Formal Distributed Protocol Development for Reservation of Railway Sections

Paulius Stankaitis[1(✉)], Alexei Iliasov[1], Tsutomu Kobayashi[2], Yamine Aït-Ameur[3], Fuyuki Ishikawa[2], and Alexander Romanovsky[1]

[1] School of Computing, Newcastle University, Newcastle upon Tyne, UK
p.stankaitis@newcastle.ac.uk
[2] National Institute of Informatics, Tokyo, Japan
[3] INPT–ENSEEIHT, 2 Rue Charles Camichel, Toulouse, France

Abstract. The decentralisation of railway signalling systems has the potential to increase railway network capacity, availability and reduce maintenance costs. Given the safety-critical nature of railway signalling and the complexity of novel distributed signalling solutions, their safety should be guaranteed by using thorough system validation methods. In this paper, we present a rigorous formal development and verification of a distributed protocol for reservation of railway sections, which we believe could deliver benefits of a decentralised signalling while ensuring safety and liveness properties. For the formal distributed protocol development and verification, we devised a multifaceted framework, which aims to reduce modelling and verification effort, while still providing complementary techniques to study protocol from all relevant perspectives.

Keywords: Formal verification · Distributed resource allocation · Performance analysis · Event-B · PRISM model checker · Railway signalling

1 Introduction

Railway signalling is a safety-critical system whose responsibility is to guarantee a safe and efficient operation of railway networks. In recent decades there have been proposals to utilize distributed system concepts (e.g. [13,24]) in railway signalling as a way to increase railway network capacity and reduce maintenance costs. These emerging distributed railway signalling concepts propose using a radio-based communication technology to decentralise contemporaneous signalling systems[1]. Because of their complex concurrent behaviour, distributed systems are notoriously difficult to validate and this could curtail the development and deployment of novel distributed signalling solutions.

In recent years there has been a push (e.g. [12,22]) by the industry with a strong focus on distributed systems to incorporate formal methods into their

[1] A single signalling computer may be responsible for controlling tens of routes (case studies [18,20]) whereas novel distributed systems would reduce that number.

system development processes to improve system assurance and time-to-market. Yet, despite that for years the railway domain has proved to be a fruitful area for applying various formal methods [3,7], considerably less has been done in applying them for distributed railway systems by industry and academia. Therefore, the long-term aim of our research is to lower the effort the barriers to applying formal methods in developing correct-by-construction distributed signalling systems.

In order to manage the modelling and verification complexity of distributed protocols we are working towards an integrated multifaceted methodology, which is based on three concepts: stepwise renement, communication modelling patterns and validation through proofs. In spite of advancements in proof automation, it might be too onerous to mathematically prove the model in early development stages. Therefore, it is also desirable that the framework should support model animation and scenario validation. It is also paramount that the framework should support quantitative evaluation; as stated by Fantechi and Haxthausen [10], distributed signalling solutions will only be adopted in practice if system availability is demonstrated. The authors (as discussed in [10]) of related researches did not consider liveness and fairness properties, which directly affect system availability. In our proposed multifaceted methodology we integrate stochastic simulators for quantitative analysis.

In this paper, we present a research, which uses the proposed methodology to formally develop and verify a distributed railway signalling protocol, which would deliver decentralised signalling benefits, while meeting high safety requirements. The developed distributed signalling protocol is based on serialisability and is inspired by protocols used in transactions processing [4,8,11] in centralised and distributed database systems. The main objective of our protocol is to guarantee mutual exclusion of railway sections while ensuring systems liveness. In a nutshell, our key contributions are the formally proved distributed railway section allocation protocol inspired by past protocols for database systems and the formalisation of the multifaceted verification framework.

Related Work. In Fantechi and Haxthausen [10] the authors formalise the railway interlocking problem as a distributed mutual exclusion problem and discuss the related literature on distributed interlocking (e.g. [9,13,24]). In principle all railway models share similar high-level safety, liveness and fairness requirements, as summarised on page 2 in [10]. One difference between our work and the studies overviewed in [10] is the interlocking engineering concept and the system model (e.g. allowed message delays). Another difference is the formal consideration of liveness and fairness requirements. In our work we not only prove the safety properties of the protocol, but also ensure systems liveness, fairness and analyse performance.

A similar distributed signalling concept is presented as a case study in [1]. The authors verified their system design via a simulation approach and only considered scenarios with up to two trains. In our verification approach we prove the distributed signalling system mathematically and hence guarantee its safety for any number of trains. In the paper by Morley [21] the author formally proved a distributed protocol, which is used in the real-world railway signalling systems to reserve a route, which is jointly controlled by adjacent signalling systems. Even though, the distributed signalling concepts of our works are different, the effects of message delays to the safety were considered in both works.

The rest of the paper is organised as follows. Section 2 outlines the motivation for developing the protocol, semi-formally describes its functionality, elicits the requirements and introduces its specifications and the properties to be proved. Section 3 further discusses the integrated methodology we are proposing. The following section briefly discusses formal model development and also provides technical details on property verication and performance analysis. In the last section we summarise our work and discuss future work directions.

2 Distributed Resource Allocation Model and Protocol

The distributed railway signalling can increase networks capacity (as trains could run closer), improve systems agility to delays and possibly reduce repair costs. On the other hand, an increased system complexity and a safety-critical (SIL4) nature requires the highest level of safety assurance. In order to apply formal methods one must clearly state system requirements and specifications. In the following subsections we describe an abstract model of the distributed railway system and its requirements as well as the stage_1 of the distributed protocol, which guarantees the safety and liveness of the distributed system.

