D4.2 – Preliminary Trial Report

Deliverable ID	**D4.2**
Deliverable Title	**Preliminary Trial Report**
Work Package	**WP4**
Dissemination Level	**PUBLIC**
Version	**1.1**
Date	**2018-11-27**
Status	**Final Version for Review**
Lead Editor	**CNR**
Main Contributors	**CNR, SIRTI**

Published by the ASTRAIL Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 777561
Call identifier: H2020-S2RJU-2017 | Topic: S2R-OC-IP2-01-2017 – Operational conditions of the signalling and automation systems; signalling system hazard analysis and GNSS SIS characterization along with Formal Method application in railway field

Document History

Version	Date	Author(s)	Description
0.1	2018-09-12	CNR	TOC
0.2	2018-11-06	CNR	First Draft with Introduction and Tool Usability Assessment
1.0	2018-11-20	CNR, SIRTI	Final Draft for Internal Review
1.1	2018-11-27	ISMB, CNR	Version with corrections from ISMB

1 Legal Notice

Table of Contents

1 Introduction

1.1 Purpose and Scope

Formal methods are mathematically-based techniques to support the development of software intensive systems [CLA96] [BOC09]. Normally, formal methods oriented to design and verification of systems include (i) a modelling language, which is used to model a system, and (ii) a verification strategy, which is used to verify properties on the system. Formal methods are usually associated to formal tools, which can provide textual or visual editors to create a model of the system, as well as automated verification capabilities. Formal methods have been largely applied in industrial projects, especially in the safety-critical industry, including railways [BBF18]. However, it cannot yet be said that a single mature technology has emerged.

This **Work Package 4 (WP4)** of the ASTRail project aims to identify, based on an analysis of the state-of-the-art and on concrete trials, the candidate set of formal and semi-formal methods that appear as the most adequate to be used in the railway context. In the following, when we will use the general term “formal method”, we will implicitly include also semi-formal methods, i.e., those methods that use languages for which the semantics is not formally defined but depends on their execution engine. Since formal methods are normally associated with tools, we will also use formal methods and formal tools interchangeably.

To address the goal of identifying the most adequate formal methods, WP4 is structured into four tasks (T4.3, in **bold**, is the focus on the current deliverable):

- Task 4.1 Benchmarking: this task aims at studying the state-of-the-art and state of the practice of formal and semi-formal methods, by gathering knowledge from the literature and railway practitioners.
- Task 4.2 Ranking: this task aims at providing a ranking matrix to support the selection of the most adequate formal methods to be used in a certain development context.
- **Task 4.3 Trial Application:** this task aims at experimenting the usage of a set of selected formal methods through the modelling of the moving-block system, from Task 2.1.
- Task 4.4 Validation: this task aims at validating the usage of the selected formal methods by integrating the moving-block model with the automated driving technologies from Task 3.3.

The current deliverable, **D4.2 Preliminary Trial Report,** reports on Task 4.3 Trial Application, while the results of Task 4.1 and 4.2 have been reported in D4.1.

1.2 Executive Summary

The description of Task 4.3 Trial Application, as reported in the ASTRail proposal, is as follows:

*“This task will address the **modelling of the moving-block** model from Task 2.1, by means of selected languages and tools making also some experimental parallel application of multiple techniques/tools in order to evaluate and **compare their usability** and applicability in the domain. The modelling activity will be complemented by a **requirements quality analysis and consolidation** phase, in which the system requirements will be automatically evaluated for non-ambiguity and clarity with NLP techniques, and will be updated/consolidated based on potential variations triggered during the modelling phase.”*

This deliverable follows the task description, as it was planned in the proposal. Figure 1 provides an overview of the followed approach. We first selected a set of 8 formal tools based on the results of Task 4.1 and Task 4.2. The tools are Simulink, SCADE, UPPAAL, NuSMV, SPIN, ProB, Atelier B, UMC. Through these tools, we **modelled the moving-block system** according to requirements derived from Task 2.1. This modelling activity provided insight on the actual capabilities of the different tools, and highlighted that each technique is applicable for certain purposes and certain contexts.

Some tools, such as Simulink and SCADE, are more appropriate if one aims at creating prototypes that can be simulated and wishes to generate code from the models. Other techniques, such as UML and associated tools, are more appropriate if one wishes to provide a high-level view of the system's architecture, and aims to communicate with stakeholders with different backgrounds. Tools such as SPIN or NuSMV, are more oriented to brute-force formal verification of large systems. Statistical model-checkers, such as UPPAAL, are appropriate when one wishes to verify real-time and probabilistic aspects of a system. Finally, tools based on the refinement paradigm, such as ProB and Atelier B, are more oriented towards top-down development of single systems rather than composition of systems.

The models produced with the 8 formal techniques were used to provide a **usability assessment** of the tools, which was performed by means of a showcase involving industrial railway stakeholders. The usability assessment showed that the stakeholders considered Simulink and SCADE as the most usable tools, among those presented, indicating a preference for tools that have graphical modelling languages and support powerful graphical simulations.

After this activity, we performed a **requirements consolidation and refinement** phase, in which the initial requirements of the moving-block model were refined through multiple iterations, and were automatically analysed by means of NLP techniques. After this activity, a final set of requirements for the moving-block system was produced.

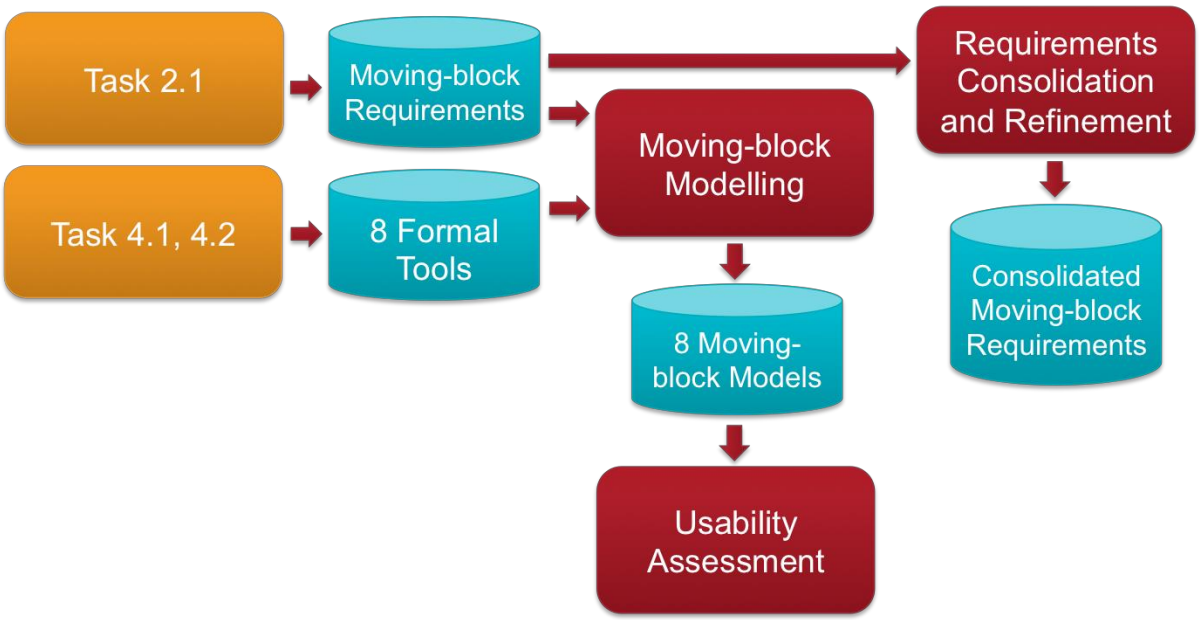

Figure 1 Overview of the deliverable content and results

The remainder of this deliverable is structured as follows:

- In Sect. 2, we report the activity of modelling the moving-block system with the selected tools;
- In Sect. 3, we characterise the different tools based on the contexts in which their usage is more appropriate, according to the experience gained through the modelling activity;
- In Sect. 4, we report the results of the usability assessment activity;
- In Sect. 5, we report the results of moving-block requirements consolidation and refinement.
- In Sect. 6, we provide conclusion and final remarks.

1.1 Related documents

ID	Title	Reference	Version	Date
[RD.1]	D4.1 Report on Analysis and Ranking of Formal methods	D4.1	4.0	31/05/2018
[RD.2]	D2.1 Modelling of the moving block signalling system	D2.1	1.0	29/11/2017
[RD.3]	D2.2 Moving Block signalling system Hazard Analysis	D2.2	1.0	28/02/2018

2 Moving Block Modelling

The goal of this section is to illustrate the approach followed to provide the different models of the moving-block system as described in Deliverable D2.1. In D2.1, the moving-block system was modelled by means of a UML diagram. This diagram was used as information source to define a set of general natural language requirements of the moving-block system (Sect. 2.1). Together with the UML diagram, the requirements were used as a reference to develop eight different models of system by means of different formal tools, selected in accordance to the results of Task 4.1 and 4.2 (Sect. 1.2).

The objective of this modelling activity was not to replicate the exact same design across all the tools, but:

A. to exploit and assess the capabilities of the different tools, and understand which tools are more appropriate for which development context, as described in Sect. 3;
B. to have different models to be showcased for a usability assessment of the tools, as presented in Sect. 4.

2.1 Moving-block Principles and Requirements

This section describes the moving block principles, according to the source model provided in D2.1. Furthermore, it lists the preliminary requirements derived from the source model, and used as a reference to develop the different formal models.

The main moving block principles and components, based on D2.1, are illustrated by Figure 2. We have three components: two on the train, and one wayside system. The train carries the Location Unit (LU) and the Onboard Unit (OBU). The wayside system is the Radio Block Centre (RBC). The location of the train is received through GNSS satellites by the LU. This sends the location to the OBU, which, in turn, sends the location to the RBC. Upon reception of a location from a train, the RBC sends a movement authority to the OBU.

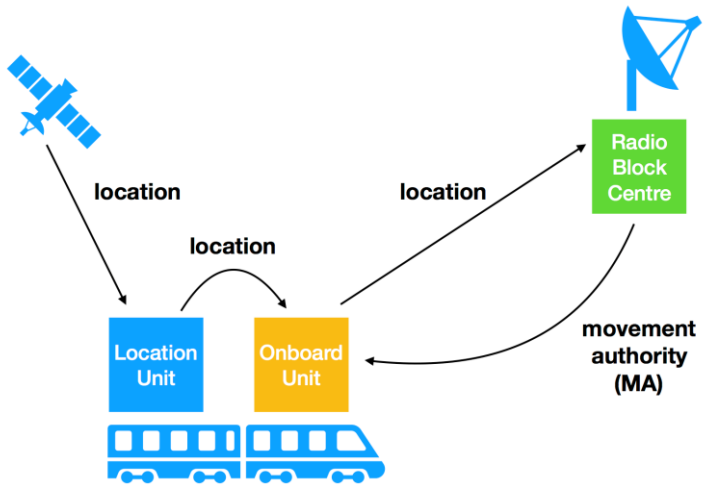

Figure 2 Moving-block principles and components

The model from D2.1, which implements the principles described above, is reported in Figure 3. This statechart model is composed of eight different parallel regions, each one characterised by a state machine.

In the 1st region, a location request is generated by the OBU, and then it is sent to the LU in the 2nd region. The LU receives the request and sends the location to the OBU in the 3rd and 4th region. In the 5th region the OBU sends the location to the RBC. In the 6th and 7th region, the RBC processes the location, and sends the movement authority to the OBU. The last region, apportioned again to the OBU, checks a timeout: if the OBU does not receive a movement authority within 20 seconds, it breaks the train – after interaction between the partners, the 20 seconds parameter was later refined in the requirements to 15 seconds, and then to 10 seconds in the final requirements (Sect. 5).

The model was used as a basis to interact between WP4 participants, from ISTI-CNR and SIRTI, and WP2 participants, from Ardanuy and SIRTI. Based on this interaction, made of brainstorming meetings and exchange of e-mails, a set of requirements for the moving-block system was defined by ISTI-CNR, reported below. This initial set was then refined and consolidated, as it will be described later (see Sect. 5).

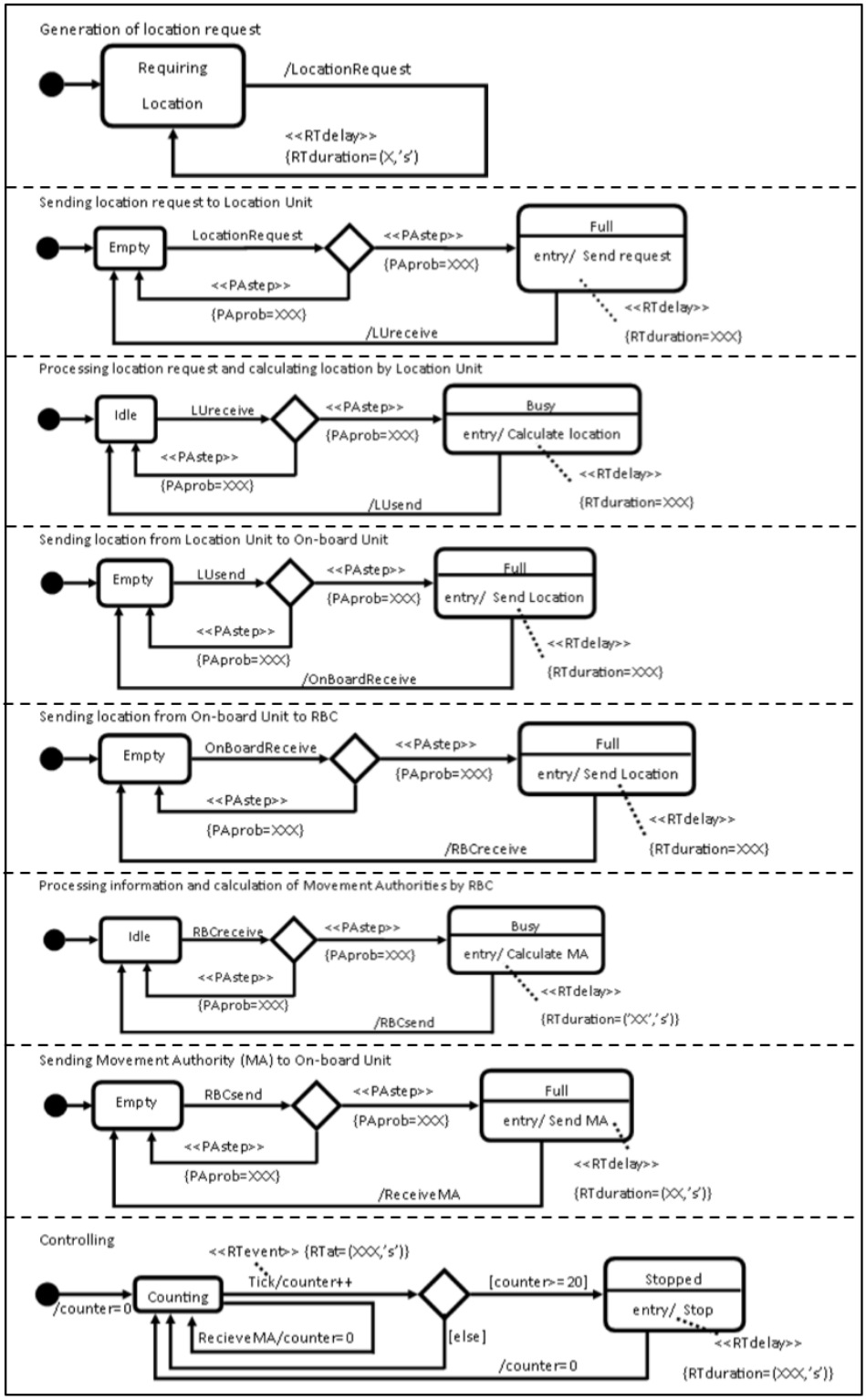

Figure 3 Moving-block UML Statechart from Deliverable D2.1

2.1.1 Moving-block Requirements

Below we report the initial requirements for the moving-block system. As the reader will notice, the requirements at this stage do not account for failures in the different components (e.g., when the location cannot be identified). These aspects are treated later during the refinement of the requirements, reported in Sect. 5.3.

