# A Formal Methodology for Engineering Heterogeneous Railway Signalling Systems

Paulius Stankaitis

A thesis submitted for the degree of *Doctor of Philosophy*

School of Computing

Newcastle University

Newcastle upon Tyne, UK

August 2021

# Abstract

Over the last few decades, the safety assurance of cyber-physical systems has become one of the biggest challenges in the field of model-based system engineering. The challenge arises from an immense complexity of cyber-physical systems which have deeply intertwined physical, software and network system aspects.

With significant improvements in a wireless communication and microprocessor technologies, the railway domain has become one of the frontiers for deploying cyber-physical signalling systems. However, because of the safety-critical nature of railway signalling systems, the highest level of safety assurance is essential. This study attempts to address the challenge of guaranteeing the safety of cyber-physical railway signalling systems by proposing a development methodology based on formal methods. In particular, this study is concerned with the safety assurance of heterogeneous cyber-physical railway signalling systems, which have emerged by gradually replacing outdated signalling systems and integrating mainline with urban signalling systems. The main contribution of this work is a formal development methodology of railway signalling systems. The methodology is based on the Event-B modelling language, which provides an expressive modelling language, a stepwise model development and a proof-based model verification. At the core of the methodology is a generic communication-based railway signalling Event-B model, which can be further refined to capture specific heterogeneous or homogeneous railway signalling configurations. In order to make signalling modelling more systematic we developed communication and hybrid railway signalling modelling patterns.

The proposed methodology and modelling patterns have been evaluated on two case studies. The evaluation shows that the methodology does provide a system-level railway signalling modelling and verification method. This is crucial for verifying the safety of cyber-physical systems, as safety is dependent on interactions between different subsystems. However, the study has also shown that automatic formal verification of hybrid systems is still a major challenge and

must be addressed in the future work in order to make this methodology more practical.

# Acknowledgments

First and foremost, I would like to express my enormous gratitude and thank my supervisor, Alexander Romanovsky, especially for his patient support and guidance throughout the entire journey. I also would like to thank for the opened opportunities to collaborative with some many talented and wonderful people from all across the world.

I would like to express a special thanks to Alexei Iliasov for all the valuable discussions and collaboration from which I learnt so much. I am also very grateful to Simon Chadwick and other colleagues from Siemens Rail Automation for useful feedback and hospitality. For insightful feedback, I also would like to thank Leon Freitas.

Throughout my studies, I had an amazing opportunity to collaborate with colleagues from INPT – ENSEEIHT (Toulouse, France), National Institute of Informatics (Tokyo, Japan) and Systra Scott Lister (London, United Kingdom). In particular, I would like to thank Yamine Aït-Ameur, Fuyuki Ishikawa, Tsutomu Kobayashi, Guillaume Dupont, Guillaume Babin, Dominic Taylor and Milo Lloyd for the fruitful discussions and hospitality.

Special thanks goes to the thesis examiners Michael Harrison and Marc Frappier for productive discussions and highly valuable feedback on this work.

I would like to thank and dedicate this thesis to my amazing family, friends and Claire without their encouragement and support I would not have done this.

The work presented in this thesis is supported by an iCASE studentship (EPSRC and Siemens Rail Automation).

# Publications

Some of the work presented in this thesis has been published and presented at the international conferences.

1. P. Stankaitis and A. Iliasov, ”Safety Verification of Heterogeneous Railway Networks,” in Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification (T. Lecomte, R. Pinger, and A. Romanovsky, eds.), pp. 150–159, Springer International Publishing, 2016.

   The student-category paper introduced a new problem of safety assurance of heterogeneous railway signalling networks and discussed key challenges to formal modelling and verification of such systems. The research and paper was lead (and written) by P. Stankaitis under a supervision of the co-author.

2. P. Stankaitis, A. Iliasov, Y. Aït-Ameur, T. Kobayashi, F. Ishikawa, and A. Romanovsky, ”A Refinement Based Method For Developing Distributed Protocols,” in IEEE 19th International Symposium on High Assurance Systems Engineering (HASE), pp. 90–97, 2019.

   The paper introduces Event-B patterns for communication modelling and distributed protocol refinement (presented in Chapter 4 of this thesis). The research was carried out by P. Stankaitis while the co-authors provided useful feedback throughout a number of discussions. The paper was primarily written by P. Stankaitis with significant changes made due to co-authors corrections and suggestions.

3. P. Stankaitis, G. Dupont, N. K. Singh, Y. Aït-Ameur, A. Iliasov and A. Romanovsky, ”Modelling Hybrid Train Speed Controller using Proof and Refinement,” in 24th International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 107-113, 2019.