2.1 High-Level Distributed System Model and Requirements

We abstract the railway model and instead of trains, routes and switches our system model consists of agents and resources (resources controllers). The system model permits message exchanges only between agents and resources, and messages can be delayed. Each resource controller has an associated queue-like memory, where agents allocation order can be stored. A resource also has a promise (ppt) and read pointers (rpt), which respectively indicate the currently available slot in the queue and the reserved slot (with an associated agent) that currently uses the resource. An agent has an objective, which is a collection of resources an agent will attempt to reserve (all at the same time) before using and eventually releasing them.

SAF_1 | A resource will not be allocated to different agents at the same time.

SAF_2 | An agent will not use a resource until all requested resources are allocated.

LIV_1 | An agent must be eventually allocated requested set of resources.

LIV_2 | Resource allocation must be guaranteed in the presence of message delays.

Requirements 1: High-level systems safety and liveness requirements

The main objective of the protocol is to enable safe and deadlock-free distributed atomic reservation of collection of resources. Where by a safe resource reservation we mean that no two different agents have reserved the same resource at the same time. The protocol must also guarantee that each agent eventually gets all requested resources - partial request satisfaction is not permitted. The main high-level safety and liveness requirements of the distributed system are expressed in Requirements 1.

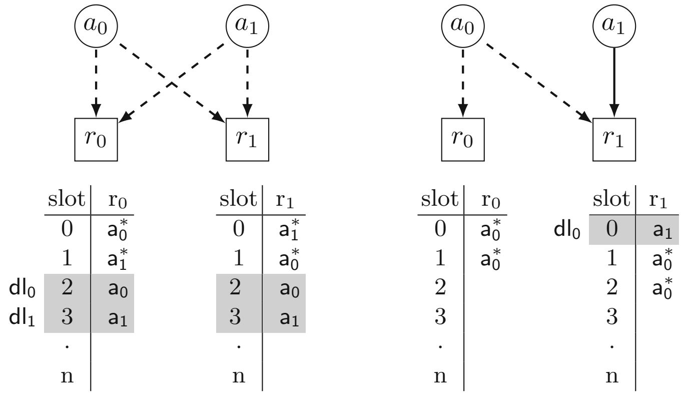
The following section attempts to justify the need for an adequate distribute protocol by discussing problematic distributed resource allocation scenarios.

2.2 Problematic Distributed Resource Allocation Scenarios

Let us consider Scenarios 1–2 (visualised in Fig. 1) to see how requirement LIV_1 could not be guaranteed (while ensuring SAF_2) without an adequate distributed resource allocation protocol.

Scenario 1. In this scenario, agents a_0 and a_1 are attempting to reserve the same set of resources $\{\mathsf{r}_0, \mathsf{r}_1\}$. Agents start by firstly sending request messages to both resources. Once a resource receives a request message, it replies with the current value of the promised pointer ($\mathsf{ppt}(\mathsf{r}_\mathsf{k})$) and then increments the $\mathsf{ppt}(\mathsf{r}_\mathsf{k})$. For instance, in this scenario, resource r_0 firstly received a request message from agent a_0 and thus replied with the value $\mathsf{ppt}(\mathsf{r}_0) = 0$, which was then followed by a message to a_1 with an incremented $\mathsf{ppt}(\mathsf{r}_0)$ value of 1. In Figure, we denote a_n^* as the $\mathsf{ppt}(\mathsf{r}_\mathsf{k})$ value sent to a_n. Request messages at resource r_1 have been received and replied in the opposite order.

In this preliminary protocol, after an agent receives promised pointer values from all requested resources, it sends messages to requested resources to lock them at the promised queue-slot. In this scenario, agent a_0 was promised queue-slots $\{(\mathsf{r}_0, 0), (\mathsf{r}_1, 1)\}$ while a_1 queue-slots $\{(\mathsf{r}_0, 1), (\mathsf{r}_1, 0)\}$. If agents would lock these exact queue-slots, resource r_0 would allow a_0 to *use* it first, while r_1 would concurrently allow a_1. The distributed system would deadlock and fail to satisfy LIV_2 requirement as both agents would wait for the second *use* message to ensure SAF_2.



	slot	r0
	0	a_0^*
	1	a_1^*
dl_0	2	a_0
dl_1	3	a_1
	.	
	n	

slot	r1
0	a_1^*
1	a_0^*
2	a_0
3	a_1
.	
n	

slot	r0
0	a_0^*
1	a_0^*
2	
3	
.	
n	

	slot	r1
dl_0	0	a_1
	1	a_0^*
	2	a_0^*
	3	
	.	
	n	

Fig. 1. Problematic scenarios: Scenario 1 (left) and Scenario 2 (right)

In order to prevent the cross-blocking type of deadlocks, an agent should repeatedly re-request the same set of resources (and not lock them) until all received promised queue slot values are the same. We define a process of an agent attempting to receive the same promised queue slots as an agent forming a distributed lane (dl).

A distributed lane of agent $\mathsf{a_n}$ is $\mathsf{dl(a_n)} = \{(\mathsf{r_k}, \mathsf{s}), (\mathsf{r_{k+1}}, \mathsf{s}), \dots, (\mathsf{r_{k+m}}, \mathsf{s})\}$, where $\{\mathsf{r_k}, \mathsf{r_{k+1}}, \dots, \mathsf{r_{k+m}}\}$ are all resources requested by agent $\mathsf{a_n}$ and s is the queue slot value promised by all requested resources. Important to note, that this solution relies on the assumption, that there is a non-zero probability of distinct messages arriving at the same destination in different orders, even if they are simultaneously sent by different sources.