Architectural Requirements

1. The system shall be composed of three components one On-board Unit (OBU), one Location Unit (LU), and one Radio Block Center (RBC)
2. The OBU and LU components communicate through a bi-directional channel
3. The OBU and RBC components communicate through a bi-directional channel
4. Each component is structured into phases
5. Each phase is independent from the others, and each phase does not suspend itself based on the status of the following one
6. Each phase produces information to be used by the following phase
7. The information is stored in a buffer of SIZE 1, for each phase
8. The buffer can be overwritten by the phase which writes on the buffer

Functional Requirements

1. Every 5 seconds OBU shall send a location request to LU
2. When the LU receives a location request, LU shall compute the location
3. After computing the location, the LU shall send the location to OBU
4. When OBU receives the location, OBU shall send the location to RBC
5. When RBC receives the location, RBC shall compute the movement authority (MA)
6. After computing the MA, RBC shall send the MA to the OBU
7. If OBU does not receive a new MA within 15 seconds from the reception of the last MA, the OBU shall stop the train

OBU phases

- OBU shall be composed of four phases: Generate Request, Send Request, Send Location to RBC, Receive MA
  - *Generate Request (GR)*
    - Every 5 seconds GR shall produce a location request for SR
  - *Send Request (SR)*
    - When SR receives a location request, SR shall send the location request to LU
  - *Send Location to RBC (SLRBC)*
    - When SLRBC receives a location, SLRBC shall send the location to RBC
  - *Receive MA (RMA)*
    - If RMA does not receive a new MA within 15 seconds from the reception of the lastMA, RMA shall stop the train

LU phases

- LU shall be composed of two phases: Calculate Location and Sending Location
  - *Calculate Location (CL)*
    - When CL receives a location request, CL shall produce the location of the train for SL
  - *Sending Location (SL)*
    - When SL receives the location of the train, SL shall send the location to OBU

RBC phases

- RBC shall be composed of two phases: Calculation of MA and Sending of MA
  - *Calculation of MA (CMA)*
    - When CMA receives a location, CMA shall produce the MA
  - *Sending of MA (SMA)*
    - When SMA receives the MA, SMA shall send the MA to the OBU

2.2 Moving-block Models Development

This section provides references to the tools adopted for modelling, the motivation for choosing such languages and tools, and refers to the models developed by means of the different techniques. Discussion on specific modelling languages and tools peculiarities, with model excerpts, is provided in Section 3.

2.2.1 Language and Tools Selection

The selection of formal languages and associated tools was based on the results of Task 4.1 and Task 4.2 (Deliverable D4.1) about the most mature formal languages and tools to be used in the railway context. From this analysis, a set of 14 languages and associated tools supporting modelling and formal verification was selected. Out of these 14 languages, in the context of the current deliverable, we selected a subset of 8 tools for modelling the moving-block system, namely Simulink, SCADE, UPPAAL, ProB, Atelier B, NuSMV, SPIN and UMC. We recall that the goal of this activity was mainly to assess the usability of the tools, and identify those that were considered more usable by railway practitioners.

Therefore, from the original set of 14 tools, we excluded those ones that, from the Ranking Matrix produced during Task 4.2, were marked as ADVANCED for the feature ``Easy to Use''. We recall that this feature was evaluated basing on the mathematical knowledge required to use the tool. One exception is Atelier B, which, although considered to require ADVANCED mathematical knowledge, was largely used in the railway context, and therefore was included among the selected tools[1]. A second exception is CPN Tools, which was excluded by the selection since, according to the review of the literature performed in Task 4.1 did not appear to have been used in industrial railway projects. Although this drawback was shared also by UMC, this tool was retained in the list since it is the only one that natively allows UML statecharts modelling and formal verification.

Below, we report a brief description of the selected languages and tools, together with the version adopted for the trial. For each tool below, and for the other tools considered in Task 4.1 and Task 4.2, a detailed analysis was reported in Annex 3 of Deliverable D4.1, named Tool Evaluation.

- **Simulink[2] (2017b):** Simulink is a model-based development tool that allows the user to graphically draw diagrams of the system modelled in the form of input-output blocks, which are assumed to be executed in a sequential order. The blocks can be further refined in the form of hierarchical state machines through the tool Stateflow, included in Simulink. Simulink supports graphical simulation of the diagrams, and Simulink Design Verifier, a package also included in Simulink, allows formal verification of properties on the diagrams. Simulink comes with several packages, also for code generation from the models.
- **SCADE[3] (19.0):** SCADE is a model-based development tool that, similarly to Simulink, supports the modelling of a system by means of input-output blocks. Differently from Simulink, the execution of the blocks is synchronous, i.e., at each execution step, all the blocks perform their computation simultaneously. Like in Simulink, the blocks can be further refined with hierarchical state machines, and the tool allows simulation and formal verification with SCADE Design Verifier. SCADE also comes with several packages, as, e.g., SCADE Architect, which allows the modelling of the system architecture in terms of high-level blocks, similarly to what can be done with the SysML/UML languages.

[1] Atelier B is widely used in the railway context, although it requires ADVANCED competences. Normally, the usage of the tools involves formal methods experts -- the Clearsy company (https://www.clearsy.com) offers consultancy services for Atelier B.

[2] https://www.mathworks.com/products/simulink.html

[3] http://www.esterel-technologies.com/products/scade-suite/

- **UPPAAL[4] (4.1):** UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types. It is appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables. Typical application areas include real-time controllers and communication protocols, in particular those where timing aspects are critical. The tool is developed in collaboration between the Department of Information Technology at Uppsala University, Sweden and the Department of Computer Science at Aalborg University in Denmark.
- **ProB[5] (1.10.2018):** ProB is an animator, constraint solver and model checker for the B-Method (see the B-Method site of Clearsy - http://www.methode-b.com/en/). It allows fully automatic animation of B specifications, and can be used to systematically check a specification for a wide range of errors. The constraint-solving capabilities of ProB can also be used for model finding, deadlock checking and test-case generation. The B language is rooted in predicate logic, arithmetic and set theory and provides support for data structures such as (higher-order) relations, functions and sequences. In addition to the B language, ProB also supports Event-B, CSP-M, TLA+, and Z. ProB can be installed within Rodin, where it comes with BMotionStudio to easily generate domain specific graphical visualizations. (See for an overview of ProB's components). Commercial support is provided by the spin-off company Formal Mind (http://formalmind.com)
- **Atelier B[6] (4.2.1):** Developed by ClearSy, Atelier B is an industrial tool that allows for the operational use of the B Method to develop defect-free proven software (formal software). It is used to develop safety automatisms for the various subways installed throughout the world by Alstom and Siemens, and also for Common Criteria certification and the development of system models by ATMEL and ST Microelectronics. Additionally, it has been used in a number of other sectors, such as the automotive industry. Atelier B is also used in the aeronautics and aerospace sectors.
- **NuSMV[7] (2.6.0):** NuSMV is a reimplementation and extension of SMV symbolic model checker, the first model checking tool based on Binary Decision Diagrams (BDDs). The tool has been designed as an open architecture for model checking. It is aimed at reliable verification of industrially sized designs, using as a backend for other verification tools and as a research tool for formal verification techniques. NuSMV has been developed as a joint project between ITC-IRST (Istituto Trentino di Cultura in Trento, Italy), Carnegie Mellon University, the University of Genoa and the University of Trento. Since version 2, it combines BDD-based model checking with SAT-based model checking. Its last evolution, called nuXmv, allows the verifications also of infinite-state systems.
- **SPIN[8] (6.4.8):** SPIN (Simple Promela Interpreter) is an advanced and very efficient tool specifically targeted for the verification of multi-threaded software. The tool was developed at Bell Labs in the Unix group of the Computing Sciences Research Center, starting in 1980. In April 2002 the tool was awarded the ACM System Software Award. The language supported for the system specification is called Promela (PROcess MEta LAnguage).
- **UMC[9] (4.6):** UMC is a verification framework developed at the FM&&T Laboratory of ISTI-CNR for the definition, exploration, analysis and model checking of system designs represented as a set of communicating (UML) state machines. Its current state is still that of an experimental framework mostly used for teaching and research purposes. Even if not ready for real industrial software development, it can play a role in disambiguating, animating and verifying early UML based designs.

[4] http://www.uppaal.org
[5] https://www3.hhu.de/stups/prob/index.php/The_ProB_Animator_and_Model_Checker
[6] https://www.atelierb.eu/en/
[7] http://nusmv.fbk.eu
[8] http://spinroot.com/spin/whatispin.html
[9] http://fmt.isti.cnr.it/umc/V4.6/umc.html

2.2.2 Modelling of the Moving-block System

The modelling process for the moving-block system was performed as follows. Three researchers from ISTI-CNR were appointed to develop the models based on:

A. the graphical UML model from D2.1
B. the requirements derived from that model, reported in Sect. 2.1.1

Each researcher independently developed the models for a subset of the tools. The researchers were required to interpret the graphical UML model and the requirements, and provide their interpretation using the languages of the tools. Furthermore, they were required to explore the capabilities offered by each tool at design and formal verification level, as, e.g., to verify certain properties, or to observe the graphical simulation capabilities of the tool. The result of this activity is a set of models of the moving-block system, together with a set of observations about peculiarities and capabilities of the different tools. The source models of the moving-block system can be downloaded from our public repository[10]: https://goo.gl/rcdVm2

We do not describe each single model in this section, as all the models follow the moving-block principles already described. Instead, in the following section, we present our observations about peculiarities and capabilities of the different tools, using excerpts of the developed models as a reference to explain such characteristics.

3 Discussion on Formal Modelling Techniques and Tools Trial

This section describes the peculiarities of the different methodologies and associated tools, based on excerpts of the models developed within this task, and on the experience gained by the researchers through the usage and comparison of the different tools. The goal of this section is to highlight how, depending on the phase of the development and the verification needs, certain tools may be more appropriate than others.

The information reported below is based on brainstorming performed by the three researchers involved in this modelling activity. During the brainstorming, each researcher using a certain tool was required to think aloud by answering three main questions, reported below. The other researchers challenged the speaker in case of disagreement.

When to use a certain tool/methodology?

- in which phase is it most appropriate? -- early prototyping, detailed design, etc.
- for which purposes or needs? -- verification of temporal properties, verification of data, etc.
- for which types of railway systems is it most adequate? Single systems: IXL -- working on tables; ATP -- complex control logic; systems-of-systems, with different interacting component (entire ERTMS, CBTC)

How does modelling and verification work with the tool/methodology? *(based on the lessons learned through the moving-block modelling)*

- what does a model look like?
- what does a formula to be verified look like?
- what does a simulation look like?

[10] For the SCADE model, we cannot provide the source for licensing reasons. Instead, we provide a video of the model. This video was also used in the tool showcase, as the SCADE model was developed by students from the University of Florence.

What issues should be considered by a user when choosing to use a tool/methodology?

- what are the situations, i.e., systems or phases, for which the tool is not appropriate?
- what are the potential hurdles that a company should consider if a certain tool is adopted? (e.g, steep learning curve, little documentation, etc.)
- how to address the mentioned issues in practice? (e.g., involvement of a consultancy company in case of higher competences required, usage of tools with different verification capabilities)

In the following sections, the different tools are treated in groups, when appropriate. Furthermore, each section is structured according to the questions listed above.

3.1 Modelling for Simulation and Code Generation

Two of the tools considered, namely Simulink and SCADE, are model-based development tools that offer graphical modelling and provide powerful simulation capabilities as well as code generation. Given their similarities, we discuss these tools together in this section, noting differences when appropriate.

3.1.1 When to use tools such as Simulink and SCADE?

Simulink and SCADE are mostly appropriate in two, rather different phases of the development, namely: (1) **requirements phase**, in which system prototypes are developed to support the definition of the requirements; (2) **detailed design phase**, in which the system model shall be close to the final implementation.

Indeed, these tools support **simulation** in the form of animation of graphical models, which can be useful in the initial phases of the development process, to provide first experiments, increase the confidence on the initial design, facilitate interaction with the customer, and establish the initial requirements. The simulation feature is also useful in the detailed design phase, in that it enables debugging capabilities. Furthermore, for the detailed design phase, these tools offer the **code generation** feature. The code generator of SCADE is also certified according to the CENELEC norms, making the tool particularly suitable to be employed for detailed design.

Both Simulink and SCADE are appropriate for modelling high-level system design control logics, with different subsystems interacting with each other, as well as single components, when these can be represented as state machines. They are not the most appropriate tools if one wishes to target formal verification, since the supported modelling languages are quite complex, and verification of models developed with these tools tend to incur in state-space explosion problems.