In this paper Event-B hybrid modelling patterns developed by G. Dupont have been applied to modelling and verification of train speed controllers (presented in Chapter 5 of this thesis). The research was jointly lead by P. Stankaitis and G. Dupont who jointly developed the formal Event-B model and co-wrote the paper. The remaining co-authors have significantly contributed towards improving the quality of the paper.

4. P. Stankaitis, A. Iliasov, T. Kobayashi, Y. Aït-Ameur, F. Ishikawa, and A. Romanovsky, ”Formal distributed protocol development for reservation of railway sections,” in Rigorous State-Based Methods (A. Raschke, D. Méry, and F. Houdek, eds.), pp. 203–219, Springer International Publishing, 2020.

   This paper describes a novel distributed protocol for reservation of railway sections formally developed by utilizing the methodology proposed in this thesis (presented as a case study in Chaoter 6 of this thesis). The formal protocol Event-B model was developed by P. Stankaitis while T. Kobayashi helped completing few proofs in the final refinement and A. Iliasov developed a stochastic Monte-Carlo protocol model for a performance analysis. The paper was primarily written by P. Stankaitis with changes made due to co-authors corrections and suggestions.

5. A. Iliasov, P. Stankaitis, D. Adjepon-Yamoah, and A. Romanovsky, ”Rodin Platform Why3 Plug-in,” in Abstract State Machines, Alloy, B, TLA, VDM, and Z (M. Butler, K.-D. Schewe, A. Mashkoor, and M. Biro, eds.), pp. 275–281, Springer International Publishing, 2016.

   The author of this thesis has developed an axiomatic Why3 library of Event-B operators in order to support Event-B proof obligation verification in Why3.

# Research Reproducibility

The formal Event-B and PRISM models presented in this thesis are available online for the research reproducibility purposes as Portable Document Format files or in the Rodin and PRISM modelling environment formats.
A generic communication-based signalling Event-B model (discussed in Chapter 5) available at:
`http://stankaitis.uk/2019/06/`

The Event-B and PRISM models of a distributed signallig system (discussed in Chapter 6) available at:
`http://stankaitis.uk/2019/02/` [Event-B model]
`http://stankaitis.uk/2019/04/` [PRISM model]

A heterogeneous railway signalling Event-B model (discussed in Chapter 6) available at:
`http://stankaitis.uk/2019/06/`

# Contents

# List of Figures

# Listings

# List of Tables

# Chapter 1

# Introduction

This chapter describes the motivations behind the thesis and the main topics related to this work. Research problems, objectives and thesis contributions are also defined in this chapter. In this chapter, we also discuss the related work and provide the structure of the thesis.

## 1.1 Cyber-Physical Systems

Over the last few decades, the complexity of computer-based systems has grown significantly (software aspect [1], hardware aspect [2]) as did society's reliance on them. The advance of computer-based systems has been so significant that in the early 2000s a new, more comprehensive term has been coined to describe modern computer-based systems - cyber-physical systems [3]. Today, they are paramount in transportation, medical care and the financial sector and are often classified as a safety (or mission) critical as their failure is likely to have tragic implications on human lives, property and the environment. Because of their universal application, complexity and safety-critical nature, one of the grand challenges in computer science is to develop rigorous and practical methods, which would allow to design and reason about complex cyber-physical systems.

### 1.1.1 Cyber-Physical Transportation Systems

For years, the transportation sector has been at the forefront of using computer-based systems with the first use of computers (with software) on aircraft in 1968 (Litton LTN-51 Inertial Navigation Systems [1]), 1980s for railway signalling (Solid State Interlocking [4], United Kingdom) and automotive systems. Due to significant advances in communication and computing technologies, modern computer-based transportation systems with tight integration of computing, communication and control are called cyber-physical transportation systems [3]. The objective of a cyber-physical transport system is to observe and safely control the physical processes of a plant through a network of distributed embedded computers.

To some extent, many challenges in designing cyber-physical transportation systems have been studied (e.g. physical environment constraints) in the field of embedded systems. Generally, an embedded application would be considered a small, closed *box* that does not interact with other systems and so extensive bench testing could be performed to assure its safety [5]. On the other hand, cyber-physical transportation systems cannot be considered a simple closed system due to networking, and so system bench testing is no longer an adequate verification technique. The verification problem becomes significant for safety-critical transportation systems, as the properties of an entire system must be analysed [6]. There are many cyber-physical transport system development and verification challenges, such as ensuring cyber-physical transportation systems dependability, safety, availability, security and many others properties. The general challenges of cyber-physical systems are more widely and excellently discussed in the surveys by Gunes et al. [7], Broy & Schmidt [8] and Cyber-Physical Systems Steering Group [9]. In the following, we would like to emphasise four system's properties-challenges, which are particularly relevant to the particular cyber-physical transportation systems we considered and addressed in this research.