The modified situation is depicted in Scenario 1, where, after agents $\{\mathsf{a_0}, \mathsf{a_1}\}$ initially receiving $\{(\mathsf{r_0}, 0), (\mathsf{r_1}, 1)\}$ and $\{(\mathsf{r_0}, 1), (\mathsf{r_1}, 0)\}$ slots, mutually re-request resources again. This time they receive $\{(\mathsf{r_0}, 2), (\mathsf{r_1}, 2)\}$ and $\{(\mathsf{r_0}, 3), (\mathsf{r_1}, 3)\}$ slots, and are able to form distributed lanes $\mathsf{dl_0(a_0)}$ and $\mathsf{dl_1(a_1)}$.

Scenario 2. However, simply re-requesting the same resources might result in a different problem. In Scenario 2, agent $\mathsf{a_1}$ has requested and has been allocated a single resource $\mathsf{r_1}$ which in turn modified $\mathsf{ppt(r_1)}$ to 1 while $\mathsf{ppt(r_0)}$ remained 0. If another agent $\mathsf{a_0}$ attempts to reserve resources $\{\mathsf{r_0}, \mathsf{r_1}\}$, it will never receive the same promised pointer values from both resources, and hence, will not be able to lock them.

To address the two issue described above, we developed a two-stage protocol, where the $\mathsf{stage_1}$ of the distributed protocol specifies how an agent forms a distributed lane. $\mathsf{Stage_2}$ of the protocol, which is out of this paper scope, addresses other deadlock scenarios, which can occur after agents form distributed lanes. In the following subsection we semi-formally describe the $\mathsf{stage_1}$ of the protocol.

2.3 Semi-formal Description of the $\mathsf{Stage_1}$

An agent, which intends to reserve a set of resources starts by sending $\mathsf{request}$ messages to resources. The messages are sent to those resources which are part of agents current objective. In the provided pseudocode excerpt, we first denote relations $\mathsf{sent\_requests}$ and $\mathsf{objective}$ where they are mappings from agents to resource collections (ln. 1–3 Algorithm 1). The messages $\mathsf{request}$ are sent by an agent $\mathsf{a_n}$ to a resource $\mathsf{r_k}$ ($\mathsf{r_k} \in \mathsf{objective[a_n]}$) until $\mathsf{sent\_requests[a_n]} = \mathsf{objective[a_n]}$ (images are equal). When a resource $\mathsf{r_k}$ receives a $\mathsf{request}$ message from an agent $\mathsf{a_n}$ it responds with a reply message which contains the current promised pointer value of resource $\mathsf{ppt(r_k)}$ to that agent and increments the promised pointer (ln. 2–4 Algorithm 2). After sending all $\mathsf{request}$ messages an agent waits until reply messages are received from requested resources and then makes a decision.

Algorithm 1 Agent stage_1 communication algorithm

```
1: while $\mathsf{sent\_requests}[a_n] \neq \mathsf{objective}[a_n]$ do
2:     $\mathsf{request}(a_n) \rightarrow r_k$   sending request message from agent $a_n$ to resource $r_k$
3: end
4: wait until $\mathsf{received\_replies}[a_n] = \mathsf{objective}[a_n]$
5: while $|\mathsf{replies}[a_n]| \neq 1$ do   cardinality of agents received slot indices
6:         $m' = \max(\mathsf{replies}[a_n]) + 1$
7:         $\mathsf{sent\_srequests}[a_n]' = \varnothing$   reset sent special request messages buffer
8:         $\mathsf{received\_replies}[a_n]' = \varnothing$   reset received reply messages buffer
9:             while $\mathsf{sent\_srequests}[a_n] \neq \mathsf{objective}[a_n]$ do
10:                $\mathsf{srequest}(a_n, m) \rightarrow r_k$
11:            end
12:        wait until $\mathsf{received\_replies}[a_n] = \mathsf{objective}[a_n]$
13: end
14: while $\mathsf{sent\_write}[a_n] \neq \mathsf{objective}[a_n]$ do
15:     $m' = \max(\mathsf{replies}[a_n])$
16:     $\mathsf{write}(a_n, m) \rightarrow r_k$
17: end The end of $\mathsf{stage}_1$ of the protocol.
```

When all received promised pointer values are the same (a distributed lane can be formed) an agent completes the stage_1 by sending write, to all requested resources, messages which contain the negotiated index (ln. 14–17 Algorithm 1). But if one of the received promised pointer values is different an agent will start a renegotiation cycle (ln 5–13 Algorithm 1). By sending a srequest messages which contain a desired slot index to resources. A desired index is computed by taking the maximum of all received promised pointer values and adding a constant (one is sufficient) - ln. 6 Algorithm 1. A resource will reply to srequest message with the higher value of the current $\mathsf{ppt}(r_k)$ or received srequest message value and will update the promised pointer (ln. 5–7 Algorithm 2). After sending all srequest messages, an agent waits for reply messages and then restarts the loop if received slot indices are not the same.

Algorithm 2 Resource stage_1 communication algorithm

```
1: switch received_message do
2:     case $\mathsf{request}(a_n)$
3:         $\mathsf{reply}(\mathsf{ppt}(r_k),\ r_k) \rightarrow a_n$
4:         $\mathsf{ppt}(r_k)' = \mathsf{ppt}(r_k) + 1$
5:     case $\mathsf{srequest}(a_n, n)$
6:         $\mathsf{reply}(\max(\mathsf{ppt}(r_k), n),\ r_k) \rightarrow a_n$
7:         $\mathsf{ppt}(r_k)' = \max(\mathsf{ppt}(r_k), n) + 1$
```

SAF_3 | An agent will not send write (form a distributed lane) messages until all receive promised pointer values are identical.

SAF_4 |Agents with overlapping resource objectives will negotiate distributed lanes with different index.

LIV_3 | An agent will eventually negotiate a distributed lane.