Both tools are appropriate for users with electronic engineering or software engineering background, while formal methods experts may find the tools too weak from the formal verification capabilities standpoint. SCADE is particularly suited for people with electronic engineering background, since also the supported statechart notation recalls the notation and the philosophy of electronic circuits. Instead, the statecharts notation used by Simulink -- and in particular by Stateflow, the Simulink package for state machines diagrams --, is more oriented towards computer scientists.

Finally, these tools are appropriate when a company wishes to have a holistic platform, covering different phases of the development -- from prototyping to code to testing--, and having different packages for multiple purposes, as, e.g., report generation, model-based testing. It should be noticed that, while SCADE is associated to SCADE Architect, which enables the development of the system architecture, in Simulink there is no such package for high-level architecture design. This problem can be addressed by creating different architectural hierarchies with the Simulink subsystems, i.e., blocks that can contain other blocks.

3.1.2 How does modelling and verification work with Simulink and SCADE?

Modelling with Simulink and SCADE is performed through drag-and-drop of elements from a graphical palette, similarly to what a user can do with tools such as Microsoft Visio. Both tools include libraries of pre-defined elements, which are called “blocks” by Simulink and “operators” by SCADE. Each block/operator is basically an input-output box, which perform some operation or implements some state-based function. Blocks/operators can be connected to each other to create complex models. The high-level model of the moving-block system for Simulink is presented in Figure 4. The first block on the left (in red) is external to the system, and models the behaviour of the train, in terms of distance travelled based on a speed value set by the user. Instead, the other three blocks model the OBU, the LU and the RBC, respectively. Information is exchanged between the blocks through signals, which are graphically represented either as direct links between the blocks or through named labels. INPUT can be modified (e.g., the train speed), and OUTPUT can be observed directly on the model (e.g., the brake activation value).

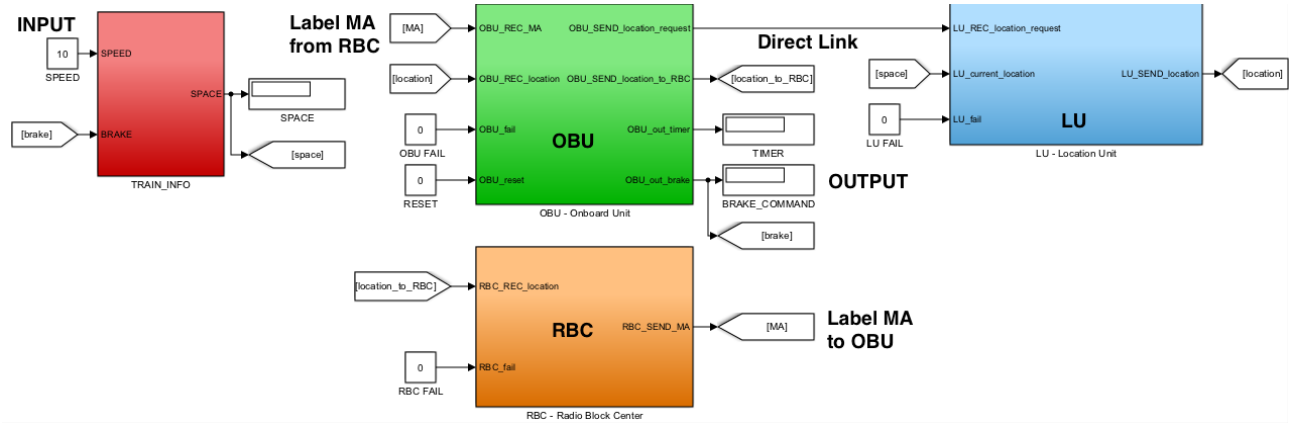

Figure 4 Simulink model

Similarly, the high-level model for SCADE is presented in Figure 5. With some graphical differences, the models include the same blocks and are conceptually similar.

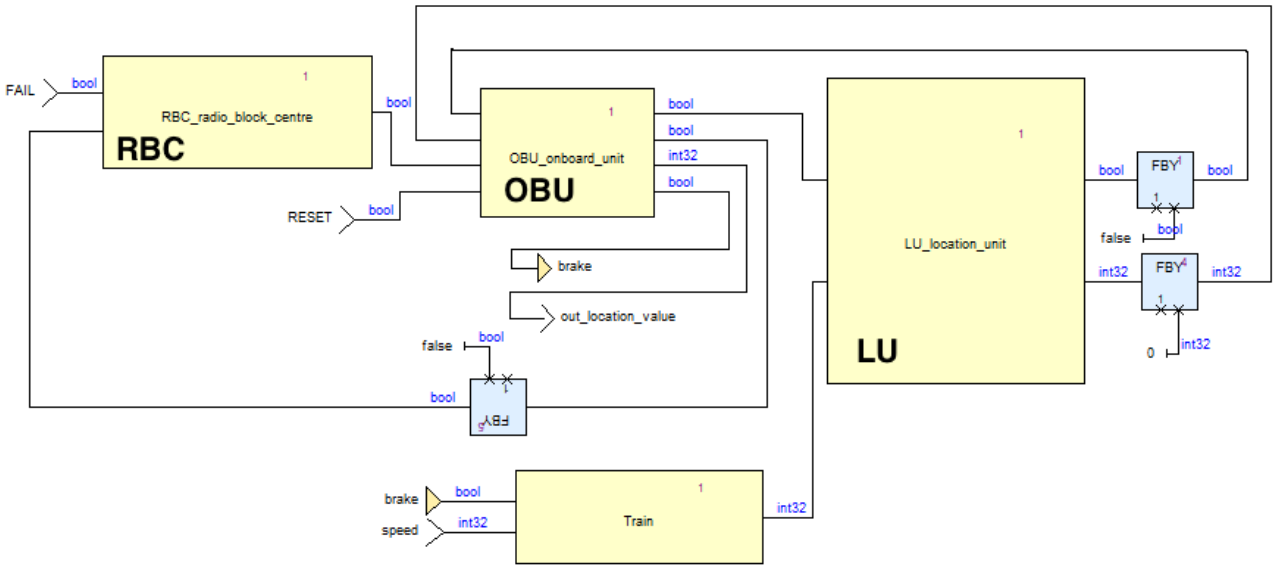

Figure 5 SCADE Model

Simulation with the tools is performed similarly, and the user can provide input to the modelled system, and observe the output. The user can also directly observe the system execution by means of different views. For example, as shown in Figure 6, the user can monitor the status of the Internal View of the OBU, and see the currently active states in the statechart. As shown in the right part of the picture, in Simulink, the user can also look at the contextual view of a certain module, and check the communication between sub-systems, in the form of message sequence charts.

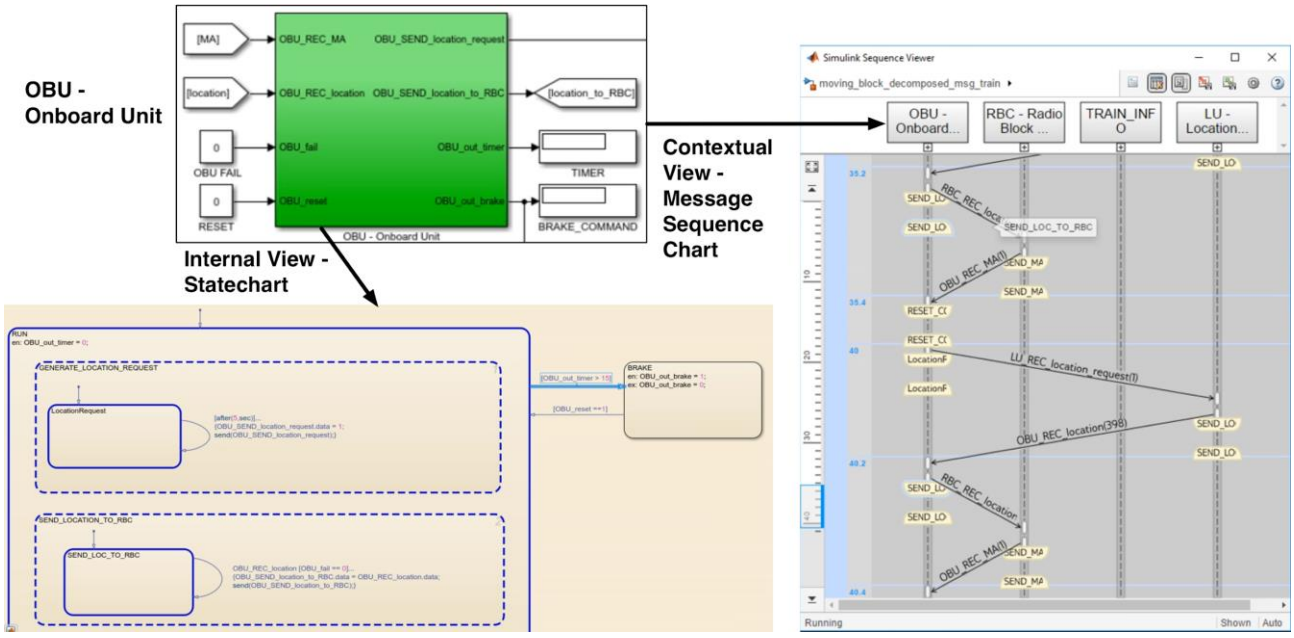

Figure 6 Hierarchical statecharts in Stateflow and Message Sequence Charts

Verification with the two tools can be performed by graphically defining properties to be verified, as shown in Figure 7. In the figure, the system with its input and output is included in the block of the left-hand side of the picture. The external elements on the right side are used to represent the property to be verified. The

property in this case can be written as: *anytime the timer associated to the OBU exceeds 15 seconds (because a new movement authority was not received), the OBU shall activate the brake*. Design Verifier generates a counter-example in case the property is violated, i.e., a scenario made of a sequence of input data that lead to the violation of the property. The user can then simulate the model using the counter-example as input, observe property violations, and debug the model to solve the problem.

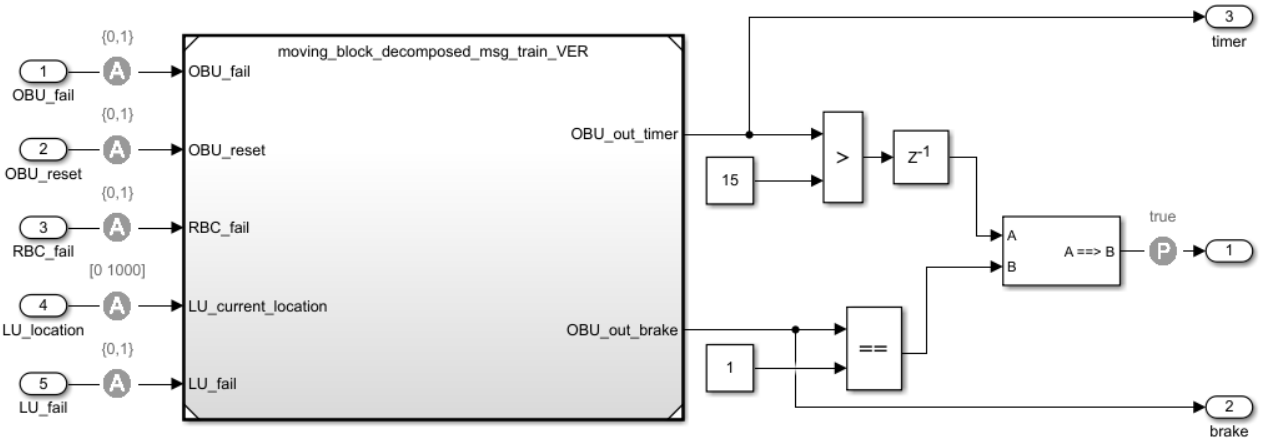

Figure 7 Property to be verified with Simulink Design Verifier

3.1.3 What issues should be considered by a user when choosing Simulink or SCADE?

The first issue to consider is that these tools, differently from other formal tools, are not open source, and have a licensing cost, which varies depending on the number of packages that one wishes to purchase. Therefore, the models developed with these tools can be normally read and executed solely by these tools -- some porting capabilities are available, but the formats are not open. Hence, if a company invests in these tools, and creates artifacts through these tools, creates a dependable business relationship with the tool vendors. Furthermore, the investment in terms of cost for the adoption of these tools may not be negligible.

Simulink and SCADE are similar for the features offered, but the supported languages are radically different in terms of semantics. SCADE is based on the synchronous language Lustre, and the state machines developed with SCADE evolve simultaneously. Differently, in Simulink each block and each state machine executes sequentially, similarly to what occurs with a sequential computer program. Although Simulink supports the modelling of different systems interacting with each other, as in the case of our moving-block system, the execution of the blocks is not *parallel*, as the parallelism is only simulated: each block executes after the preceding one, where default precedence follows the western reading direction -- left to right, top to bottom.

In practice, with Simulink one can modify a variable, and read the variable with the modified value in the same execution cycle, as one can expect in a sequential computer program. With SCADE, instead, a variable can be modified, but its new value will be available solely at the next execution cycle: to read a variable's value, one has to use the notation **last '<variable_name>** (indicating the previous value of <variable_name>), if the variable is modified somewhere in the model.

Another issue to consider is that these tools are not particularly suited for formal verification. Hence, if powerful verification capabilities are required, other tools should be used.

3.2 UML Modelling

UML statecharts, which describe the dynamic behaviour of a system in the form of a state machine, are often considered a reasonable way to communicate the expected behaviour of a system among team partners, developers, and assessors. The description of the moving block system included in Deliverable 2.1 (see Figure 3) is an example of the use of a UML-aware drawing tool to create pictures that complement a textual description of the desired system behaviour.

3.2.1 When to use UML-based tools?

The main driving force for using a UML-based tool is the need to have a graphical representation of the system behaviour in a format that is not particularly tool dependent and which can be rather easily understood by other persons or teams. The UML description of system behaviour is mainly centred on the control flow of the system. The data-dependent aspects do not appear in an evident way inside the behavioural description of a system as provided by statechart diagrams. By using UML-aware drawing tools (e.g. Visual Paradigm online[11], Magic Draw[12], etc.), it becomes easy to generate a picture that can actually be of help in communicating an approximate idea of what is the expected dynamic behaviour of a system.

By using UML-based frameworks that allow to translate a UML design into an executable program (e.g. IBM Rational Rhapsody[13], Enterprise Architect[14]), the user may have the possibility to animate the system executions, to exploit model-based testing features of the framework, and finally generate the programming code associated to each system component.

