interfaces · Railway Signalling System.

1 Introduction

The railway infrastructure is constituted by a large, heterogeneous, and distributed system with components that are on board, trackside, centralized, crossing regional and national borders, managed by different authorities, and developed by different providers. Not surprisingly, the current trend is to standardize the requirements of the various system components together with their interfaces (see, e.g., the EULYNX and the ERTMS initiatives[1]). Standardization, indeed, is expected to increase the market competition with the additional benefits of reducing both vendor lock-in effect and long-term life-cycle costs. However, the defined standard interfaces for the various system components must be *precise* and *correct* to produce the desired effects. They must not suffer from ambiguities in their interpretation and must not give rise to compatibility problems. In this respect, the Shift2Rail Joint Undertaking[2] aims to foster research and innovation in the railway sector by promoting the application of rigorous formal verification techniques to the standard interface development process.

Performing a formal analysis of a signalling standard is a very different - in the process adopted, the models generated, the tools used, the results expected - and more difficult task than performing a verification of a specific product design. In the case of the signalling standard, we are likely to have a more generic

[1] https://eulynx.eu, https://www.ertms.net/
[2] https://shift2rail.org/

specification with many parameters and options, and its description is expected to be at a higher level, not forcing any unnecessary implementation detail. This is quite different from the case of a specific product design, where parameters and options can be somewhat constrained and where certain implementation choices can be deemed acceptable. So, while in the case of a specific product we might have the goal of validating the specification, e.g., with respect to its safety and functional requirements, in the case of a generic, abstract signalling standard, our goals cannot go further than a partial formal analysis of its properties, built on the definition of some specific scenarios. In doing that, we might need to abstract some aspects not needed for the verification of the intended properties and possibly make specific implementation choices. This does not mean at all that the partial formal analysis is not useful. In very simple terms, while the use of formal methods within the development process has the goal of ensuring that the final product satisfies the stated requirements, the use of formal methods within the system requirements specification phase has the goal of improving the confidence that the specification itself - usually expressed in natural language - is precisely what needed.

In line with the Shift2Rail philosophy, the 4SECURail project[3] aims to observe the possible approaches, benefits, limits, and costs of introducing formal methods in the system requirements definition process. This is done with the set up of a *structured evaluation* (a.k.a. the demonstrator), consisting in applying state-of-the-art tools and methodologies with the purpose to collect meaningful information and data on *one* of the possible paths that could be followed to associate a system requirements definition (or a standard interface) with a formal base. Notice, however, that it is not a purpose of the project the definition or the proposal of an overall methodology for the analysis of the requirements in the railway sector; the specific choices and the approaches exemplified with the demonstrator are simply those that have been considered the most fitting with respect to our specific case study, to our background, and to the project timelines. The project activity plan involves three steps:

1. Selection of a railway signalling case study and its initial specification expressed in natural language [1].
2. Derivation of semi-formal and formal models from the initial requirements specification and conduction of the formal analysis using all the generated evidence and artifacts to improve the initial specification [2–5].
3. Performing a quantitative analysis of the costs and benefits derived by the introduction of formal methods in the requirements definition process, leveraging the data collected during the demonstrator process. [6].

In this paper, we introduce the first two steps of the above process, focusing on the presentation of the methodological approach followed in our demonstrator activity, without entering into the details of the formal analysis that has been conducted. The approach adopted for the quantitative cost/benefits analysis (partly still in progress) is not the subject of this paper. The rest of the paper is

[3] https://4SECURail.eu (November 2019 - November 2021)

structured as follows: In Section 2, we give details about the case study that has been the object of the experimentation; in Section 3, we present the approach adopted by the demonstrator. In Section 4, we briefly describe some related studies, and in Section 5, we summarize the results of the experience, and we give insights for future research advancements in the field.

2 The 4SECURail Case Study

The transit of a train from an area supervised by a Radio Block Centre (RBC) to an adjacent area supervised by another RBC occurs during the so-called RBC-RBC handover phase and requires the exchange of information between RBCs according to a specific protocol. This exchange of information is supported by the communication layer specified within the UNISIG SUBSET-039 [7] and UNISIG SUBSET-98 [8]. Figure 1 summarizes the overall structure of the UNISIG standards supporting the handover of a train.