Requirements 2: Low-level protocol stage_1 safety and liveness requirements

It is important to note that the stage_1 protocol solution to the described deadlock scenarios has a stochastic nature and one needs to guarantee that a desirable state is probabilistically reachable. In Requirements 2 we summarise requirements for the stage_1 of the protocol.

After an agent completes stage_1 and thus negotiates a distributed lane it will start protocol stage_2 to prevent other deadlock scenarios. Predominantly because of papers verification focus towards properties from stage_1 (all complimentary verification/analysis techniques used) we provide protocol stage_2 description in the online appendix[2].

3 Multifaceted Modelling and Verification Framework

As stated before, the long-term objectives of our research are to reduce modelling and verification effort of distributed systems and to have a multifaceted framework to study protocols from all relevant perspectives. In the introduction, we defined key formal concepts the framework should rely on and in the following section we discussed protocol requirements we need to guarantee.

The following subsections proposes an engineering process with different formal techniques each of which is efficient to handle parts of above requirements and help to manage modelling and verification complexity.

3.1 Formalised Multifaceted Verification Framework

For any adequate formal system development, system requirements should be clearly stated, and so, this is the first step (Step 1 in Fig. 2) in the modelling process. Currently, we do not suggest or provide a specific structural approach for defining distributed system requirements. The next step (Step 2) in the process is developing and verifying a pivotal formal model. The purpose of formally modelling a distributed system is to have a formal artefact, which can be animated, analysed and formally verified.

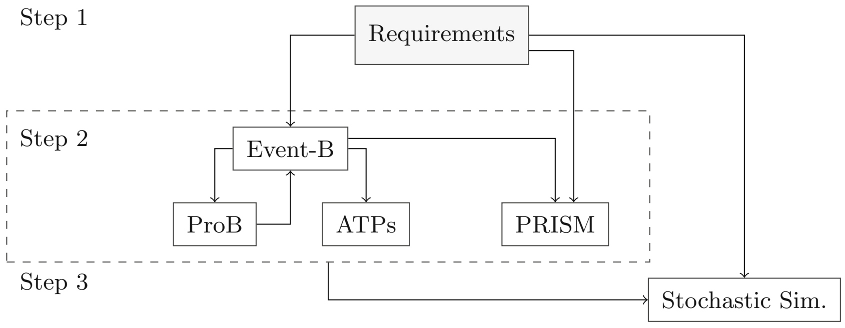


Fig. 2. Multifaceted modelling and verification framework

[2] A complete protocol description and formal models can be found at http://stankaitis.uk/2019/02/.

For the development and verification of pivotal functional system models we selected the Event-B [2] specification language, which has previously been successfully used for modelling and verification of various distributed protocols [5,15,16]. The Event-B method provides an expressive modelling language, flexible refinement mechanism and is also proof driven, meaning model correctness is demonstrated by generating and discharging proof obligations with available automated theorem provers [6,17]. The method is supported by tools such as ProB [19] which enable animating and model-checking a model. On the other hand, the Event-B method does not have an adequate probabilistic reasoning support, which, for example, was essential for verifying the distributed railway section reservation protocol. Therefore, it was decided to integrate the well-known PRISM [14] stochastic model checker into the framework, so stochastic system's properties can be verified.

The last step (Step 3) in the proposed engineering process is analysing a developed distributed system's performance. For that, we have implemented a high-fidelity protocol simulator which could help to evaluate protocols under normal or *stressed* conditions. Following subsections provide more detail on how each of the formal techniques would be used in the development and verification of a distributed protocol.

3.2 Step 2: Developing Functional Pivot Models in Event-B

A formal functional Event-B model can have a multitude of uses, but the main application is for formally proving properties about the distributed system. The completed distributed system's model in Step 2 should cover all requirements and specifications, and would be considered correct when all generated proof obligations are proved.

The model development approach we propose is a rather standard and starts with the abstract model which formally specifies the objective of the distributed protocol. In fact, distributed aspects of the system are ignored at this model level and the abstract model considers a centralised configuration. The abstract model is then iteratively refined by introducing more details about the distributed protocol, primarily by modelling communication aspects. To reduce modelling effort we previously developed communication modelling patterns and described a generic model refinement plan in [23]. A key aspect of our methodology is the scenario validation and analysis. Particularly, in early protocol development stages, it might be too onerous to verify a model only to discover design mistakes. To facilitate design exploration we apply animation and model-checking enabled by ProB. Nonetheless, the final (concrete) model should be proved by adding invariants to the model and proving generated proof obligations with available automated theorem provers.

3.3 Step 2: Proving Stochastic Properties with PRISM

As the distributed signalling protocol had a stochastic nature it was important to formally demonstrate that a satisfying state could be reached. Probabilistic or liveness properties are hard to formalise and prove in the Event-B method. Therefore, it was decided to prove progress of the protocol outside of Event-B by redeveloping part of the model (stage_1) in the PRISM model checker.

The drawback of using PRISM model checker, if a bounded problem abstraction cannot be found, the verification is limited to bounded models. As we could not find protocol's stage_1 abstraction, we created a skeleton model, which then could be instantiated to model specific scenarios of stage_1 with n agents, m resources and other initial conditions. Additionally, we developed a model generator, which can automatically instantiate the skeleton model to capture a random scenario and run probabilistic verification conditions.

3.4 Step 3: Analysing System's Performance

With Event-B and PRISM we aim to demonstrate that the protocol addresses the formulated requirements but it is necessary in our application domain to understand how the protocol is going to perform under various conditions if it were deployed in a real system. To conduct such a simulation we have implemented a high fidelity protocol simulator that can be populated with any number of resources and agents while realising any conceivable agents' goal formation and message delivery policies.