When it comes to verifying overall behavioural properties of the system under development, UML suffers from the absence of mature industry-ready formal verification tools. So this choice is probably not the best one if some kind of formal verification of the system is actually a major concern.

3.2.2 How does modelling and verification work with UML?

In the activity of WP4, being our focus on formal methods, we are not particularly interested in the possible use of just graphical or semi-formal UML based tools. In the absence of industry-mature frameworks we have experimented the use of an academic UML-based verification framework that is UMC.
This framework allows the user to describe a system as a collection of communicating state machines specified by statecharts, and to animate, and verify properties directly on these UML-based designs. UMC is not a tool for quantitative verification of system properties, therefore the time/probabilistic aspects of the system will not be represented, and the focus will be centred on the functional aspects of the system.

The original WP2 description of the system (as shown in Figure 3) identifies seven functional phases, logically connected in sequence, that define the intended behaviour of the system. In particular, in that presentation the system has been structured as a single parallel state where each functional phase is modelled by its specific concurrent region. Each phase activates the next one through the sending of a signal. There is also an external "tick" event triggering the beginning of a new cycle. The above structure can be directly encoded in UMC (see umc-WP2orig.txt in our public repository https://goo.gl/rcdVm2) as a single state machine definition, and the overall system behaviour can be analysed by model checking of CTL-like temporal logic formulas.

[11] https://online.visual-paradigm.com
[12] https://www.nomagic.com/products/magicdraw
[13] https://www.ibm.com/us-en/marketplace/rational-rhapsody
[14] https://sparxsystems.com/products/ea/

We can show, for example, that the system design satisfies a rich set of properties like:

- *as soon as the OBU executes 10 cycles without receiving an MA the OBU moves to the STOPPED state*;
- *every system evolution path from a non-stopped train status eventually leads either to the stopped status or to the handling of a MA*;
- *there is a possible system evolution path in which the OBU infinitely receives sequences of MA*;
- *there are no deadlocks (any final state is the state in which the train is stopped)*.

The UML design of the Moving Block modelled in this way reflects rather well the desired requirements under the implicit assumption that the activity of the whole system cycle is completely exhausted by the time a new "tick" event arrives. If this assumption were not true, we would have a parallel system with several concurrent regions progressing at the same time, and this would lead to ambiguities and uncertainties of the really intended behaviour.

Another limit of this original model is that RBC and OBU are considered as two synchronous regions of the same system state, i.e. performing the same cycles in a fully synchronous way, while in reality these two components are two separate (and remote) entities, that may have two cycles of the same duration, but which are not necessarily perfectly synchronized. To overcome these limits we have generated a new UML design (see umc-WP2-OBUseq+RBCseq.txt in our repository https://goo.gl/rcdVm2) that, still preserving the functional description of the system in different phases, does not have the above mentioned drawbacks.

In this second case we model RBC and OBU as two distinct (but communicating) objects. For simplicity, the LU component is in this case just considered an internal activity of the OBU component.

Each object is described by a sequential statechart structured in sequential phases, as shown, for the case of the OBU component, in Figure 8. This new model can be proved to satisfy the properties mentioned above of the original model, however its behaviour differs in some other aspects. For example, it is now also possible that during one OBU cycle a position report is sent to the RBC, while the corresponding MA message is received during the next cycle. Moreover, it is now possible that during one OBU cycle the train has to handle two incoming MA messages (one arrived from the PR sent during the previous cycle, and one as response to a new PR freshly generated and sent in this same cycle). This situation will no longer be present in the revised version of the Moving Block system, because in this new version Position Reports are no longer sent by the OBU at each cycle, but only after a certain minimal delay. The current model continues to rely on the assumption that each OBU and RBC cycle lasts less than the planned duration of the cycle itself. This assumption is released in the revised version, in which it is explicitly required that when an OBU (see umc-WP43-newOBU.txt in our repository https://goo.gl/rcdVm2) or RBC cycle lasts more than the expected cycle duration a fatal error is generated.

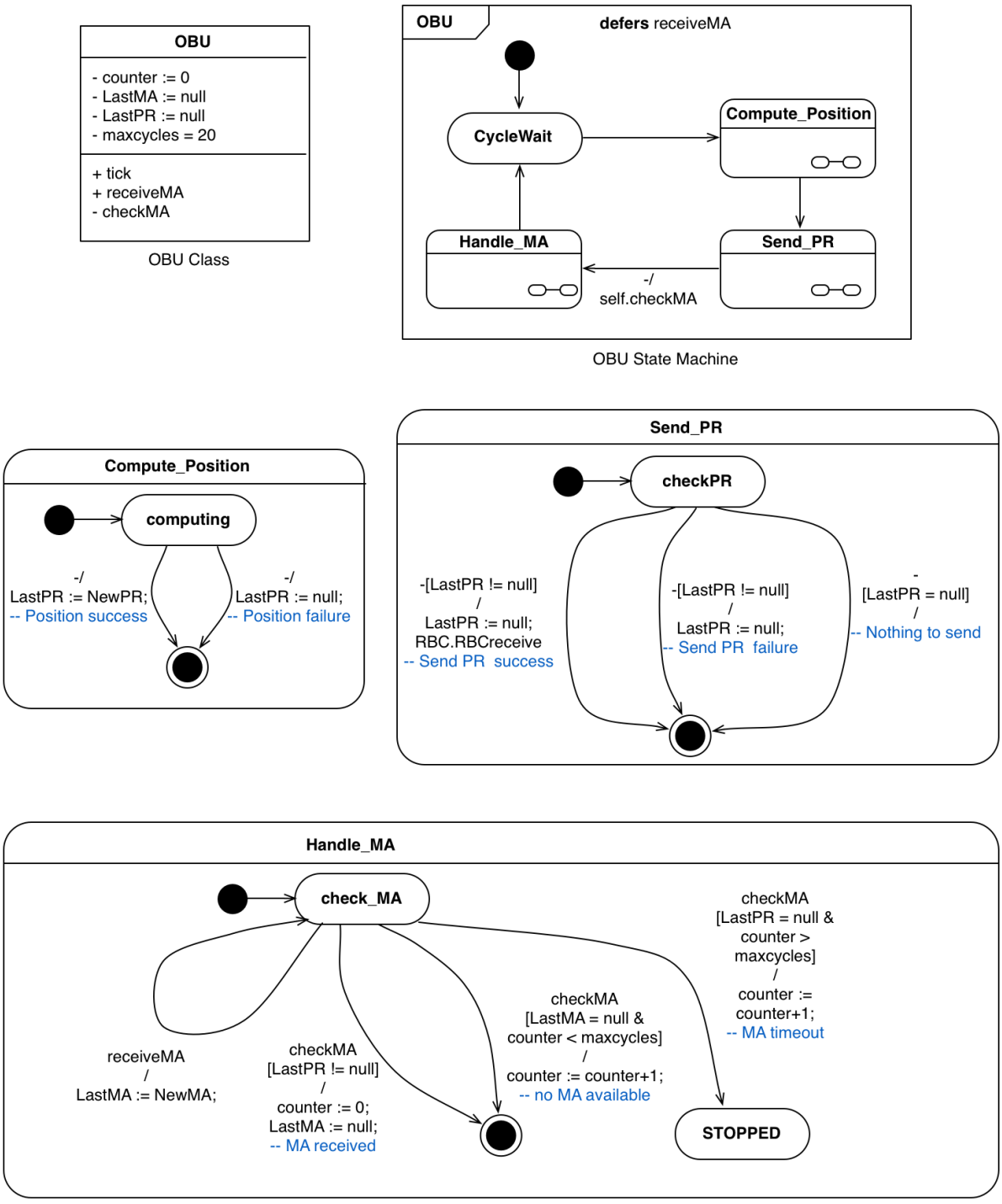

Figure 8 Sequential OBU Component

3.2.3 What issues should be considered by a user when choosing UML-based tools?

A first well-known difficulty is the absence of a really rigorous semantics of the UML notation, which currently contains several aspects described by the official UML specification in an ambiguous way. Another difficulty caused by the high level of freedom left to the implementation, is deciding what will be the effective behaviour of a system with respect to a rich set of implementation-dependent aspects. A third difficulty is that, while the behaviour of a single system component describing a state machine is still reasonably well

covered by the UML standard, the description of behaviour resulting from the parallel composition of these components, and the way in which the communications among them occurs, is left almost completely undefined.

Clearly, if some UML-aware statechart drawing-only tool is used, the fact of not being able to animate/simulate/verify the design prevents the user from having a precise feedback on the correctness of the drawing with respect to the original user's idea. By using some proprietary UML-based framework that allows to translate a UML design into some executable program, the user may achieve a much greater confidence of the correctness of the design, possibly also exploiting model-based testing features of the framework, but in this way the design may actually hide many implementation-dependent choices that might be easily misunderstood when that design is observed in a different context.

The use of UML-based formal verification and analysis tools like UMC, that allow to describe a system as a set of concurrent state machines with a precise semantics, simulate the possible system evolutions, and verify temporal properties on the dynamic behaviour of the system, might greatly increase the level of confidence of the overall correctness of the abstract UML design. But the issues related to the presence of some platform-specific implementation dependencies and the issues related to the correctness of the actual code eventually generated from the original design still remain unresolved. Moreover, tools of this kind, that are being developed in academic contexts, do not have the maturity for being included into an industrial development process.

3.3 Modelling Real-time and Probabilistic Aspects

The description of the moving block system included in Deliverable 2.1 (see Figure 3) makes use a semi-formal UML State Machine dialect called Real-Time UML (RT UML) that allows to specify both real-time and probabilistic aspects.

In particular, the semi-formal model in Figure 3 specifies timed events denoted as *RTat* of stereotype <<*RTevent*>>, probabilistic delayed events *RTduration* of stereotype <<*RTdelay*>>, and probabilistic events <<*Pastep*>> with probability weight *Paprob*. These real-time and probabilistic aspects have been used to model both failures in communications, and delays in transmitting messages and elaborating them. However, such a semi-formal model lacks a really precise semantics and hence it is not directly amenable to formal analysis and verification, and even less to automatic code generation.

Formalisms and tools (like UPPAAL) for specifying and analysing these types of specifications are useful because they primitively allow to model both probabilistic and real-time aspects, making the formal model ready for automatic analysis without further codifications that would tamper productivity and simplicity of the model.

3.3.1 When to use a tool that supports real-time and probabilistic aspects?

Clearly, a time/probabilistic aspects-oriented tool is useful when these aspects play an important role in the definition of the expected system properties. It is also reasonable to imagine that, given a design specified in a functional way with other approaches, fragments of it are also modelled with tools like UPPAAL for a more specific time-oriented verification. In our specific case of the trial moving block application, we have several requirements expressed under the form of time assumptions (e.g., OBU cycle of 500 ms, the train must stop if no MA is received for 15 seconds), however these assumptions are very simple and can be easily approximated without a rigorous modelling of the flow of time (e.g., the train must stop if no MA is received for 20 consecutive OBU cycles).

Tools like UPPAAL would allow to better understand the underlying time/probabilistic aspects, allowing to analyse properties (when all the relevant number are provided) such has:

- what is the probability for the train to enter in the Stopped state within 10 seconds?

- if the OBU-LU communications delays are in the range 10-100 ms, what is a reasonable requirement for the response time of LU to guarantee that the OBU cycle of 500 ms is never preempted?

3.3.2 How does modelling and verification works with UPPAAL?

UPPAAL allows to describe a system as a set of concurrent *Timed automata*. These are drawn graphically as simple statecharts, and interact either through the exchange of synchronous (not buffered) messages or through shared memory. *Timed automata* combine discrete systems with real-valued variables that evolve during the time a system spends in a state. These variables, called clocks, evolve uniformly and they can be used for guarding transitions between states and they can be used to specify invariant properties.

UPPAAL SMC is an extension of UPPAAL that allows to express both stochastic and non-linear dynamic features, by adopting a stochastic and hybrid extension of timed automata. *Stochastic timed automata* include also probabilistic transitions and delays.

Figure 9 shows the formalization of the Moving Block trial application (from the original WP2 design). The explicit numbers included in the specification are just examples of possible values.

Verification allows to verify, among others, safety properties and reachability properties. In addition to standard model-checking techniques for properties such as reachability or deadlock-freedom, in UPPAAL it is possible to evaluate the probability that a random run of a network M satisfies a property ϕ in a given amount of time.

UPPAAL SMC uses Statistical Model Checking (SMC) to evaluate probabilistic properties of interest. SMC is concerned with running a sufficient number of (probabilistic) simulations of a system model to obtain statistical evidence (with a predefined level of statistical confidence) of the quantitative properties to be checked.

SMC offers advantages over exhaustive (probabilistic) model checking. Most importantly: SMC scales better, since there is no need to generate and possibly explore the full state space of the model under scrutiny, thus avoiding the combinatorial state-space explosion problem typical of model checking, and the required simulations can trivially be distributed and run in parallel. This comes at a price: contrary to (probabilistic) model checking, exact results (with 100% confidence) are out of the question.

A highly appreciated feature of UPPAAL is the possibility to interactively simulate possible paths of executions and visualise them in the form of message sequence charts.