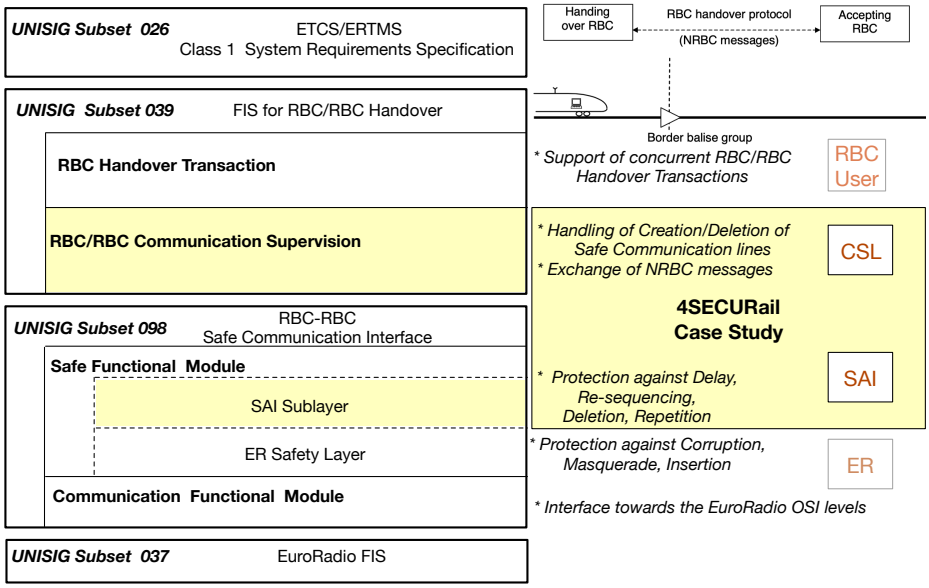


Fig. 1. Overall structure of the 4SECURail case study

The 4SECURail case study focuses on two main sub-components of the communication layers, supporting the RBC-RBC handover communications. The considered components are the Communication Supervision Layer (CSL) of the SUBSET-039 and the Safe Application Intermediate Sub-Layer (SAI) of the SUBSET-098. These two components are the main actors that support the creation/deletion of safe communication lines and protect the transmission of messages exchanged on such lines. In particular, the CSL is responsible for requesting

the activation - and in case of failure, the re-establishment - of the communication line for continuously controlling its liveliness and for forwarding the RBC handover transaction messages on the active line. The SAI is responsible for ensuring the absence of excessive delays, repetitions, losses, or reordering of messages during their transmissions. This is achieved by adding sequence numbers and time-related information to the RBC messages. The two sides of the communication line are configured one as initiator and the other as called.

With respect to the SUBSET-98, the 4SECURail case study does not include, for obvious time and budget constraints, the EuroRadio Safety Layer (ER), which is responsible for preventing corruption, masquerading and insertion issues during the communications, nor the lower Communication Functional Module (CFM) interface. With respect to the SUBSET-039, the 4SECURail case study does not include the description of the activation of multiple, concurrent RBC-RBC handover transactions when trains move from a zone supervised by an RBC to another one. From the point of view of the CSL, the RBC messages are forwarded to/from the other RBC side without the knowledge of the specific content or session to which they belong. The official initial requirements specification document describing the case study and the rationale for its choice is publicly available as project Deliverable D2.3 [1]

3 The Requirements Analysis Process

The formal analysis of the natural language system specification that describes the case study passes through an intermediate step consisting in designing SysML models of the various components. The choice of introducing this intermediate step is motivated by two main reasons. Firstly, the semi-formal modelling of system components is in line with the current trend adopted by the EULYNX initiative, which has selected SysML as accompanying semi-formal notation. And secondly, it is felt natural for a signalling standard to be complemented as far as possible by widely known graphical notations. However, the latter may be a source of troubles, mainly because SysML/UML, despite all the current attempts [9–14], still lacks a recognized, clear, and rigorous semantics. To overcome this problem, we have opted to use an extremely simple subset of the SysML instructions, whose semantics is considered stable and well-defined. The subset used is not the largest subset with the necessary characteristics, but it is just the smallest subset needed to model our case study. Extensions to this subset are definitely possible, but more investigations are needed, and this issue is out of our project goals.

In the modelling and analysis of the case study, a few choices have been made. In particular, the requirements of the SAI component allow two alternative options in modelling the safe connection initialization phase: One option is based on the "Triple Time Stamping (TTS)" approach, while the other is based on the "Execution Cycle (EC) Defence Technique" approach. Our modelling takes into account the EC option which, at a first glance, seemed less dependent by real-time aspects.

The overall approach followed during the modelling and analysis process is incremental and iterative. About 53 versions of the system have been generated, each one widening the set of requirements of the case study modelled, and each one passing through the steps of semi-formal and formal modelling and analysis. During this iterative process, four kinds of artefacts have been generated and kept aligned:

1. A more abstract, semi-formal UML state machine design of the components under analysis.
2. A more detailed executable version of the same UML state machines.
3. A set of formal models derived from the executable UML state machine.
4. A natural language rewriting of the requirements based on the designed and analysed models.

Figure 2 depicts the relationship between these artefacts, whose detailed description is given in the following subsections. The activity of generating and elaborating most of the shown artefacts (currently) requires a human problem understanding and solving activity. The only part that can be mechanically automated (partly achieved within the project) is the generation of the formal models starting from the UML executable models.

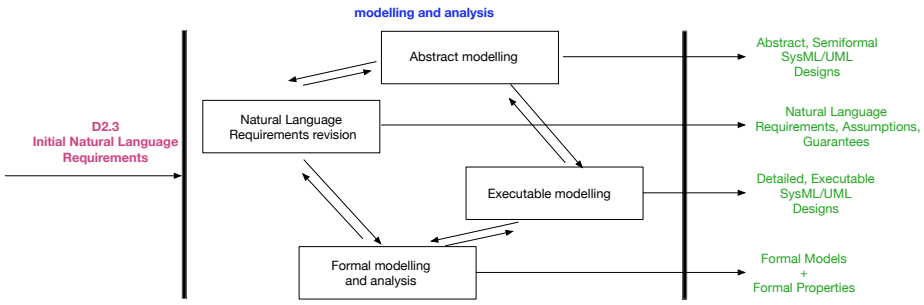


Fig. 2. The 4SECURail demonstrator generated artefacts

3.1 Semi-formal Designs

The first step in trying to associate an operational model to our input requirements specification consists in drawing an abstract design of the state machine describing the various components, putting the accent of the control flow relation between the most relevant system states, the events that trigger the corresponding state transitions, and the communication events occurring among such components. Figure 3 shows an example of such abstract/semi-formal design. The corresponding designs of the two sides of the modelled CSL and SAI components can be found in Appendix B of Deliverable 2.5 [4]