The simulator is parametrised with a function of probability of picking a certain message out of a pool of available messages. The probability function is itself parametrised by message source, destination, timestamp and type. The simulation would help to answer how fast, in terms of vital steps such as messages sent, a protocol's stage_1 can be completed and how the performance is affected by messages delays. With function D we can simulate slow agents and resources, fair, arbitrary and unfair delivery policies, agents that operate much faster than others and so on.

4 Formal Protocol Modelling, Verification and Analysis

In this section we present the application of previously introduced modeling and verification framework for developing distributed railway signalling protocol. In Sect. 2 we defined protocol's requirements (Step 1), thus following subsections focuses on formal methodology aspects.

4.1 Step 2. Formal Protocol Model Development in Event-B

We apply the Event-B formalism to develop a high-fidelity functional model and prove the protocol functional correctness requirements. We follow the modelling process presented in Sect. 3.2. Important to note that the protocol model was redeveloped multiple times as various deadlock scenarios were found with ProB animator and model-checker. Below, we overview the final (verified) model.

Modelling was started by creating an abstract model context which contains constants, given sets and uninterpreted functions. In the abstract context, we introduced three (finite) sets, to respectively represent agents (agt), resources (res) and objectives (obj). The context also contains an objective function which is a mapping from objectives to a collection of resources ($\mathsf{ob} \in \mathsf{obj} \rightarrow \mathbb{P}(\mathsf{res})$) and an enumerated set for agents status counter.

The dynamic protocol parts, such as messages exchanges, are modelled as variables and events computing next variable states and contained in a *machine*. According to the proposed model development process, the initial machine (abstract) should summarise the objective of protocol, which is an agent completing an objective (locking all necessary resources). To capture that, the abstract protocol machine contains two events, respectively modelling an agent locking and then releasing a free objective ($\mathsf{ob} \in \mathsf{obj}$). The abstract model is refined by mostly modelling communication aspects of the distributed signalling protocol and for that we use a backward unfolding style where the next refinement step introduces preceding protocol step. Below, we overview the refinement chain and properties we proved at that modelling stage.

Refinement 1 (Abstract ext.). In this refinement we introduce resources into the model and now an agent tries to fulfill the objective by locking resources. Previous two events (lock/release) are now decomposed to two for each and capture iterative locking and releasing of resources.

Refinement 2. The abstract models are firstly refined with stage_2 part of the protocol. In the refinement, r_2, we introduced lock, response and release messages and associated events into the model. In this step we also demonstrated that the protocol stage_2 ensures safe distributed resource reservation by proving an invariant. The invariant states that no two agents will be both at resource consuming stage if both requested intersecting collections of resources.

Refinement 3. Model r_3, is the bridge between protocol stages stage_1 and stage_2 and introduces two new messages write and pready into the model.

Refinement 4. The final refinement step - r_4 - models stage_1 of the distributed protocol which is responsible for creating distributed lanes. Remaining messages request, reply, srequest and associated events are introduced together with the distributed lane data structure. In this refinement we prove that distributed lanes are correctly formed (req. $\mathsf{SAF}_{3\text{-}4}$).

4.2 Step 2: Proving Functional Correctness Properties in Event-B

As shown in Sect. 2.2 (Scenarios 1 - 2) high-level system's requirements can only be met if an agent invariably and correctly forms a distributed lane. The probabilistic lane forming eventuality (LIV_3) is discussed separately while in the following paragraphs we focus on the proof regarding requirements $\mathsf{SAF}_{3\text{-}4}$.

SAF_3 is required to ensure that agent's resource objectives are not satisfied or satisfied on full. The model addresses this via event *guards* restricting enabling states of the event that generates an outgoing write message. To cross-check this implementation we add an invariant that directly shows that SAF_3 is maintained in the model. For illustrative purposes we focus on details of verifying a slightly more interesting case of SAF_4 and assume that SAF_3 is proven.

Requirement SAF_4 addresses potential cross-blocking deadlocks or resource double locking due to distributed lane overriding. The strategy is to prove the requirement is to show that agents that are interested in at least one common resource (related) always form distributed lanes with differing indices. We start by assuming that agents only form distributed lanes if all received indices are the same (proved as SAF_3). Then, if a resource (or resources) shared between any two related agents send unique promised pointer values to these agents, these indices will be distributed lane *deciders* as all other indices from different resources must be the same to form a distributed lane. Hence, to prove SAF_4 it is enough to show that each resource replies to a request or special request message with a unique promised pointer value.

```
resource_reply_general $\widehat{=}$
ANY
        rq, rp
WHERE
        grd$_1$   $\mathsf{rq} \in \mathsf{req}$   take a sent request message
        grd$_2$   $\mathsf{rp} \in \mathsf{REQ} \setminus \mathsf{rep}$   create a new reply message
        grd$_3$   $\mathsf{repd(rp)} = \mathsf{reqs(rq)}$ destination of reply message is source of request message
        grd$_4$   $\mathsf{reps(rp)} = \mathsf{reqd(rq)}$ source of reply message is destination of request message
        grd$_5$   $\mathsf{repn(rp)} = \mathsf{ppt(reps(rp))}$ reply message contains promised pointer
THEN
        act$_1$   $\mathsf{rep} := \mathsf{rep} \cup \{\mathsf{rp}\}$   add new message to reply channel
        act$_2$   $\mathsf{req} := \mathsf{req} \setminus \{\mathsf{rq}\}$   remove request message from request channel
        act$_3$   $\mathsf{ppt(res)} := \mathsf{ppt(res)} + 1$   increment promised pointer
        act$_4$   $\mathbf{his_{ppt}(res)} := \mathbf{his_{ppt}(res)} \mathbin{\lhd\!\!\!+} \{(\mathbf{his_{wr}(res)}) \mapsto \mathbf{ppt(res)}\}$
        act$_5$   $\mathbf{his_{wr}(res)} := \mathbf{his_{wr}(res)} + \mathbf{1}$
END
```