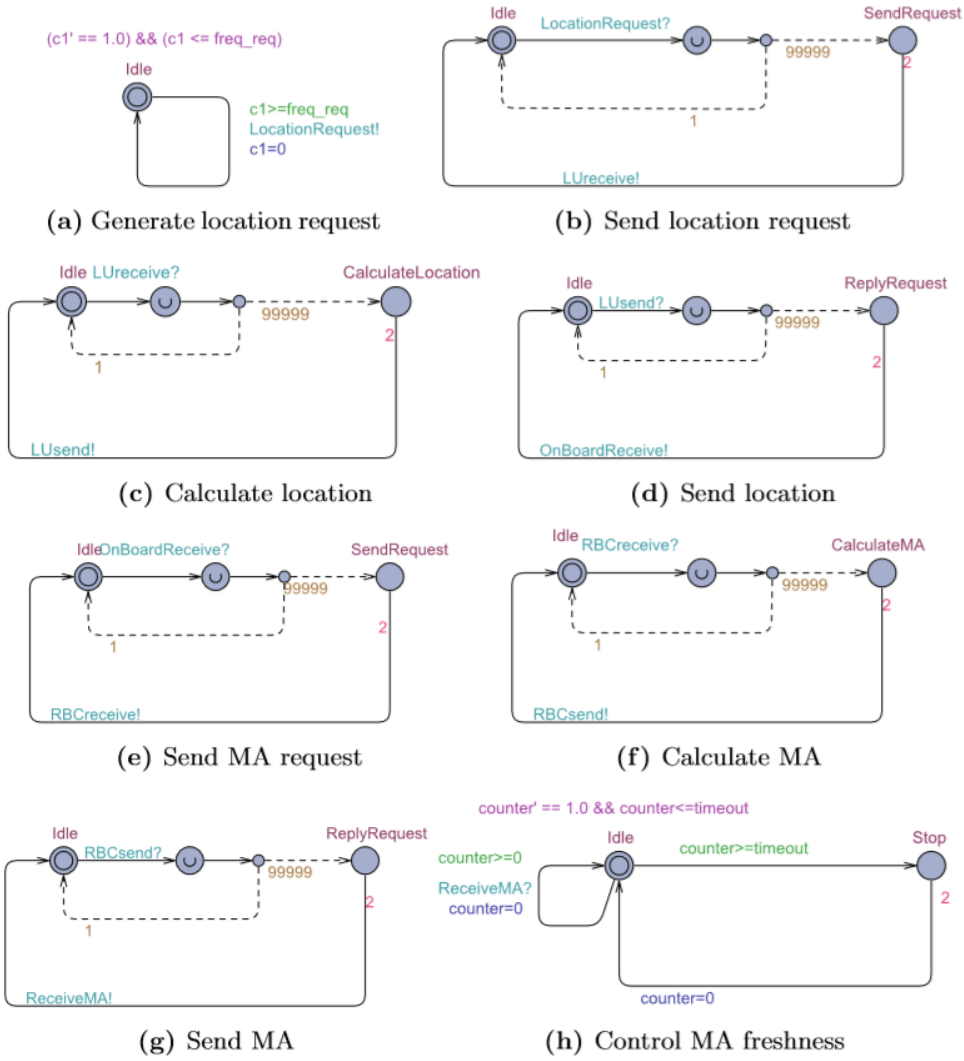

Figure 9 Stochastic timed automata formalisation of the moving-block system

3.3.3 What issues should be considered by a user when choosing to use UPPAAL?

While standard functional analysis methods are a subject studied for decades and for which several industry-mature tools (such as model checkers) exist, tools oriented to SMC, as in our case, are very recent. For example, the first version of UPPAAL SMC has been released in 2014. As in the case of classical model checkers, certification of UPPAAL SMC for the development of critical software is not available.

Several problems, which are known to be decidable in discrete systems, cease to be so in real-time probabilistic frameworks. Hence the set of properties on which analysis can be carried out is more limited. Since this is a rather new and hot topic in the research community, the technological transfer and the adoption to industry is still limited. Indeed, new theories are being developed nowadays, which use results from statistics as well as machine learning, and it is expected that in the future more powerful tools will become available.

Moreover, even if the formalism is closer to state machines used in tools such as Simulink, a knowledge of the underlying mathematical formalisms used to specify both the model and the properties as well as to analyse the obtained results is still required. Therefore a certain specialisation of developers, engineers and other users on real-time stochastic model-based analysis is needed.

3.4 Modelling with Event B State Machines (Refinements)

A formal design methodology, that appears to be quite widely used in the railway context (see Deliverable D4.1 on literature and tools survey), is the design methodology based on State Machines as described by the Event B methodology. The B/Event B method appear to be a very productive approach, that has led to the development of a rich ecosystem of tools and frameworks. E.g. Rodin[15], ProB, Atelier B, are all examples of coordinated verification environments supporting the formal analysis and development of Event B specifications.

The characteristics of this approach is that a system is described as a sequential (possibly non-deterministic) state machine, whose evolutions are triggered by operations enabled by specific local state machine conditions, but originated from the external environment. The specification of a state machine includes the definition of the set of local variables, a set of state-based invariant properties that are required to hold throughout the machine's evolutions, and the specification of the set operations that under specific conditions are allowed to modify the value of the local variables of the machine.

With this kind of methodology three different kinds of formal verification can be carried out:

1) The specified set of invariants are checked to remain satisfied throughout the machine's evolutions (i.e., no operation ever causes the violation of an invariant).

2) A machine definition can be refined by adding more implementation-dependent details, and the refined one can be proved to be correct (i.e., still satisfying the invariants of the higher level machine). This may lead to a sequence of refinements that eventually reaches a level very near to final programming code.

3) Behavioural properties (expressed in linear- or branching-time logics) can be stated and verified by means of model checking.

This methodology is particularly well suited for data-oriented systems, in which the consistency and correctness of the data (status of the local variables of the state machine) is of primary importance. Indeed, the formal modelling of interlocking system are among the most common applications of this methodology.

This approach is also well suited for modelling a system "out of the loop", i.e., as a single component interacting with a generic "external environment", whose safety can be guaranteed by the verification that for any possible external interaction the validity of the machine invariant properties always remain satisfied.

This approach instead does not fit well the need of modelling a system composed by several concurrent components interacting through events, signals, messages.

The moving block design which we are interested in modelling is still a high-level design that reflects more the designer's requirements than the designer's detailed specifications to be used for the actual coding, and is not particularly data oriented. Therefore, we are more interested in the third kind of formal verification of the three mentioned above. We have found that the tool that fits better this need is the ProB tool. Atelier B has also been considered during the tool trial and will be discussed afterwards.

3.4.1 When to use tools such as ProB?

The main characteristics of ProB is that it allows to observe, simulate, analyse the dynamic behaviour of a state machine as described by the system evolutions graph (where each step corresponds to a state

[15] http://www.event-b.org

transformation triggered by external event). Therefore, it fits very well the initial needs of observing the behaviour of a prototypical design, as well as the need of proving dynamic properties of the possible evolutions of a more consolidated design.

3.4.2 How does modelling and verification work with ProB?

We have developed three examples of Event B models for the moving block case study. A first one (see prob_MB_WP2.mch in our repository https://goo.gl/rcdVm2) directly reflects the original WP2 design with its known limits. A second one (see prob-OBU.mch) describes just the sequential OBU component (the same described in the UML specification of Figure 8) and its possible interactions with an external environment that replies to sending of position reports with the return of MA messages). A third one (see prob-OBU+RBC.mch) models the composition of the OBU component and the RBC component (that communicate through global variables) merging them inside a single state machine.

Even if the ProB tool is provided with a nice graphical user interface, the specification itself is in plain textual form. Figure 10 shows a fragment of the OBU specification as a ProB state machine. In terms of behavioural properties, we can prove by model checking that the prob-OBU.mch specification satisfies the same sample list of properties verified on the UML model.

Figure 11 shows the Check LTL/CTL Assertion window that is opened when we want to verify some temporal property on the model. We can see how the tool allows the verification of all the LTL or CTL assertions included in the model specification, or the introduction and verification of new temporal assertions. A green flag notifies the validity of the formula. When red, clicking over a flag results in the display of a counterexample for that formula.

The Prob tool seems to be a rather industry-mature tool, well integrated with other verification / analysis / visualisation tools. Beyond the model checking of LTL/CTL properties it allows the verification of the absence of invariant violations or deadlocks; it allows to support some kind of model-based testing and produces abstract coverage data as the result of analysis and exploration of the system. System evolutions can easily be simulated step by step, and the generated state space (in its entirety or some projection of it) can be graphically visualised.

Moreover, a ProB specification might be exported (after some rewriting) to other tools of the Event B family, like Atelier B that provides, beyond refinement and invariant verification functionalities, also actual code generation facilities.

3.4.3 What issues should be considered by a user when choosing ProB?

The strong point of ProB is its capability of performing model checking of LTL or CTL formulas on the Event B state machine under design. The efficiency of the built in model-checking capabilities are however not comparable to those of more specific verification engines like those behind SPIN, SMV, CADP, or FDR. ProB is therefore a good choice only if the system is not particularly big. This is somewhat coherent with the expected use of the Event B method to model single system components, and not in large architectures.

ProB is however a very flexible tool, that allows to export the developed design to other tools of the Event B ecosystem like the Rodin and Atelier B tools. Moreover, ProB also allows to model check (not too complex) specifications imported by other frameworks like SPIN, TLA+ or FDR, providing additional verification capabilities with respect to those provided by the corresponding original tools.

```
MACHINE MOVINGBLOCK
SETS  DEFINITIONS  CONSTANTS  PROPERTIES  INVARIANT  INITIALISATION

OPERATIONS

  OBU_Compute_Position =
    PRE
       STATUS = CYCLEWAIT
    THEN
       STATUS := SENDING;
       CHOICE
         // success
         LastPR := NewPR
       OR
         // failure
         LastPR := null
       END
    END;

  RBCsim =
    PRE
       SentPR = NewPR
    THEN
       CHOICE
         // success
         LastMA := NewMA;
         SentPR := null
       OR
         // failure
         SentPR := null
       END
    END

  OBU_Send_Position =
    PRE
       STATUS = SENDING
    THEN
       STATUS := CHECKING;
       CHOICE
         // success
         SentPR := LastPR;
         LastPR := null
       OR
         // failure
         LastPR := null
       END
    END;

  OBU_Check_MA =
    PRE
       STATUS = CHECKING
    THEN
       IF LastMA = NewMA THEN
          // handle MA
          counter :=0;
          STATUS := CYCLEWAIT;
          LastMA := null
       ELSE
         IF counter >= maxcycles THEN
           //TIMEOUT expired
           counter := counter+1;
           STATUS := STOPPED
         ELSE
           // no new MA
           STATUS := CYCLEWAIT;
           counter := counter+1
         END
       END
    END;

END
```

Figure 10 A fragment of the ProB OBU state machine

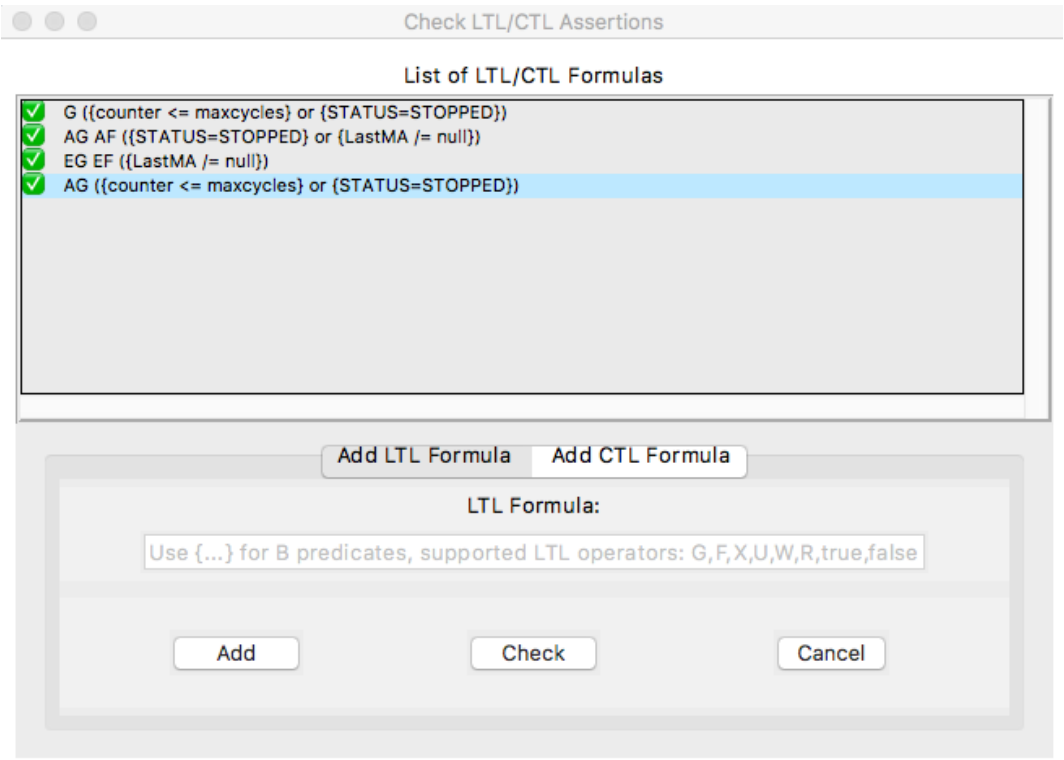

Figure 11 Properties in ProB

3.4.4 When to use tools such as Atelier B?

Atelier B is an interactive theorem prover for assessing the preservation of structural properties (in the form of invariants) on the status of an Event B model. This approach is particularly useful when a state machine has a complex internal status, of which the possibility to guarantee the preservation of consistency is of primary importance. In our trial case study the design under examination does not have this characteristic, since the local machine status is essentially constituted by a "counter" variable (the maximum number of cycles allowed to be executed before stopping the train in absence of an MA message) whose value should invariantly remain in the range 0..20. Nevertheless, in the railway context several examples can be found for which this approach might be useful (e.g., interlocking systems). Given that it belongs to the B method suites, it can be applied in the whole life-cycle of software.

The strongest point of using a theorem prover with respect to, for example, a model checker such as ProB, is the possibility of verifying properties for systems with a potentially infinite number of states. Indeed, in this case an exhaustive exploration of the whole state space of a system would not terminate. However, thanks to theorem proving, it is still possible to prove certain requirements using, e.g., induction or other proof techniques.

On the other side, proving a theorem is not a completely automatic procedure and it requires several interactions with the user, who is in charge of selecting the specific strategy to prove a certain result. Nevertheless, Atelier B is equipped with a feature for trying to automatically prove certain simple properties, which does not always succeed but can often be helpful.

When an Event B model is refined, Atelier B automatically generates proof obligations to be discharged by proving them. This is crucial to verify that the new model is indeed a correct refinement of the previous one. This feature is of help for driving developers in creating and proving correct refinements. Such proof

obligations mainly consist in proofs of preservation of invariants by refinement, and that pre and post conditions of events are correctly refined.

The language of Atelier B uses mathematical objects, as for example sets, existential or universal quantifiers, data types, relations, functions, etc. This aspect has great advantages when the system to be modelled and verified can be naturally rendered as a set of logical connectives, as is generally the case for interlocking tables.

Of course, one of the strongest point of Atelier B is the integration in the whole B method and its previous notorious applications in the railway domain, with a supporting community and a company who offers support to developers at various levels.

3.4.5 How does verification work with Atelier B?

During the tool trial, Atelier B has been used inside the Rodin environment, displayed in Figure 12.