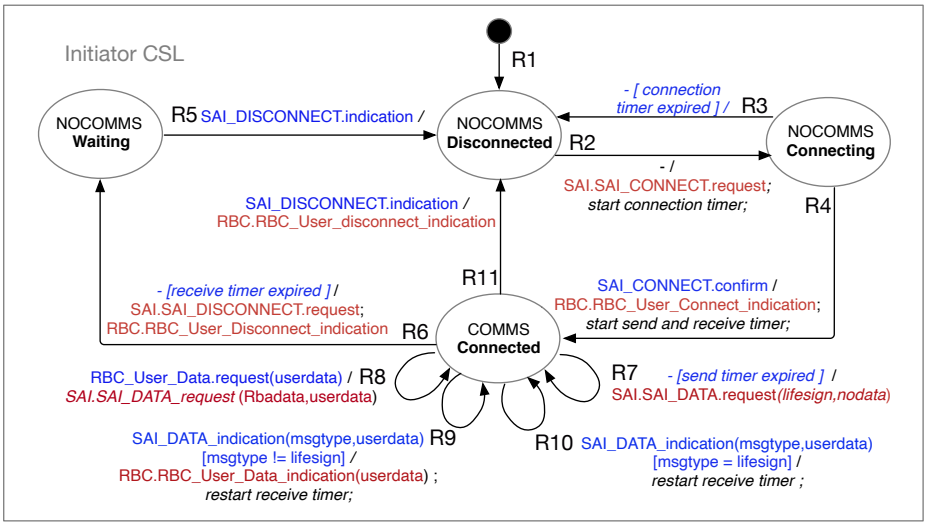


Fig. 3. The Initiatr CSL (ICSL) abstract design

We can observe that no details are given at this step on how some abstract feature is implemented (let us consider, for example, the case of timers or the specific calculations being performed as the effect of a transition). These kinds of designs, however, are already useful as a reference and base documentation for the revision (or confirmation) of the overall structure of the natural language requirements describing the various system components. This initial step has already allowed us to clarify duplications and ambiguities in the initial requirements document. Appendix B of Deliverable 2.2 [3] of the 4SECURail project shows some of the annotations made to the initial requirements in the early stages of the design. As the modelling process evolves and becomes more formal this kind of design is updated to continue reflecting the actual structure of the system.

3.2 Executable UML Designs

The next step towards a formal model is the completion of the abstract design by providing an implementation of all the informally specified aspects. This means to precisely define all the needed local variables of the various components and clearly describe how they are manipulated within the effects of the various transitions. This also means providing a way to model a reasonable temporal flow since the overall system behaviour depends on several time-dependent aspects. Moreover, in order to generate a closed executable system, it is necessary to build parts of the environment capable of receiving data from our modelled components and stimulate them with appropriate events. In our specific case, we need three kinds of environment components: two components modelling the possible behaviour of the RBC users, and a component modelling the ER

that allows the two SAI components to communicate. We also added a Timer component that allows all the components to proceed still in an asynchronous way, but relatively at the same speed. Figure 4 shows the resulting structure of the whole system. All the added environment and timer components can be designed in UML to facilitate the system encoding into the selected formal notations.

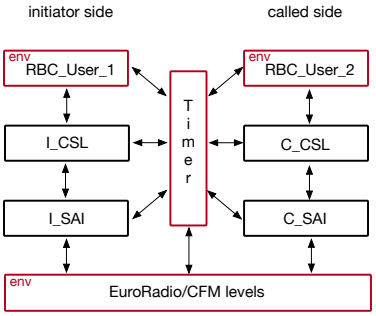


Fig. 4. The complete executable system structure

Figure 5 shows an example of executable state UML design corresponding to the abstract version of the component shown in Figure 3.

3.3 Formal Modelling

The desirable approach for passing from a SysML/UML executable design (possibly generated with commercial tools like PTC, Yakindu, Rhapsody, Cameo Modeling Tool (once Magic Draw), SPARX-EA, Papyrus) to a set of formal models is to use available translation tools. During the initial phases of the project, we experimented with the SPARX-EA tool for the design of the executable SysML models. Still, no translation tool was found to be available, and an effort to build it was beyond the project effort and outside the project goals. Moreover, linking such translation tools to a specific commercial SysML design tool was considered not desirable. Our solution has been to make a first manual translation of the executable SysML design into the design notation accepted by the UMC tool of the in-house developed KandISTI [15] framework. The UMC notation for specifying a collection of interacting state machines is, in fact, a simple textual, user-friendly encoding of the state machines that allows an almost direct translation of the case study with minimal effort. A fragment of the UMC notation for the state machine depicted in Figure 5 is shown in Figure 6.