Fig. 3. Event-B model excerpt of a resource sending a reply message (Color figure online)

To prove that all resources replies to a request or special request message with a unique promised pointer value, we firstly introduced a history variable $\mathsf{his_{ppt}}$ of type $\mathsf{his_{ppt}} \in (\mathsf{res} \to (\mathbb{N} \nrightarrow \mathbb{N}))$ into our model. The main idea behind the history variable was to chronologically store the promised pointer values sent by a resource. We also introduced a time-stamp variable $\mathsf{his_{wr}}$ of the type $\mathsf{his_{wr}} \in \mathsf{res} \to \mathbb{N}$ to chronologically order the promised pointer values stored in the history variable.

After introducing history variables, we modified events resource_reply_general and resource_reply_special, which in the protocol update the promised pointer variables, by adding two new actions (see Fig. 3). The first action **act$_4$** updates the history variable with the promised pointer value ($\mathsf{ppt(res)}$) that was sent to the agent at the time stamp ($\mathsf{his_w r(res)}$). The second action, $\mathbf{act_5}$, simply increments resource's res time-stamp ($\mathsf{his_w r(res)}$) variable.

$$\textbf{inv\_saf\_4} \quad \forall \mathsf{r}, \mathsf{n_1}, \mathsf{n_2} \cdot \mathsf{r} \in \mathsf{RES} \land \mathsf{n_1}, \mathsf{n_2} \in \mathsf{dom}(\mathsf{his_{ppt}}(\mathsf{r})) \land \mathsf{n_1} < \mathsf{n_2} \Rightarrow \mathsf{his_{ppt}}(\mathsf{r})(\mathsf{n1}) < \mathsf{his_{ppt}}(\mathsf{r})(\mathsf{n2})$$

Action $\mathbf{act_4}$ updates a history variable for a resource res with the current write stamp and promised pointer ($\mathsf{ppt(res)}$) value sent. The next action $\mathbf{act_5}$ simply updates the resource's write stamp. We can then add the main invariant to prove (**inv_saf_4**) which states that if we take any two entries $\mathsf{n1}, \mathsf{n2}$ of the history variable for the same resource where one is larger, then that larger entry should have larger promised pointer value.

$$\begin{aligned} \textbf{inv\_his\_ppt} \quad \forall \mathsf{res} \cdot \quad & (\mathsf{his_{wr}}(\mathsf{res}) = 0 \land \mathsf{his_{ppt}}(\mathsf{res}) = \varnothing) \\ & \lor(\quad \mathsf{dom}(\mathsf{his_{ppt}}(\mathsf{res})) = 0 \mathinner{..} \mathsf{his_{wr}}(\mathsf{res}) - 1 \\ & \quad \land \mathsf{his_{ppt}}(\mathsf{res})(\mathsf{his_{wr}}(\mathsf{res}) - 1) = \mathsf{ppt}(\mathsf{res}) - 1) \end{aligned}$$

To prove that $\mathsf{resource\_reply\_\{general, special\}}$ preserve **inv_saf_4**, the following properties play the key role: (1) the domain of $\mathsf{his_{ppt}}$ (i.e., 'indices' of $\mathsf{his_{ppt}}$) is $\{0, \ldots, \mathsf{his_{wr}} - 1\}$, (2) $\mathsf{his_{ppt}}(\mathsf{his_{wr}} - 1) < \mathsf{his_{ppt}}(\mathsf{his_{wr}})$. Property (2) holds because $\mathsf{his_{ppt}}(\mathsf{his_{wr}})$ is the maximum of promised pointer (ppt) and special request slot number and promised pointer is incremented as $\mathsf{resource\_reply\_\{general, special\}}$ occurs. We also specified these properties as an invariant (**inv_his_ppt**) and proved they are preserved by the events which helped to prove **inv_saf_4**.

Proof Statistics. In Table 1 we provide an overall proof statistics of the Event-B protocol model which may be used as a metric for models complexity. The majority of the generated proof obligations were automatically discharged with available solvers and even a large fraction of interactive proofs required minimum number of steps. We believe that a high proof automation was due to modelling patterns [23] use and SMT-based verification support [6,17].

Table 1. Event-B protocol model proof statistics

Model	No. of POs	Aut. discharged	Int. discharged
context $\mathsf{c_0}$	0	0	0
context $\mathsf{mes.}$	9	9	0
machine $\mathsf{m_0}$	12	12	0
machine $\mathsf{m_1}$	23	21	2
machine $\mathsf{m_2}$	59	43	16
machine $\mathsf{m_3}$	43	32	11
machine $\mathsf{m_4}$	103	57	46
Total	249	174	75

4.3 Step 2: Proving Liveness (req. LIV_3) with PRISM

In this subsection, we discuss stochastic model checking results with which we intend to prove level that LIV_3 requirement is preserved. In particular, we focus on showing that LIV_3 requirement is ensured in Scenario 2 (Sect. 2.2).

In order to demonstrate that LIV_3 requirement holds in Scenario 2 (Sect. 2.2) we used stage_1 protocol's skeleton PRISM model to replicate Scenario 2. In this experiment we were interested in observing the effects a promised pointer offset has on an probability of agent forming a distributed lane while the upper limit of the promised pointer is increased[3] (n in Scenario 2). Early experiments showed that verification would not scale well (several hours for a single data-point) if we would increase the number of resources and agents above two resources and three agents (each agent trying to reserve both resources) so we kept these parameters constant.