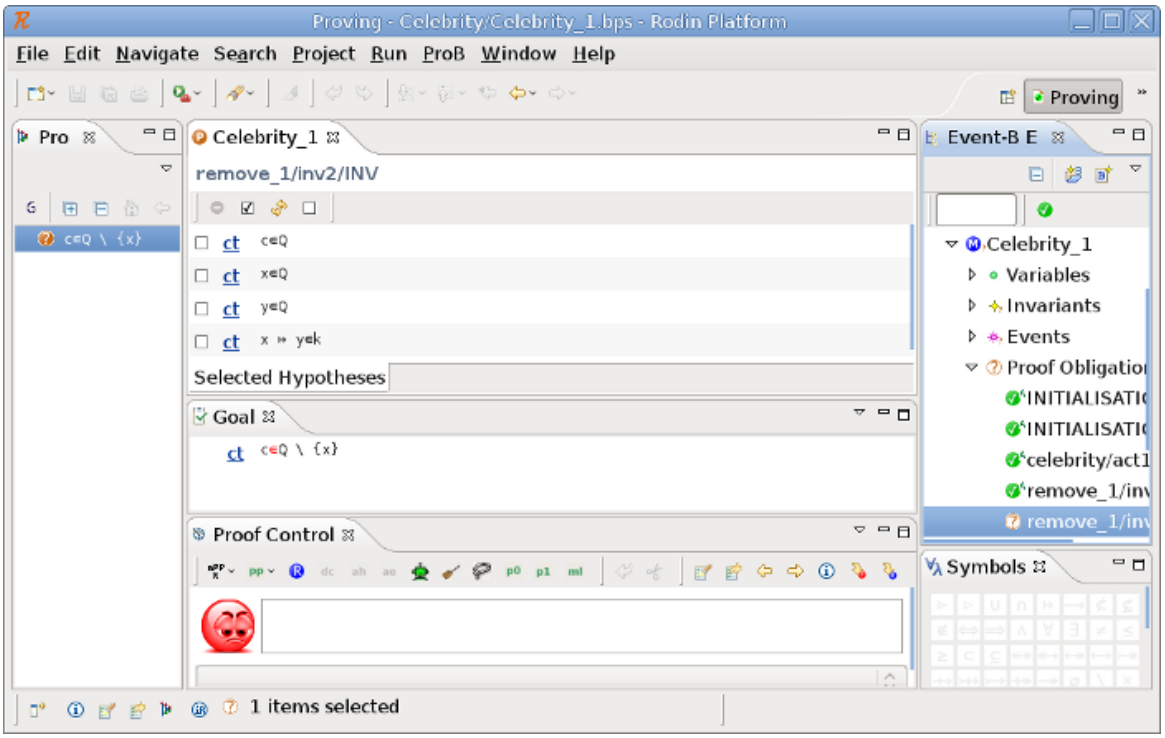

Figure 12 Atelier B within the Rodin environment

When interacting with the theorem prover, several proof obligations to be discharged are displayed on the right column. In the bottom center panel the proof control allows the user to apply different strategies for trying to prove the current proof obligation.

As an example, it is possible to select and apply previously proved properties, to introduce new hypotheses, that are displayed in the upper central panel, to instantiate an existential quantifier, to divide by cases, to try to automatically prove the current proof obligation, and others. The goal to be reached is displayed in the middle center part. The left hand-side panel shows the current proof tree, i.e., the steps of the proof that have been taken so far.

Fully explaining how theorem proving works is out of the scope of this document. However, it generally must be considered that performing theorem proving requires some skills and previous experience with such techniques.

3.4.6 What issues should be considered by a user when choosing Atelier B?

When planning to integrate Atelier B in the development process, a first issue to consider is its strong mathematical foundation in theorem-proving techniques. Hence, the cost of training developers who are not familiar with such techniques is non-negligible and must be taken into account.

Contrary to the case of model checking, it is not a completely automatic verification technique, hence it requires more human effort and expertise. Moreover, Atelier B is not well-suited when properties related to the temporal evolution of a system have to be analysed, for example through temporal logics. Indeed, in this case a model checker like ProB is suggested.

Furthermore, when non-functional aspects such as for example reliability or availability have to be studied, Atelier B might not be the best option. Indeed, quantitative aspects such as real-time and probabilities are not provided off-the-shelf in Atelier B, if compared to other tools that instead primitively provide such capabilities.

Finally, from the Event B language Atelier B inherits difficulties in modelling concurrent systems and interacting components.

3.5 Other Model-Checking Approaches to deal with State Explosion

The literature survey conducted as part of Task 4.1 has revealed uses of the SPIN and SMV frameworks (NuSMV, nuXmv) for the verification of systems in the railway context. These are command-line tools which are also widely known within the formal method community.

3.5.1 When to use tools such as SPIN or SMV?

Both SPIN and SMV can better be seen as very powerful model-checking engines, that exploit state-of-the-art techniques for the reduction or the control of the state-space explosion problem. A SPIN or SMV specification, however, can hardly be seen as a friendly notation for the unambiguous sharing of a design among different stakeholders, neither can it be seen as a usable document that can be passed to software developers for its actual refinement into final executable code. Nevertheless, sometimes a translation of a system design into Promela (the specification language of SPIN) or SMV can be an effective way to verify properties of the initial design. For example, one of the most interesting uses of SMV is the verification of hardware designs. Indeed, in this case the initial specification is likely a graphical design, and the final product a printed circuit, nevertheless the possibility of modelling the design and proving its correctness before production could be of paramount importance.

3.5.2 How does modelling and verification work with SPIN and SMV?

Both tools take as input a textual description of the system in their own language and allow to verify formulas expressed in a temporal logic (LTL for SPIN, LTL and CTL for SMV). SPIN allows to design a system as a collection of non-deterministic sequential asynchronous processes that interact through buffered message passing. SMV instead sees a model as a global state machine whose non-deterministic evolutions can be defined using a data flow or a transition-oriented approach. In SMV a system can be decomposed into a set of fully synchronous submachines. Examples of coding of the Moving Block system for both frameworks can be found in SMV_MB.txt and SPIN_MB.txt (in our repository https://goo.gl/rcdVm2)

3.5.3 What issues should be considered by a user when choosing tools line SPIN or SMV?

When the system under investigation becomes rather big, the effective use of the tools requires a deep experienced knowledge of the verification framework. The choice of the appropriate execution options may become essential for the success of the verification task.

3.6 Modelling for System Wide Analysis (Systems in the Large)

In our specific case our reference model is composed of just three components. We might sometimes be interested in checking the behaviour of richer systems, e.g., with more OBUs and more interacting RBCs.

In this case, a compositional approach that allows to verify the properties of a component by composing it in parallel with a (minimised) abstraction of all the other components would be very useful to avoid the problems of state explosion usually arising when we have a system composed by many concurrent objects.

In order to support this compositional approach to model checking, one needs an underlying specification language with a solid mathematical definition and upon which rigorous theories of equivalence of behaviours can be applied. All the tools that support this approach are indeed based on specification languages formally defined as process algebras. In the survey performed as part of Task 4.1 and 4.2 we have mentioned and described three such frameworks, namely, CADP[16], mCRL2[17], and FDR4[18]. As part of Task 4.3, even if with the current architectural design we are not compelled to do so, we have experimented the modelling of our system composed by one OBU and one RBC also with two of these specification languages (LNT for CADP and CSP for FDR4). Given that the languages used by these tools required advanced competences, these tools were not subject to the showcase and tool usability assessment presented in Sect. 4. We however consider it useful to discuss the applicability of these tools, for the sake of completeness.

3.6.1 When to use tools such as CADP or FDR4?

The reasons for using these tools for the specification and the verification of a system are twofold. From one side we can rely on a specification language that has a formally defined (tool-independent) semantics. This allows to have specifications that can be shared among the various stakeholders, being certain of their unambiguous meaning. On the other side, the theories that have been developed for such languages allow to fully exploit the above mentioned compositional approach to systems-in-the-large verification. A typical exemplary use of these approaches is the formalisation of communication protocols. Indeed, in this case a rigorous mathematical specification, whose correctness can be formally verified and that can be used as a reference by different producers for the development of mutually compatible products, is an important starting point for the development of correct working software, even if the final software products are actual separately developed by the various producers according to their own software development process.

3.6.2 How does modelling and verification work with CADP or FDR4?

All the mentioned specification languages rely on a textual description of the system. The single components are defined by sequential processes (with no shared memory), these processes are composed in parallel with appropriate operators and synchronised through the execution of actions. In our case we have that the communications between OBU and RBC logically behave as read/write operations upon one position re-writable buffer. While in the case of UML these buffers were implicitly provided by the language (events queue) or directly mapped into memory locations (Event B), in the case of process algebras these buffers must be explicitly encoded as additional system components (therefore their precise semantics becomes explicit, and not implementation-dependent as in the case of UML). Examples of the resulting specifications can be found in the models cadp-MB.lnt and fdr4-MB-WP2.txt reported in our repository https://goo.gl/rcdVm2. In the case of CADP the properties to be verified can be expressed in a quite powerful

[16] https://cadp.inria.fr/
[17] https://www.mcrl2.org/
[18] https://www.cs.ox.ac.uk/projects/fdr/

branching-time logic (MCL). In the case of FDR4 the properties to be verified must be expressed as refinement assertions.

3.6.3 What issues should be considered by a user when choosing CADP or FDR4?

We remember that the system specification is in this case provided in textual format, and this may sometimes appear less intuitive or user friendly than a drawing (at least for very small systems). Moreover, in order to be able to extract the full power of these tools, a certain degree of competence in theoretical aspects of formal verification is needed. Finally, the issue of translating a system specification into actually executable, understandable and verifiable code is not generally taken into considerations by these frameworks. In conclusion, these tools and approaches are particularly useful in case one needs a mathematically precise specification to be shared, verified and agreed, among the interested stakeholders, and when such a specification involves the presence of several independent interacting components.

4 Tool Usability Assessment

4.1 Methodology for Usability Evaluation

This section outlines the methodology adopted for the usability evaluation of the tools performed by railway experts from the SIRTI company. First, a set of models of the moving-block system was developed using 8 different formal and semi-formal tools, as explained in Sect. 2. The models were used to showcase the different tools at SIRTI and evaluate their *usability* from the point of view of railway experts.

A usability assessment in which the experts would directly interact with the tools was not considered reasonable, due to the skills required to master the tools, and to time constraints. Therefore, 3 researchers from CNR showed the different characteristics of the tools in a three-hours meeting with 9 railway experts from SIRTI, using the moving-block system as a case study. The researchers asked the railway experts to evaluate the usability of each tool based on their impression. The meeting was performed as follows:

1. **Introduction:** an introduction was given to the main moving-block components and principles. Part of the participants was already confident with the moving-block model developed within D2.1. This introduction served to provide all the participants a uniform perspective on the system that they were going to see modelled.
2. **Tool Showcase:** each of the 8 tools was presented live by a researcher in a 15 minutes presentation, covering the following aspects:
    - General structure of the tool: the presenter opens the tool, and provides a description of the graphical user interface (if available);
    - Elements of the model: the presenter opens the model, describes its architecture, and navigates the model;
    - Elements of the language: minimal description of the modelling language constructs, based on the model shown;
    - Simulation features: a guided simulation is performed (if supported by the tool);
    - Verification features: description of the language used for formal verification, and presentation of a formal verification session with counter-example (if supported by the tool).
3. **Usability Evaluation:** after the presentation of each tool, a questionnaire is provided to perform the evaluation. The questionnaire is described below.
4. **Wrap-up and Discussion:** a general discussion is performed between the participants.

To evaluate the usability of the tool, we resort to use a widely adopted usability questionnaire, namely, the System Usability Scale (SUS), developed by Brooke [Bro96]. The questionnaire was submitted to railway experts, who evaluated the tools based on a brief showcase. Some questions from the original SUS need to

be tailored to the specific context of our evaluation. The adapted questionnaire submitted to railway experts is as follows:

Railway Experts SUS Questionnaire

1. I think that I would like to use this tool frequently.
2. I found the tool unnecessarily complex.
3. I thought the tool was easy to use.
4. I think that I would need the support of a technical person to be able to use this tool.
5. I found the various functions in this tool were well integrated.
6. I thought there was too much inconsistency in this tool.
7. I would imagine that most people with industrial railway background would learn to use this tool very quickly.
8. I imagine that the tool would be very cumbersome to use.
9. I imagine that I would feel very confident using the tool.
10. I imagine I would need to learn a lot of things before I could get going with this tool.

Each respondent was required to give an answer in a 5-points Likert Scale, where 0 = Completely Disagree; 1 = Partially Disagree; 2 = Undecided; 3 = Partially Agree; 4 = Agree.

Computing the SUS score: To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1,3,5,7,and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SUS.

Overall, the SUS Score varies between 0 and 100, with the following interpretations for the scores, based on the work of Bangor et al. [Ban08]:

100 = Best Imaginable
85 = Excellent
73 = Good
52 = OK
39 = Poor
25 = Worst Imaginable

4.2 Results and Discussion

This section reports the results of the usability evaluation activity, and discusses its main outcomes, based on statistical data extracted from the SUS Questionnaire, and based on the final discussion with the participants.

Figure 1 presents the results of the SUS questionnaire based on the answers of the railway experts. We see that the tools that were considered most usable are Simulink (SUS Score = 76.38), followed by SCADE (69.16). These are model-based development tools, with appealing and effective graphical interfaces, and powerful languages, and powerful simulation capabilities. However, these are also tools that have limited support for formal verification. Indeed, while they allow complex modelling, and include Simulink Design Verifier and SCADE Design Verifier as verification packages, these packages do not support verification of highly complex models.

The two model-based development tools are followed by three other tools with quite powerful graphical user interface, but supporting widely different capabilities, namely ProB (SUS Score = 62.2), UPPAAL (61.7) and UMC (57.2). ProB and UMC allow the user to model in textual form, but present the results of the simulation also in graphical form. Instead, the UPPAAL modelling language is entirely graphical, and presents a graphical simulation style that recalls the message sequence charts, which are well known by railway practitioners.

SPIN (SUS Score = 56.9), Atelier B (45.5) and nuXmv (36.6), with some differences, are considered among the least usable tools. Although SPIN is a command line tool, without a graphical user interface, its score are higher than Atelier B. This can be explained considering the following observations: (a) SPIN uses a modelling language that is very similar to the C language, and therefore was considered familiar by the participants, who, in turn, gave higher scores; (b) Atelier B uses a refinement-based approach, which requires advanced skills to be understood and mastered, and it is not intuitive for railway practitioners.