UMC allows to explore the possible system evolutions and verify branching time properties on it. This framework has been chosen as first target because it fits well the need for fast design prototyping. The resulting graph describing the evolutions of the system can be analysed or saved in the form of Labelled

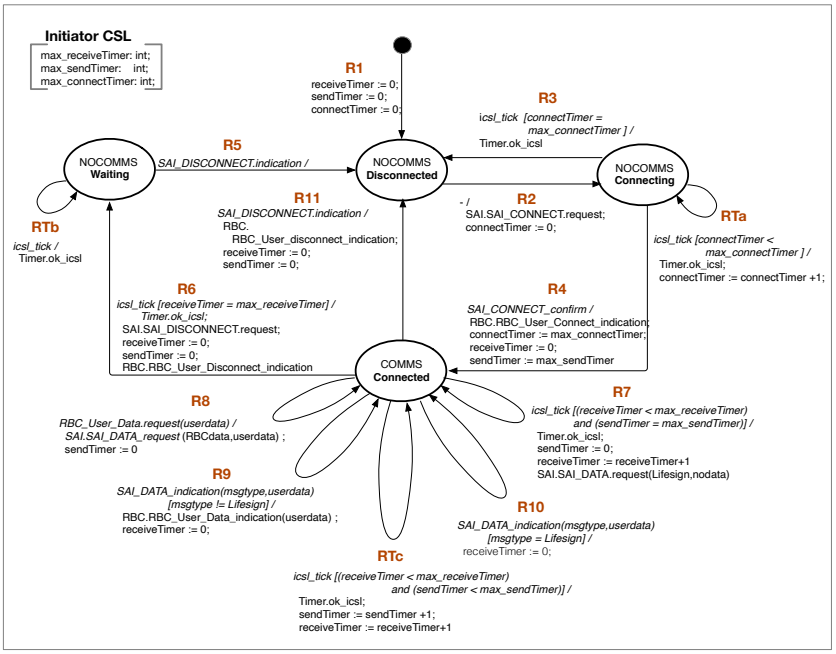


Fig. 5. The Initiator CSL executable model

Transition System (LTS), where the user has the choice to specify which kind of information should be associated with the LTS edges. This information may include the UMC transition label, the outgoing events generated by the effects of a transition, or any other custom flag associated with the firing of the transition. However, UMC is essentially an academic prototype used mainly for research and teaching purposes. Therefore, we wanted to take into account also furthermore industry-ready formal verification frameworks.

The second framework that has been chosen to support the formal analysis of the system is ProB [16]. Indeed, according to several surveys (see, e.g., [17–19]) B/EventB appears to be one of the most adopted formal methods in railways. Moreover, ProB has a very user-friendly interface requiring a small effort to be learnt and powerful verification methods. Last but not least, it is freely available as an open-source product.

A third framework that has been taken into account is the CADP toolbox with its LNT language [20, 21]. One interesting aspect of this third approach is that the mathematical representation used for the models is based on process algebras, and can exploit the rich theory around LTS for supporting the verification process (e.g., minimizations, bisimulations, and compositional verification [22–24]). Another interesting aspect of the CADP framework is that the model structure stands on events, and in particular on communication actions. The logic used to reason on these models is a very powerful, action-based,

```
Class I_CSL is
Signals
    -- from RBC
    IRBC_User_Data_request(arg1: int);

    -- from I_SAI
    ISAI_CONNECT_confirm;
    ISAI_DISCONNECT_indication;
    ISAI_Error_report;
    ISAI_DATA_indication(arg1: Token,
                         arg2: int);

    -- from Timer
    icsl_tick;

Vars
    ------ PORTS
    RBC_User: I_RBC;
    SAI: I_SAI;

    ------ CONFIGURATION PARAMS
    max_receiveTimer: int;
    max_sendTimer:    int;
    max_connectTimer: int;

    ------  LOCAL VARS
    receiveTimer:         int := 0;
    sendTimer: int := 0;
    connectTimer:         int := 0;
```

```
Behaviour
R1_ICSL:
initial -> Disconnected

R2_ICSL_connecting:
    Disconnected -> Connecting
        {- /
        SAI.ISAI_CONNECT_request;
        connectTimer := 0;}

RTa_ICSL_okicsl_incr:
    Connecting -> Connecting
        {icsl_tick [connectTimer <
                    max_connectTimer ] /
     Timer.ok_icsl;
     connectTimer := connectTimer +1}

R3_ICSL_okicsl_connect:
    Connecting-> Disconnected
        { icsl_tick [connectTimer =/
                    max_connectTimer ] /
        Timer.ok_icsl}

R4_ICSL_userconnind:
    Connecting -> COMMS
        { ISAI_CONNECT_confirm /
        RBC_User.IRBC_User_Connect_indication;
        connectTimer := max_connectTimer;
        receiveTimer := 0;
        sendTimer := max_sendTimer}
...
end I_CSL;
```

Fig. 6. The ICSL encoding in UMC (fragment)

branching-time logic. This creates another point of view from the one supported by ProB, which is more state-oriented. Similarly to ProB, CADP is freely usable with an academic licence.

ProB	UMC	LNT
• Static Analysis • Reachability Properties • Statespace Projections • Statespace Stats • State Invariants • Deadlocks • Trace Explanations as Message Sequence Diagrams • CTLe / LTLe Model Checking (state/event based) • ...	• Static Analysis • Reachability Properties • System Traces Minimization • Statespace Stats • Deadlocks • Runtime Errors • Custom system observations • Trace Explanations as Message Sequence Diagrams • UCTL Model Checking (state/event based) • ...	• Static Analysis • Reachability Properties • Statespace Stats • Deadlocks • MCL Model Checking (event based) • Compositional Verification • Strong/ Divbranching/ Sharp Minimazations • Powerful scripting language • ...