For each scenario, we would run a quantitative property: $\mathsf{P}\ =?\ [\mathsf{F}\ \mathsf{dist}_0\ >\ \mathsf{-1}]$ which asks what is the probability of an agent negotiating a distributed lane until the upper promised pointer limit is reached. The three curves (red, green and violet) in Fig. 4 show the effect a promised pointer offset has on negotiation probability as queue depth is increased. Results suggest that increasing the offset reduces the probability of negotiating a distributed lane as queue depth is increased, but the probability still approaches one as the number of rounds is increased (Fig. 4).

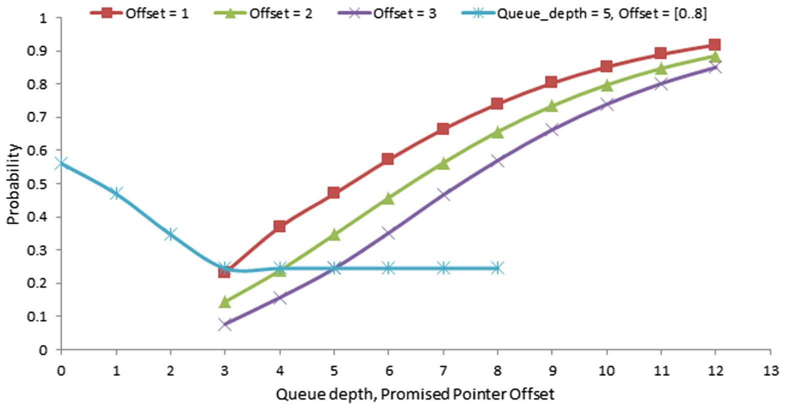


Fig. 4. Scenario 2 with varied resource promised pointer offset and queue depth.

To further see the effects of the offset, we considered a different experiment where the same quantitative property would be run when the number of possible renegotiations value is kept constant and offset is increased (light blue plot).

[3] Instead, of ppt upper limit we decided plot the probability against the queue depth, (*offset* - n) as it directly shows how many times an agent can renegotiate resources.

Results indicate that offset has only effect until a specific threshold and after that the probability of agent negotiating a distributed lane is not affected by the offset. These results suggest that the situation in Scenario 2 does not violate LIV_3 requirement as distributed lanes can be negotiated.

4.4 Step 3: Analysing Performance

The goal of this part is to study the protocol performance under various stress conditions and thus provide assurances of its applicability in real life situations. To build simulation, we simply capture protocol's stage_1 behaviour using a program. We are also able to obtain bounds on the number of messages required to form lanes in different setups. This can be directly translated into real-life time bounds on the basis of point to point transmission times.

Simulation Construction. Simulation is setup as a collection of actors of two types - agents and resources - and an orchestration component observing and recording message passing among the actors. A message is said to be in transit as soon as it is created by an actor. Every act of message receipt (and receipt only) advances the simulation (world) clock by one unit. Hence, any number of computations leading to message creation can occur in parallel but message delivery is sequential. To model delays we define a function that probabilistically picks a message to be delivered among all the messages currently in transit. A special message, called skip, is circulated to simulate idle passage of time. This message is resent immediately upon receipt by an implicit idle actor.

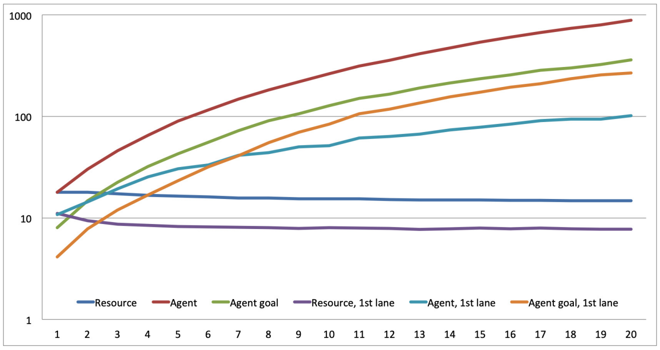


Fig. 5. Time to form all or first lanes, logarithmic scale.

Let $\mathbb{M}$ be set of all messages that can be generated by agents and resources. Also, let $\mathsf{skip} \notin \mathbb{M}$ denote the skip message and $\mathbb{M}' = \mathbb{M} \cup \{\mathsf{skip}\}$. By its structure, set $\mathbb{M}'$ is countable (each message identified by unique integer) and one can define a measure space over $\mathbb{M}'$. Let D signify the probability that some message $m \in M \subseteq \mathbb{M}'$ from message pool M is selected for reception. We shall define D via the current message pool, the attributes of m such its source, destination, time stamp and protocol stage, and the world time: $D = D(M, m, t) = D(M, m.s, m.d, m.c, m.o, t)$. Here M is the set of available message, $m.s$ and $m.d$ are the message source and destination agent or resource, $m.c$ is the message type (e.g., WRITE), $m.o$ is the message timestamp (the point of its creation) and t is the world clock. Defining differing probabilities D we are able to address most scenarios of interest.

Uniform Distribution. With $D(M, m, t) = \text{card}(M)^{-1}$ the simulator picks a message from M using a uniform distribution. It is an artificial setting as the time in transit bears no influence over the probability of arrival. Counter-intuitively, the said probability may decrease with the passage of time when new messages are created quicker than they are delivered. The skip message has equal probability with the rest so the system "speeds up" when M is large. The plots in Fig. 5 shows how the protocol performance changes when the number of resources (Resource line), agents (Agent lines), and resources an agent attempts to acquire (Agent goal) increase. We plot separately time to form all lanes and any first lane. The values plotted are averaged over 10000 runs.