When computing the average SUS Score, we obtain 58.22, which is between OK and Good. Hence, overall, the general usability of the presented tools can be considered acceptable. However, none of the tool was considered “Excellent” or “Best Imaginable” from the point of view of the railway experts. Although the presence of a powerful graphical user interface with simulation capabilities can be considered a key feature for the usability of the tools, as for Simulink and SCADE, it should be considered that a textual modelling language may be easier to edit and maintain. Further studies are needed to understand which are the specific peculiarities that would improve the usability of existing tools.

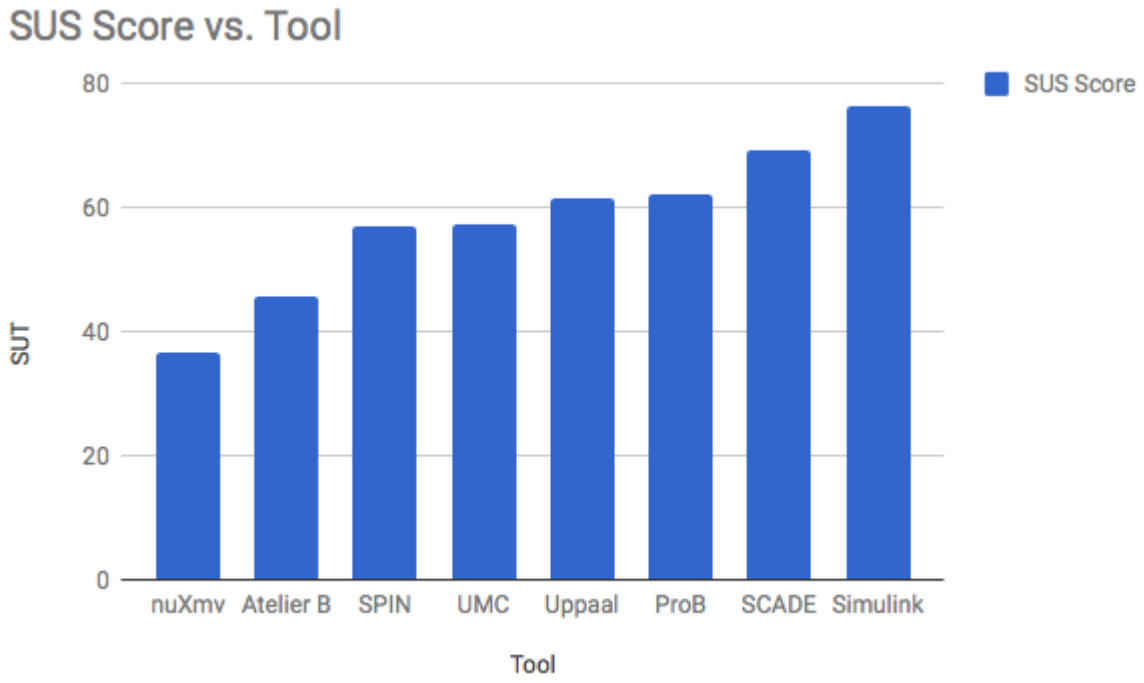

Figure 13 Results of the Usability Assessment

4.3 Threats to Validity

The results of this evaluation should be considered with care, as they suffer from a set of threats to validity, which are summarised in the following, according to the guidelines of Runeson and Höst for reporting case studies in software engineering [RH09].

Construct Validity. The tool adopted to evaluate usability, i.e., the SUS questionnaire, is widely used and has been proven effective in several works [Ban08]. On the other hand, the presented usability evaluation is

based on a showcase of tools performed by researchers, and hence the railway experts did not actually use the tools. Usability is therefore evaluated in an indirect manner, and different results may be obtained with a direct evaluation by railway experts.

Internal Validity. The tools were showcased by researchers who could have biased the audience towards a certain tool, based on their preferences. To mitigate this threat, before the evaluation the researchers rehearsed the tool showcase, with the participation of an external researchers, provided mutual recommendations, and established a time limit for each tool. Therefore, we argue that the tools were presented in a quite uniform manner. An exception is SCADE, which was presented through a video, instead of a live presentation, due licensing rights. Another issue is related to the number of subjects involved, i.e., 9, which may be regarded as limited sample size. However, it has been shown that a group of 10 users can identify 80% of the usability problems [HS10]. Therefore, we argue that, although limited, the sample size is in line with the samples normally used in usability testing.

External Validity. The evaluation involved railway experts with different roles, i.e., designers, developers, testers and managers, and with different degrees of experience in railways, although generally more than 10 years. This covers a large spectrum of perspectives. On the other hand, the participants were all from the same railway company, and this may limit the results to the specific peculiarity of the company. However, it is worth noticing that railway companies follow standardised processes, and, in addition, railway products are also well-established and even standardised, in many cases (e.g., CBTC, ERTMS). Therefore, we argue that the perspectives of different railway companies may be similar to each other, thus suggesting some degree of external validity of our results.

5 Moving Block Requirements Consolidation and Refinement

5.1 Methodology for Requirements Consolidation

This section describes the process followed for requirements consolidation, which has followed three parallel tasks:

- Automated Requirements Analysis: natural language processing (NLP) techniques have been used to analyse the requirements and automatically identify defects;
- Careful review of the model presented in D2.1, also with help of modelling tools, and its impact on requirements: this task consisted on visual inspection, modelling and brainstorming among the participants to incrementally consolidate the requirements towards a final version;
- Comparison with Hazard Analysis results as reported in Deliverable D2.2: this task consisted in the inspection of the hazard analysis entries to find aspects that were not considered in the requirements.

5.2 Automated Requirements Analysis

This section describes the activity performed to automatically analyse the requirements by means of NLP techniques.

5.2.1 NLP techniques for Requirements Analysis and QuARS.

Requirements are an abstract description of the system needs that are often open to different interpretations [FDE17]. This openness is emphasized by the use of Natural Language (NL), which is intrinsically ambiguous, even though it is commonly used to express requirements. Indeed, NL is the most widely used communication code, since it easily supports the exchange of knowledge among different stakeholders with heterogeneous backgrounds and skills. As the requirements process progresses, requirements are expected to be sufficiently clear to be interpreted in an unequivocal way by the interested stakeholders [FDE17].

A solution found within the RE community is to employ NLP tools that make the editors aware of the ambiguity in their requirements. Ambiguities normally cause inconsistencies between the expectation of the customer and the product developed, and possibly lead to undesirable reworks on the artifacts.

QuARS was introduced as an automatic analyzer of requirement documents [GLT05]. QuARS performs an initial parsing of NL requirements for automatic detection of potential linguistic defects that can determine ambiguity problems impacting the following development stages. QuARS performs a linguistic analysis of a requirements document in plain text format and points out the sentences that are defective according to the expressiveness quality model described in [BBG06]. The defect identification process is split in two parts: (i) the "lexical analysis" capturing optionality, subjectivity, vagueness, multiplicity and weakness defects, by identifying candidate defective words that are identified into a corresponding set of dictionaries; and (ii) the "syntactical analysis" capturing implicitness and under-specification defects. In the same way, detected defects may however be false defects.

5.2.2 Results obtained with the analysis of the initial requirements

Applying QUARS to the requirements listed in Sect. 2.1.1, after an inspection to remove false positives (detected defects that can actually be interpreted without any ambiguity), we remain with the following defects:

---------- QuARS [Lexical] multiplicity ANALYSIS ----------
The requirement:

each phase is independent from the others, and each phase does not suspend itself based on the status of the following one

contains a multiple sentence: more than one subject

---------- QuARS [Lexical] weakness ANALYSIS ----------
The requirement:

the buffer can be overwritten by the phase which writes on the buffer

is defective because it contains the wording: *can*

The first defective requirement exhibits two sentences, the first of which is actually redundant, since it does not give useful information on the actual operational meaning of the phases.

The second defective requirement is an ambiguous sentence that does not specify whether the buffer must be overwritten, or under some (unspecified condition) it might be not.

In both cases, it has been recognised that these sentences have been no more considered as functional requirements, and have been moved to a specific section of the requirement document, devoted to “Architectural Notes”.

5.2.3 Analysis of the final requirements

The analysis has been repeated on the revised requirements, before their final consolidation, aiding the correction of the precise expression of requirements. The consolidated requirements presented in Appendix A have undergone a final analysis with QuARS that, after ignoring false positives, has revealed no residual defects.

5.3 Moving-block Requirements and Model Refinement

While the analysis reported in Sect. 4.2 has regarded the requirement document itself, the other two tasks were conducted considering both the requirements document and the models derived from them. Hence the

consolidation of requirements has been complemented by a parallel refinement of the moving-block model, with the aim of cross-checking their correspondence. The parallel refinement of both artifacts has proceeded following two main lines:

- Careful review of the model presented in D2.1
- Comparison with Hazard Analysis results.

The main issues revealed by the review of the model presented in D2.1 were related to the:

- Possibility of autonomous generation by OBU of an MA_req, possibility not initially considered in D2.1, for compatibility with ERTMS L2. After discussion, it was decided not to include this feature for the sake of simplicity;
- Separation of concerns between OBU/LU and RBC;
- More details of timing of communications.

The comparison with the Hazard Analysis (Deliverable D2.2, Annex A) has shown that most of the requirements introduced for hazard mitigation were taken into account in the final moving-block requirements, except for the following entries:

Entry OBU-LU-6

Description: «LU is unable to send position information to the OBU, and OBU doesn't generate an alarm»
Requirement: «Communication between LU and OBU must be safe and continuously supervised, if the connection is lost an alarm must be triggered.»

The D2.1 model did not directly enforce this requirement, and a correction has been introduced at this regard.

Entries OBU-TI-1, OBU- TI-2, OBU- TI-3, OBU- TI-4, RBC- TI-1, RBC- TI-2

These hazards pose requirements over the safe communication between OBU and the Train Integrity Monitor (TIM), and between the RBC and OBU regarding Train Integrity information.

However, neglecting the aspects of Train Integrity is consistent with the developed model, which is focused on the moving-block and the localization aspects related to GNSS. The aspects of communication with the TIM module could also be considered (in the same way as those with the LU are considered), but it is important to underline that the safety aspects connected to Train Integrity have not been examined in ASTRail.

5.4 Final Requirements

The final Requirements Document for the moving block system resulting from the iteration described in this section is reported in Appendix A.

The main differences with respect to the requirements expressed in D2.1 can be summarized as:

1) The Position Report is sent every 5 seconds;
2) The Position Report is sent only if the position information is more recent than 1 seconds;
3) OBU replies with an ACK message to any received MA;
4) If the ACK message is not received by RBC, RBC repeats three times the sending, each sending after X seconds the previous one;
5) FATAL ERROR is declared if a OBU or RBC cycle takes more than 500 ms.
6) Train BRAKE is commanded when MA is expired (even if the same MA continues to arrive)
7) Initial train movement does not start until an initial MA is received.
8) It is made explicit that OBU and RBC cycles may be not synchronous, even if they have the same period.

Position Report requirements, 1) and 2), concern real-time needs for the integrity of the train position information.

Requirements 3) and 4) have been introduced to complete the communication from OBU and RBC with vitality and feedback messages, focused on closing the ring for the exchanged messages.

Strict real-time processing requirements are introduced with 5), meaning that we have to manage full deterministic elaboration processes for OBU and RBC. A fail-safe defence programming technique is detailed with addiction of 6). Observation 7) is introduced to clarify initial conditions, and 8) is used to clarify that no synchronisation exist among the considered components.

The final requirements reported in Appendix A have been also modelled in UML, using the UMC tool. Interpretations and simplifications were applied when considered appropriate. The model, together with its graphical UML representation, is available in the folder "UML-WP4-revised-model" at https://goo.gl/rcdVm2.

6 Conclusion

This deliverable reports the results of the preliminary trial on formal/semi-formal methods and tools, on the basis of the moving-block system described in Deliverable D2.1. The objective of this deliverable is twofold: firstly, we aim to model the initial design of the moving-block system with different formal/semi-formal tools, to evaluate their usability and their specific peculiarities; secondly, we want to refine and consolidate the initial requirements of the moving-block system to provide a stable version of such requirements.

To address the first objective, we select a set of formal methods and tools, which will be subject to the trial, based on the results of the analysis of the state-of-the-art and state-of-the-practice presented in Deliverable D4.1. Eight tools in total are selected, namely **Simulink, SCADE, UPPAAL, NuSMV, SPIN, ProB, Atelier B, UMC**. Each tool is used to develop a model of the moving-block system. The models are showcased to 9 industrial railway experts, and the widely adopted system usability scale (SUS) questionnaire is used to assess the usability of the tools. The results show that commercial tools with powerful user interface, such as Simulink and SCADE, are considered to be the most usable by the railway experts.

However, besides usability, other factors need to be taken into account when selecting a tool, and each tool is appropriate for a specific context:

- Simulink and SCADE are appropriate for both early prototyping and detailed design towards code generation, other tools need to be used when aiming at formal verification.
- UMC is appropriate for initial prototyping, when one wants to adopt a design based on UML state machines to facilitate communication with different stakeholders, but wants also verification capabilities as the ones provided by UMC.
- UPPAAL is appropriate when one needs to focus on the verification quantitative, real-time properties and probabilistic aspects.
- NuSMV and SPIN are appropriate when the system, or composition of systems, has a large state space, and one needs to verify temporal logic properties.
- Atelier B and ProB are the right choice for top-down development (i.e., from initial design to code) of single systems, and have somewhat complementary verification capabilities, with Atelier B supporting invariants checking, and ProB supporting model checking.

To address the second goal, which is refining and consolidating the initial moving-block requirements, we rely on an automated quality analysis based on natural language processing (NLP) technologies, and on iterations of brainstorming among industrial and academic partners. We present the final requirements in Appendix A.

Acronyms

Acronym	Explanation
OBU	On Board Unit
RBC	Radio Block Center
LU	Location Unit
MA	Movement Authority
PR	Position Report
MA_Req	Movement Authority Request
IP	Information Point
ACK	Acknowledge
LRBG	Last Relevant Balise Group (IP)
BC	Braking Curve
LR	Location Request

List of figures

References

[Ban08] Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An empirical evaluation of the system usability scale. Intl. Journal of Human–Computer Interaction, 24(6), 574-594.