Fig. 7. Table of verification features

Once available the UMC encoding of our model, we can exploit two other in-house translators to directly translate the UMC model into the ProB and LNT notations. We omit here the details of the translations, for which we refer to Appendix A of 4SECURail Deliverable D2.5 [4] and to [27].

The size of a complete (closed) executable model clearly depends on the complexity of the environment components used to stimulate our communication layer. In one of the simplest scenarios, the UMC executable model consists of about 2500 lines, resulting in a ProB model of about 3500 lines and in a LNT model of about 4000 lines. The modelling and analysis of the case study within the project have required an effort of about seven person-months.

These methods and tools are not meant to be, in general, "the best ones" or the "most fitting" the railway sector. Our selected frameworks are just those "most fitting" the project's expected efforts and goals. Alternative meaningful choices, similar in style, might have been mCRL2, nuXmv, Spin, TLA+, HLL.

The choice to model and analyse the system with more than one framework is considered very important for two reasons. Firstly, it allows to take advantage of the multiple verification methods provided by the different frameworks, e.g., analysis of state invariants with ProB, system and components property-driven minimizations with CADP, reachability explanations provided in the form of sequence diagrams with UMC (Figure 7 shows a table of some of the features provided by our three frameworks). And secondly, the choice of using different formal notations allows us to verify the correctness of the mechanical translation from UML executable design (in UMC) into the other formal notations. All the three formal versions of the system can indeed be *proven* to reflect precisely the same system[4].

In Section 5.4 and 5.5, Appendix E and F of Deliverable D2.5 [4] are shown the various way in which all these frameworks have been used to analyse the system behaviour. Our experimentation shows that the selected formal frameworks can be used either in a "lightweight" or "advanced" way. In Figure 7, the verification features that can be easily exploited without any advanced prior knowledge, and in an almost "push button" way, are those appearing in black. For example, with ProB, by just selecting the "Model Check" button (see Figure 8), it is possible to analyse the full state-space for deadlocks, invariant violations, and other errors. Other features, typically those requiring the encoding of properties in temporal logic formulas, may require a prior non-trivial background on formal methods and model checking.

While the previous step of designing the UML executable models already helped to identify and remove ambiguities and unclarities, the static analysis and the model checking of the formal models have been essential to detect missing requirements leading to loss events, missing assumptions leading to deadlocks, and implementation mistakes leading to properties violation expected to be guaranteed.

[4] This has been done by comparing the formal semantics (in the form of an LTS) of the three versions of the system and mechanically proving that they are strongly equivalent

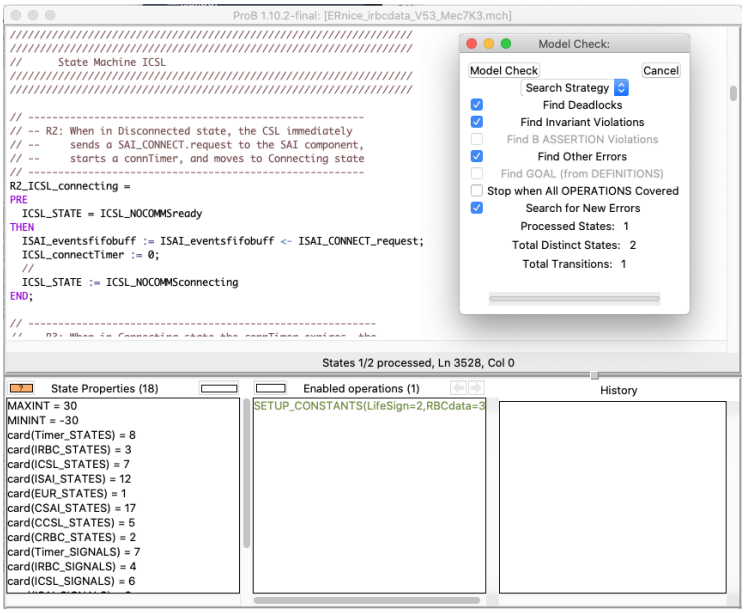


Fig. 8. Model Check GUI of ProB

3.4 Revised Natural Language Requirements

Pragmatically, we are afraid that a system requirements specification of a standard interface is doomed to have an official natural language description as well. One of the goals of the 4SECURail demonstrator process is to show a way to improve such an initial natural language specification by backing it with formal models. This improvement has two goals:

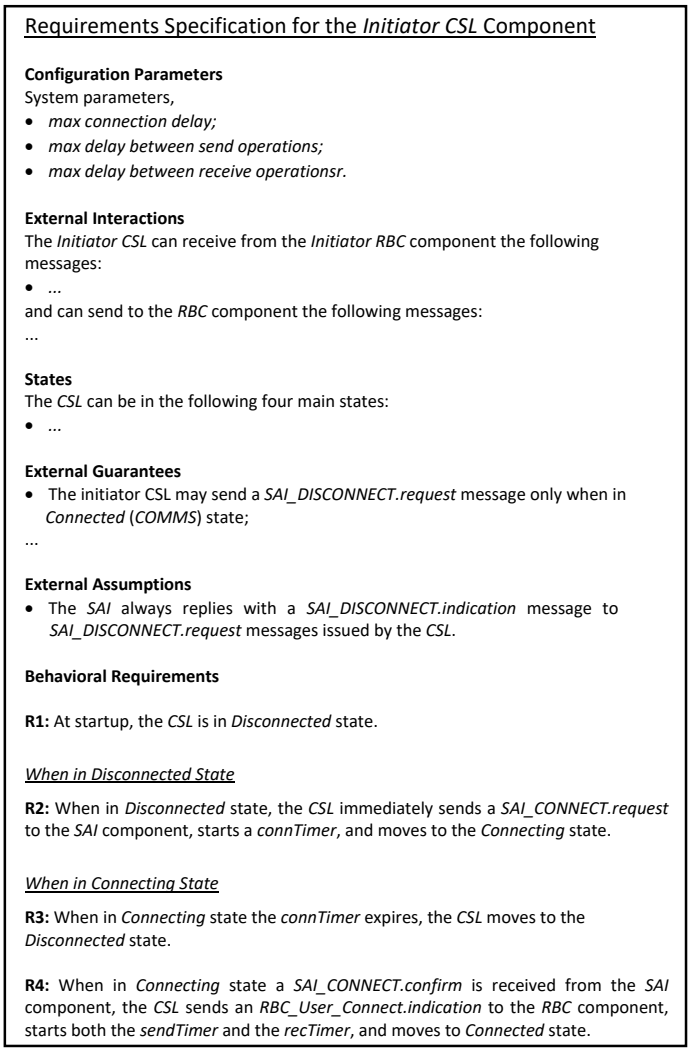
1. Guarantee that the specification is based on a rigorous, clear structure, removing ambiguities and duplications.
2. Improve the confidence that the specification is correct, i.e., capable of interoperating with other systems, with neither missing nor inconsistent requirements.

The generation of executable, formal models is the mean to achieve these goals, not the goal itself. Therefore, in our demonstrator process, we also tried to show a possible way of writing the requirements specification in a manner strictly tied to the executable, formal models but still in natural language.

The implementation choices that have been made in the construction of the executable, formal models should not appear in the natural language requirements specification, which is supposed to be at a higher level than an executable implementation. The abstract semi-formal design of a system, like the one shown in Figure 3, appears to be at the correct level of abstraction for this task.

Figure 9 shows a possible example of a rigorous natural language description of the system resulting from the aligned generations of the various artefacts

produced during the process. It is worth noticing the strict relation between the requirements describing the system behaviour, the semi-formal design, the executable design, and the formal models.



Requirements Specification for the *Initiator CSL* Component

Configuration Parameters
System parameters,
- *max connection delay;*
- *max delay between send operations;*
- *max delay between receive operationsr.*

External Interactions
The *Initiator CSL* can receive from the *Initiator RBC* component the following messages:
- ...

and can send to the *RBC* component the following messages:
...

States
The *CSL* can be in the following four main states:
- ...

External Guarantees
- The initiator CSL may send a *SAI_DISCONNECT.request* message only when in *Connected* (*COMMS*) state;

...

External Assumptions
- The *SAI* always replies with a *SAI_DISCONNECT.indication* message to *SAI_DISCONNECT.request* messages issued by the *CSL*.

Behavioral Requirements

R1: At startup, the *CSL* is in *Disconnected* state.

When in Disconnected State

R2: When in *Disconnected* state, the *CSL* immediately sends a *SAI_CONNECT.request* to the *SAI* component, starts a *connTimer*, and moves to the *Connecting* state.

When in Connecting State

R3: When in *Connecting* state the *connTimer* expires, the *CSL* moves to the *Disconnected* state.

R4: When in *Connecting* state a *SAI_CONNECT.confirm* is received from the *SAI* component, the *CSL* sends an *RBC_User_Connect.indication* to the *RBC* component, starts both the *sendTimer* and the *recTimer*, and moves to *Connected* state.

Fig. 9. Natural Language requirements for ICSL

At this level, an important role is played by the "guarantees" that each component should ensure to the other components making use of it, and the "assumptions" on the external environment which are supposed to hold. An example for all: When a connection request is sent from the initiator SAI to the

ER, we should assume that it will always have a reply from ER either through a connection-confirmation or a disconnect-indication. The formal analysis of the system, indeed, allows to check if such an assumption is not satisfied by the ER level, and deadlocks may appear in the behaviour of the SAI component.

4 Related Works

The analysis of still "unstable" requirements has been widely investigated by Heitmeyer [25, 26] with the Software Cost Reduction (SCR) tabular notation and method. While Avnur [28], differently, has based its analysis on Finite State Machines. In [29, 30], Giannakopoulou, Mavridou et al. have exploited the FRET requirements elicitation tool for analysing requirements and generating Simulink models. In [31], Lutz and Ampo have used the Paradigm Plus tool to model the requirements and verify them with PVS, while Ferrari et al. [32] have used Simulink for modelling and verification of the system requirements. Another quite related effort is that one in [33], where Basile et al. have modelled and analysed part of the UNISIG SUBSET 98 using Uppaal.