5 Conclusions and Future Work

In this paper we proposed a multifaceted framework with which we aim to reduce modelling and verification of distributed (railway signalling) systems. The framework was applied in the development of the novel distributed signalling protocol. Starting only with high-level system requirements we developed an early formal protocol prototype which with the help of ProB was refined as subtle deadlock scenarios were discovered. This in part is the advantage of a stepwise development supported by Event-B as complex distributed models can be *decomposed* into smaller problems and errors found earlier. The stepwise distributed protocol development as also shown before [5,15,16] together with adequate tools [6,17] helped to achieve fairly high verification automation. On the other hand, protocol verification was complicated by the need of stochastic reasoning and not adequate Event-B support for reasoning about probabilistic properties. The current solution relied on a model redevelopment in stochastic model checker PRISM which did not scale well for verification of larger scenarios. As a future direction it is essential to address this problem by most likely improving stochastic reasoning in Event-B. In the future we would also like to a much closer tool integration and support an automatic translation to PRISM and the stochastic simulator.

References

1. INTO-CPS Project. Case Studies 2, Deliverable D1.2. Technical report, November 2016. http://projects.au.dk/fileadmin/D1.2a_Case_Studies.pdf
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, New York (2013)

3. Behm, P., Benoit, P., Faivre, A., Meynadier, J.-M.: Météor: a successful application of B in a large project. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) FM 1999. LNCS, vol. 1708, pp. 369–387. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48119-2_22
4. Bernstein, P.A., Shipman, D.W., Rothnie Jr., J.B.: Concurrency control in a System For Distributed Databases (SDD-1). ACM Trans. Database Syst. **5**(1), 18–51 (1980)
5. Cansell, D., Méry, D.: Formal and incremental construction of distributed algorithms: on the distributed reference counting algorithm. Theor. Comput. Sci. **364**(3), 318–337 (2006)
6. Déharbe, D., Fontaine, P., Guyot, Y., Voisin, L.: Integrating SMT solvers in rodin. Sci. Comput. Program. **94**(P2), 130–143 (2014)
7. Essamé, D., Dollé, D.: B in large-scale projects: the canarsie line CBTC experience. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 252–254. Springer, Heidelberg (2006). https://doi.org/10.1007/11955757_21
8. Eswaran, K.P., Gray, J., Lorie, R.A., Traiger, I.L.: The notions of consistency and predicate locks in a database system. Commun. ACM **19**(11), 624–633 (1976)
9. Fantechi, A., Haxthausen, A.E., Nielsen, M.B.R.: Model checking geographically distributed interlocking systems using UMC. In: 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 278–286, March 2017
10. Fantechi, A., Haxthausen, A.E.: Safety Interlocking as a distributed mutual exclusion problem. In: Howar, F., Barnat, J. (eds.) FMICS 2018. LNCS, vol. 11119, pp. 52–66. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00244-2_4
11. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco (1992)
12. Hawblitzel, C., et al.: IronFleet: proving safety and liveness of practical distributed systems. Commun. ACM **60**(7), 83–92 (2017)
13. Haxthausen, A.E., Peleska, J.: Formal development and verification of a distributed railway control system. IEEE Trans. Softw. Eng. **26**(8), 687–701 (2000)
14. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: a tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006). https://doi.org/10.1007/11691372_29
15. Hoang, T.S., Kuruma, H., Basin, D., Abrial, J.R.: Developing topology discovery in event-B. Sci. Comput. Program. **74**(11), 879–899 (2009)
16. Iliasov, A., Laibinis, L., Troubitsyna, E., Romanovsky, A.: Formal derivation of a distributed program in event B. In: Qin, S., Qiu, Z. (eds.) ICFEM 2011. LNCS, vol. 6991, pp. 420–436. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24559-6_29
17. Iliasov, A., Stankaitis, P., Adjepon-Yamoah, D., Romanovsky, A.: Rodin platform why3 plug-in. In: Butler, M., Schewe, K.-D., Mashkoor, A., Biro, M. (eds.) ABZ 2016. LNCS, vol. 9675, pp. 275–281. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33600-8_21
18. Iliasov, A., Taylor, D., Laibinis, L., Romanovsky, A.: Formal verification of signalling programs with SafeCap. In: Gallina, B., Skavhaug, A., Bitsch, F. (eds.) SAFECOMP 2018. LNCS, vol. 11093, pp. 91–106. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99130-6_7
19. Leuschel, M., Butler, M.: ProB: a model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 855–874. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45236-2_46

20. Limbrée, C., Cappart, Q., Pecheur, C., Tonetta, S.: Verification of railway interlocking - compositional approach with OCRA. In: Lecomte, T., Pinger, R., Romanovsky, A. (eds.) RSSRail 2016. LNCS, vol. 9707, pp. 134–149. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33951-1_10
21. Morley, M.: Safety assurance in interlocking design. Ph.D. thesis, University of Edinburgh, College of Science and Engineering, School of Informatics (1996)
22. Newcombe, C.: Why Amazon chose TLA$^+$. In: Ait Ameur, Y., Schewe, K.D. (eds.) ABZ 2014. LNCS, vol. 8477, pp. 25–39. Springer, Berlin (2014). https://doi.org/10.1007/978-3-662-43652-3_3
23. Stankaitis, P., Iliasov, A., Ait-Ameur, Y., Kobayashi, T., Ishikawa, F., Romanovsky, A.: A refinement based method for developing distributed protocols. In: IEEE 19th International Symposium on High Assurance Systems Engineering (HASE), pp. 90–97, January 2019
24. Whitwam, F., Kanner, A.: Control of automatic guided vehicles without wayside interlocking. Patent US 20120323411, A1 (2012)