[Bro96] Brooke, J. (1996). SUS: a" quick and dirty" usability scale in PW Jordan, B. Thomas, BA Weerdmeester, & AL McClelland. Usability Evaluation in Industry.

[CLA96] Clarke, E. M., & Wing, J. M. (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, *28*(4), 626-643.

[BOC09] Boca, P., Bowen, J. P., & Siddiqi, J. (Eds.). (2009). *Formal methods: State of the art and new directions*. Springer Science & Business Media.

[HS10] Hwang, W., & Salvendy, G. (2010). Number of people required for usability evaluation: the 10±2 rule. *Communications of the ACM*, *53*(5), 130-133.

[BBG06] Berry, D. M., Bucchiarone, A., Gnesi, S., Lami, G., & Trentanni, G. (2006). A new quality model for natural language requirements specifications. In *Proceedings of the International Workshop on Requirements Engineering: Foundation of Software Quality, REFSQ* (pp. 115–128).

[FDE17] Ferrari, A., Dell'Orletta, F., Esuli, A., Gervasi, V., & Gnesi, S. (2017). Natural language requirements processing: a 4D vision. *IEEE Software*, *34*(6), 28-35.

[GLT05] Gnesi, S., Lami, G., & Trentanni, G. (2005). An automatic tool for the analysis of natural language requirements. *Comput. Syst. Sci. Eng. 20*(1).

[BBF18] Basile, D., ter Beek, M. H., Fantechi, A., Gnesi, S., Mazzanti, F., Piattino, A., Trentini, D. & Ferrari, A. (2018). On the industrial uptake of formal methods in the railway domain. In *International Conference on Integrated Formal Methods* (pp. 20-29). Springer, Cham.

[RH09] Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, *14*(2), 131.

D4.2 Appendix A – Moving-block System Functional Requirements Specification (FRS)

Deliverable ID	**D4.2 Appendix A**
Title	**Moving-block System Functional Requirements Specification**
Work Package	**WP4**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**2018-11-27**
Status	**Final Version for Review**
Lead Editor	**SIRTI**
Main Contributors	**SIRTI, CNR**

Published by the ASTRAIL Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 777561
Call identifier: H2020-S2RJU-2017 | Topic: S2R-OC-IP2-01-2017 – Operational conditions of the signalling and automation systems; signalling system hazard analysis and GNSS SIS characterization along with Formal Method application in railway field

Table of Contents

1 Functional Requirements Specification (FRS) for the Moving-block System

1.1 Acronyms

OBU	On Board Unit
RBC	Radio Block Center
LU	Location Unit
MA	Movement Authority
PR	Position Report
MA_Req	Movement Authority Request
IP	Information Point
ACK	Acknowledge
LRBG	Last Relevant Balise Group (IP)
BC	Braking Curve
LR	Location Request

1.2 Architectural notes

The system shall be composed of three components:

- On-board Unit (OBU)
- Location Unit (LU)
- Radio Block Center (RBC)

The OBU and LU components communicate through a bi-directional channel.

The OBU and RBC components communicate through a bi-directional channel.

Each component is structured into phases. Each phase is independent from the others, and each phase does not suspend itself based on the status of the following one. Each phase produces information to be used by the following phase. The information is stored in a buffer of SIZE 1, for each phase. The buffer can be overwritten by the phase that writes on the buffer.

Maximum roundtrip delay for a communication between OBU and RBC (sending a request and receiving a reply) is **1.8 s**.[1]

A position report (PR) is sent by OBU every **5 s**. The OBU has a connection timeout with RBC equal to **10 s** (7 s in Italy).

The Movement Authority (MA) always refers to the distance that the train is allowed to run, starting from the Last Relevant Balise Group (LRBG).

Upon receiving a new MA, a new Braking Curve (BC) is computed.

Before entering the braking phase, the train can ask RBC for a new Movement Authority (MA_Req) that potentially extends the target point forward (and skipping the use of brakes).[2]

[1] Experimental data from HSL in Italy
[2] For simplicity, we do not consider the case of MA_Req

1.2.1 Location Unit

The Location Unit receives *location request* (LR) from the OBU and replies with messages containing the location of the train. The calculation of the current position of the train is a critical task: a ***fatal error*** must be raised in case of fault.

1.2.2 OBU

The On Board Unit periodically requires position data to the LU (LR) and forward it to the RBC every **5 s**. The OBU receives the MA from the RBC and computes the braking curve (BC), based on distance indicated in the MA and on current train position.

The OBU internal cycle is about **500 ms**:

1. Compute position
2. Send PR every **5 s**
3. RBC connection timeout check (**10 s**):
   - TO expired: *start braking, go to 6*;
   - TO not expired: *continue to next step*;
4. If a new MA has been received:
   - Reset RBC connection timeout;
   - Send ACK to RBC
   - Compute new BC;
5. BC control vs MA extension:
   - MA not ended: *repeat from 1*;
   - MA ended: *start braking, go to 6*;
6. Brake

1.2.3 RBC

The RBC reacts to reception of PR or MA_Req messages by calculating the route and generating the MA.

The RBC internal cycle is about **500 ms**:

1. Wait for PR from OBU
2. Route calculation (clearance check)
3. MA send (if the MA is equal to the last one, it could not be sent)
4. If sent, wait for MA ACK (timeout **1 s**):
   - TO expired < 3: *pause 'x' s, repeat from 3;*
5. Repeat from 1

1.3 Functional Requirements

Safety

SFT_01	The communication protocol between RBC and OBU must guarantee peer authentication, message integrity and message sequencing.
SFT_02	OBU braking curve calculation and train emergency brake must be performed with stated precision and time margins, able to guarantee the safety of the train.
SFT_03	RBC must calculate the MA with a predetermined precision, able to guarantee the safety of the train.
SFT_04	RBC must calculate the MA within a single machine cycle.
SFT_05	The computation of the current train position must be performed with high Safety Integrity Level (SIL4).
SFT_06	LU must compute the train position within a single machine cycle.
SFT_07	If the LU is not able to provide the current position, a ***fatal error*** must be raised.
SFT_08	If OBU cannot complete the processing of all the messages received in one cycle, it shall rise a ***fatal error*** and stop the train.
SFT_09	If RBC cannot complete the processing of all the messages received in one cycle, it shall rise a ***fatal error*** and shutdown.

General

GEN_01	No pre-emption of any cycle is allowed for LU, OBU and RBC. Before processing new messages or events, the computation cycle shall always be concluded.
GEN_02	OBU cycle and RBC cycle shall refer to a common time base; however, this does not imply that these cycles are synchronized.
GEN_03	In case of multiple messages of the same type from the same sender, the least recent one shall be deleted.
GEN_04	The MA always refers to the distance that the train is allowed to run, starting from the last IP.
GEN_05	When a train initiates its trip for the first time, the OBU shall require a MA to the RBC.
GEN_06	RBC and OBU send just one message per cycle.

Location Unit

CL_01	When the LU receives a LR, LU shall compute the location without additional delay.
SL_01	After computing the location, the LU shall send the location to OBU.

On Board Unit

GR_01	Every 500 ms OBU shall send a LR to LU.
SR_01	The OBU sends the LR to the LU (anytime a PR is required, see GR_01) without additional delay.
SLRBC_01	OBU shall send a PR to RBC every **5 s** (regardless of passing over a balise).[3]
SLRBC_02	Any PR must contain a position not older than **1 s** (older positions must be dropped),[4] based on internal clock.
SLRBC_03	PR must contain timestamps, based on internal clock.
RMA_01	When a MA is received, the connection timeout is reset.
RMA_02	If OBU does not receive a new MA within **10 s** from the reception of the last MA, the OBU shall **stop the train**.
RMA_03	Upon receiving a new MA, a new BC is computed by OBU.
SARBC_01	Upon receiving a new MA, an ACK message must be sent to RBC.

Radio Block Center

CMA_01	When RBC receives the PR, RBC shall check the reported position with respect to current train MA.
CMA_02	RBC sends the MA only as a reply to a PR.
CMA_03	RBC shall process in parallel messages coming from all the trains under its control.
CMA_04	Only the PR with the most recent timestamp must be processed.
SMA_01	After computing the MA, RBC shall send the MA to the OBU.

[3] We do not consider Virtual Balise as a trigger for sending PRs. This is more related to the RBC policy for generating the MAs.
[4] These are quite different respect to the Table 8 of D2.1 - UML State Machine for Moving Block - where PR are continuously generated and sent to the RBC.

SMA_02	If no ACK is received from RBC within **1 s**, the MA is sent again, up to 3 times at regular intervals of 'x' s.

2 Recall from D2.1

Moving Block has been modelled using UML State Machine Diagrams described in Methodology. It has been considered that moving block is constituted by several regions, each of them models a function or process performed in the system. Functions can be classified in two groups depending on the nature of the process. Some function aim sharing data and information between units, however some other functions process the data and lead to some calculations.

In Table 1 the regions identified are listed and the pseudo states that can be found in each region as well as a brief description of the modelled function within the region. Subsequently in Table 2 can be found the respective diagrams for each function and region as well as the events that trigger these function and transitions from one pseudo state to the next one.

ID	Region	Pseudostate	Description
OBU 1	Generation of location request	Requiring location	Every fixed interval of time the On-board Unit generates a request of its location.
TCOM 2	OBU sends location request to Location Unit	Empty	The On-board Unit sends the location request to the Location Unit.
		Full	
LU 3	Processing location request and calculating location by Location Unit	Idle	Once the Location Unit has received the location request, it processes it and calculates the location.
		Busy	
TCOM 4	Sending location from Location Unit to on-board Unit	Empty	The Location Unit sends location to On-board Unit.
		Full	
RCOM 5	Sending location from on-board unit to RBC	Empty	Once the On-board Unit receives its location it sends it to RCB.
		Full	
RBC 6	Processing information and calculation of movement authorities by RBC	Idle	Once the RCB receives the location of the trains it processes the information and calculates Movement Authorities.
		Busy	
RCOM 7	Sending movement authority to train	Empty	RBC sends Movement Authorities to On-board Units.
		Busy	
CON 8	Controlling	Counting	On board Unit controls the reliability of the MA and activates the emergency stop when the MA available becomes too old.
		Stopped	

Table 1 - Summary of regions and pseudostates modelled

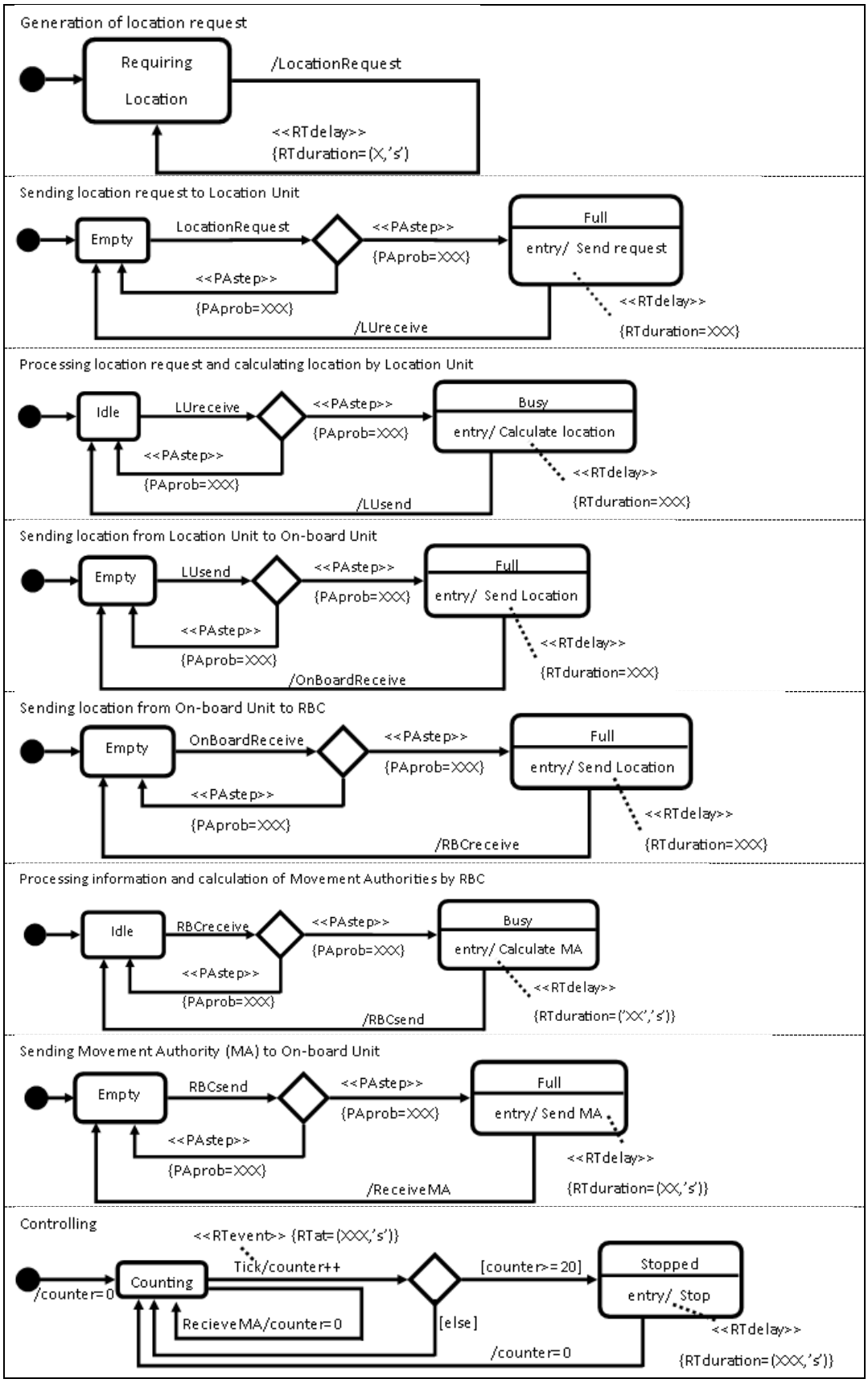

Table 2 - UML State Machine Diagrams for Moving Block

Within the processes described in the Table 7 the MA is extended automatically to the train, it shall be noted that when the train initiates its trip for the first time On-board Unit shall request the MA to the RBC. Nevertheless, the objective is to run in FS whenever possible, and the system must be designed to achieve this at the earliest opportunity. The train will automatically receive new ERTMS MAs as required, as long as it is safe to provide them.