Many more works have been done when starting from UML/SysML designs instead than from informal requirements. In [34], e.g., Caltais et al. have discussed the transformation of SysML models into NuSMV. While in [35], Snook and Butler have discussed the translation into the B notation of designs in the UML-B profile. Several other studies (e.g., [36]) instead describe the translation of UML/SysML models in mCRL2. Still, the approach which is the most similar to ours is the one by Bouwman et al. [37], which has the same goal of enriching EULYNX interfaces with formal models, that in this case, are encoded in mCRL2.

5 Conclusions

It is true that sometimes standard tends to prescribe vague goals and prohibitions, that they tend to be continuously revised to fix their weaknesses, and that implementations have often no strong legal incentive to fully comply with them. Our effort should be considered as a contribution towards the definition of clear, rigorous, stable, strongly enforced signalling standard, as required in the railway domain and as promoted by the Eulynx [5] and RCA [6] initiatives.

The executable and formal models generated during the analysis of the standard have the main purpose to provide the standard designer with some feedback from the analysis of some instantiations of the standardised interface. Surely this is not sufficient to guarantee the generic correctness of the standard for all the allowed variation points, but it still much better than relying exclusively on a plain natural language description of the standardised interface for which no executable model has ever been devised and analysed.

[5] https://www.eulynx.eu/index.php
[6] https://public.3.basecamp.com/p/jGh4E3ZdE8T1RtoxvbWLCYss

From the point of view of the provider the generated executable UML models might be useful to shed some light on some aspects that might still be considered ambiguous in the natural language description of the standard, and to suggest the structure of a feasible implementation possibly reducing the design and debugging effort of the proprietary implementation.

The goal of the 4SECURail demonstrator has been the illustration - with a real experiment - of a possible way in which formal methods, in particular, can be exploited to improve the quality of system requirement specifications. The use of formal models is indeed considered important for the analysis of the interactions inside complex systems of systems, like those typical of the railway sector.

We have shown how creating an easy-to-understand and communicate executable model is an intermediate step that already allows to detect several possible weaknesses in the initial natural language requirements. However, this step is also a passage where errors can easily be made, and a formal analysis of executable models becomes important to detect and remove them. This can be done with a "lightweight" use of formal methods, since it does not require particular advanced background and experience. More advanced properties of the system, e.g., those related to the expected interoperability properties the system should guarantee, may require a more advanced knowledge of the formal frameworks and, therefore, higher costs in terms of effort and learning curve.

Adopting a formal methods diversity approach to analyse an executable model adds the advantage of having an alternative way to verify the correctness of the generated formal models and allows to exploit a broader range of verification features. The experience gained with our experimentation allowed us to confirm the essential importance of relying on an automatic/mechanical translation of executable models into the formal notations used for formal analysis. In their absence, we would not have been able to generate 53 releases of formal design in three different notations. The experimentation conducted within the 4SECURail project has put in evidence many aspects that deserve deeper studies. Among these:

- The precise role of SysML/UML as system design notation.
- The way to support the transition from executable designs generated in industry-ready Model-Based System Engineering frameworks to formal models.
- The way to support lightweight use of formal methods to make them more easily adaptable to the existing requirements definition processes.
- The way in which the formal models and the verified properties can be explained back in a rigorous natural language style.

Another piece of work that is still missing and that we hope to be able to complete in the near future is a thorough evaluation of the experimented approach and of its positioning with respect to the state of art. The project deliverables, the initial and revised case-study requirements, the UML designs, the formal models, the (open source) translation tools are all publicly available in the 4SECURail site and in open access repositories [38, 39].

Acknowledgements This work has been partially funded by the 4SECURail project. The 4SECURail project received funding from the Shift2Rail Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement No 881775 in the context of the open call S2R-OC-IP2-01-2019, part of the "Annual Work Plan and Budget 2019", of the programme H2020-S2RJU-2019. The content of this paper reflects only the authors' view and the Shift2Rail Joint Undertaking is not responsible for any use that may be made of the included information. We are grateful to the colleagues of the Work Stream 1 of project 4SECURail, and in particular to Alessandro Fantechi, Stefania Gnesi, Davide Basile, Alessio Ferrari and Maurice ter Beek, for the comments and suggestions during the project.

References

1. Piattino, A.: 4SECURail deliverable D2.3 "Case study requirements and specification". November 2020. In "The 4SECURail Work Stream 1 Deliverables" https://doi.org/10.5281/zenodo.5807738
2. Mazzanti, F., Basile, D.: 4SECURail deliverable D2.1 "Specification of formal development demonstrator". November 2020. In "The 4SECURail Work Stream 1 Deliverables" https://doi.org/10.5281/zenodo.5807738
3. Mazzanti, F., Basile, D.: 4SECURail deliverable D2.2 "Formal development Demonstrator prototype, first release". November 2020. In "The 4SECURail Work Stream 1 Deliverables" https://doi.org/10.5281/zenodo.5807738
4. Mazzanti,F., Belli, D. .: 4SECURail deliverable D2.5 "Formal development demonstrator prototype, final release '. July 2021. In "The 4SECURail Work Stream 1 Deliverables" https://doi.org/10.5281/zenodo.5807738
5. Basile, D. et al.: Designing a Demonstrator of Formal Methods for Railways Infrastructure Managers LNCS, vol.12478, Springer, 2020
6. Vaghi, C.: 4SECURail Deliverable D2.6 "Specification of Cost-Benefit Analysis and learning curves, Final release" In "The 4SECURail Work Stream 1 Deliverables" https://doi.org/10.5281/zenodo.5807738
7. UNISIG: SUBSET-039, FIS for the RBC/RBC Handover, 17-12-2015 (Issue 3.2.0)
8. UNISIG: SUBSET-098, RBC/RBC Safe Communication Interface, 21-05-2007
9. OMG: Unified Modelling Language version 2.5.1, December 2015.
10. OMG: SysML 1.6 Specification, November 2019.
11. OMG: Precise Semantics of UML State Machine version 1.0, May 2019.
12. OMG: Action Language for Foundational UML (Alf), version 1.1, July 2017.
13. OMG: Semantics of a Foundational Subset for Executable UML Models (fUML), Version 1.5 May 2020.
14. OMG: Precise Semantics of UML Composite Structure (PSCS), Version 1.2.
15. ter Beek, M. H., Fantechi, A., Gnesi, S., Mazzanti, F.: States and Events in KandISTI. In Models, Mindsets, Meta: The What, the How, and the Why Not?, vol. 11200, pp. 110-128, Springer, Cham (2019)
16. Leuschel, M., Butler, M: ProB: An Automated Analysis Toolset for the B Method, In Software Tools for Technology Transfer (STTT), 10, 2, Springer, 185–203, 2008.
17. ter Beek, M. H. et al.: Adopting formal methods in an industrial setting: the railways case. In International Symposium on Formal Methods, volume 11800, pp. 762—772. Springer, Cham. (2019)
18. Ferrari, A. et al.: Comparing formal tools for system design: a judgment study, IEEE Int. Conf. on Software Engineering (ICSE), June 2020

19. Ferrari A., et al.: Systematic Evaluation and Usability Analysis of Formal Methods Tools for Railway Signaling System Design, IEEE Transactions on Software Engineering, November 2021.
20. Champelovier, D. et al.: Reference Manual of the LNT to LOTOS Translator, https://cadp.inria.fr/publications/Champelovier-Clerc-Garavel-et-al-10.html
21. Garavel, H., Lang, F., Mateescu, R.,and Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes, International Journal on Software Tools for Technology Transfer (STTT), 15(2):89-107, April 2013
22. Garavel, H., Lang, F. and Mateescu, R.: Compositional Verification of Asynchronous Concurrent Systems using CADP Acta Informatica vol 52 Num 4-5 April 2015, pag. 337–392, 2015.
23. Lang, F. and Mateescu, R.:and Mazzanti,F.: Sharp congruences Adequate with Temporal Logics Combining Weak and Strong Modalities, in TACAS 2020, LNCS, vol. 12079, Springer, 2020.
24. Lang, F. and Mateescu, R.:and Mazzanti,F.: Compositional verification of concurrent systems by combining bisimulations, Formal Methods in System Design (2021)
25. Ramesh Bharadwaj, R., and Heitmeyer,C.L.: Model Checking Complete Requirements Specifications Using Abstraction, in Automated Software Engineering, vol.6 n.1 1999, Springer.
26. Heitmeyer, C.L.: Formal Methods for Specifying, Validating, and Verifying Requirements, Journal of Universal Computer Science, vol. 13, no. 5, 2007
27. Mazzanti, F., Belli, D.: Formal Modelling and Initial Analysis of the 4SECURail Case Study, in Proceedings of 5th Workshop on Models for Formal Analysis of Real Systems, MARS 2022, EPTCS - to appear.
28. Avnur, A: A finite state machine model for requirements engineering, IREB Requirements Engineering Magazine (2015-03); https://re-magazine.ireb.org/articles/a-finite-state-machine-model
29. Anastasia Mavridou, A. et al.:Bridging the Gap Between Requirements and Simulink Model Analysis, REFSQ-2020, Pisa, Italy, March 24, 2020.
30. Giannakopoulou D. et al: Formal Requirements Elicitation with FRET, in Joint Proceedings of REFSQ-2020 Workshops, Pisa, Italy, March 24, 2020.
31. Lutz, R.R. and Ampo Y.: Experience Report: Using Formal Methods for Requirements Analsis of Critical Spacecraft Software.
32. Ferrari, A. et al.et al.: The Metrô Rio case study, Science of Computer Programming, Vol. 78, Isue 7, 2013.
33. Basile, D., Fantechi, A., Rosardi, I.: Formal Analysis of the UNISIG Safety Application Intermediate Sub-Layer. FMICS 2021, LNCS, vol 12863, Springer.
34. Caltais, G. et al.: SysML to NuSMV Model Transformation via Object-Orientation, in LNCS 10107, Springer 2016.
35. Snook, C. and Butler, M.: "UML-B and Event-B: an integration of languages and tools", in The IASTED International Conference on Software Engineering - SE2008, Innsbruck, Austria. 12 - 14 Feb 2008.
36. Hansen.H. et al.: Automated Verification of Executable UML Models, Springer LNCS 6957, 2010.
37. Bouwman M. et al.: "What is the Point: Formal Analysis and Test Generation for a Railway Standard" in Proceedings of the 29th European Safety and Reliability Conference (ESREL), 2020.
38. The 4SECURAil project. https://4securail.eu, https://doi.org/10.5281/zenodo.5807738
39. Mazzanti, F., Belli, D.: Supplementatry material of 4SECURail Workstream 1. https://doi.org/10.5281/zenodo.4280773,