**Heterogeneity** Heterogeneous systems encompass sub-systems with different characteristics, requirements and they are common in the cyber-physical transportation system domain. The classic component-based design approaches were successfully used to develop complex systems due to their components and requirements homogeneity [9]. Cyber-physical transportation systems are inherently heterogeneous and, therefore, new theories, methodologies and tools are needed to integrate the system's heterogeneity into the design process [10].

**Concurrency** By nature, the cyber-physical transportation systems are concurrent and time-sensitive, which has historically complicated system verification. As discussed by Edward A. Lee in his seminal paper [5], current concurrency models and the missing temporal semantics in computing could obstruct the deployment of next-generation cyber-physical systems.

**Unpredictability** The cyber-physical transportation systems do not operate in a completely controlled environment; some uncertainty can be introduced by the real world or by using computer systems based on statistical models (e.g. machine learning). Over the many years, the deterministic modelling approaches (e.g. ODEs) have proven extremely useful for designing and analysing systems [11]. However, it is often necessary to introduce uncertainty into the model, for example, to capture the system's disturbances [12]. Also, as correctly pointed out in [12], in the real-world engineering practices requirements are often probabilistic.

**Interoperability** It is a capability of a network of systems to work together and complete a task. As discussed by Baheti and Gill [13], currently there is no standardised methodology to support the development of interoperable cyber-physical systems.

### 1.1.2 Cyber-Physical Railway Signalling Systems

Over the last few decades, developments in communication and computing technologies have enabled railway engineers to develop novel railway signalling systems. The modern railway signalling systems, such as European Train Control System (ETCS) [14] or Communication-Based Train Control (CBTC) [15], are safety-critical cyber-physical transportation systems, potentially improving the capacity, interoperability and reliability of railway networks. In modern signalling systems a train continuously receives a permitted travelling distance via wireless communication and an on-board train computer calculates the fitting speed profile. The calculated speed profile of a train is then dynamically followed and supervised by a driver or a computer-based speed controller. The increased dependability on the on-board computers and algorithms for computing, controlling and monitoring the speed profile of a train requires the highest level of system assurance and poses new challenges to signalling engineers on ensuring the safety of the cyber-physical railway signalling systems.

Another engineering challenge arises by gradually modernising and replacing parts of a large regional railway network with novel railway signalling systems, thus creating a heterogeneous railway signalling network. In other instances, mainline railway networks are integrated with urban networks (e.g Crossrail Network, London [16]), similarly creating another type of a heterogeneous signalling system. The rolling stock in a heterogeneous signalling system must safely and efficiently transition from one signalling area to another. Demonstrating a safe and efficient train signalling transition has been a major issue in developing the Crossrail signalling system, resulting in significant deployment delays [17].

## 1.2 Formal Methods for Railway Signalling Systems

To address the challenges of engineering complex safety-critical systems, some companies and industries have turned their attention to formal methods. Formal methods are mathematical, model-driven methods that provide a systematic and rigorous approach to developing complex systems. Even though there has been an increase, their adaptation in the industry is still scarce [18]. However, one domain that had success in applying formal methods is the railway domain.

In 1989, the SACEM system was introduced in Paris (RER Line A) for controlling regional express trains. Today, the SACEM system is considered a major breakthrough for formal methods in railway, as a large part of that system was formally specified and verified using the B method [19]. The B method was further used in developing other systems in Paris (Line 14, Paris Roissy Airport shuttle), New York (Canarsie line) and showed that project costs could be significantly saved by investing resources in formal system specification and verification. These projects also significantly impacted on standards for developing safety-critical railway control systems and industry; currently, formal methods are rated as highly recommended [20] and several companies provide formal railway system verification as a service.

Due to early industrial success in the railway domain, academic researchers have also been interested in developing and applying new techniques and tools to the railway domain. In particular, the interlocking verification has raised the interest in the automatic verification community (model checking), as so-called control tables could be easily formalised with safety properties expressed in the model checking native temporal logic [21].

However, despite formal methods progress in the railway domain, little has been done to address the cyber-physical nature of modern railway systems. First, the more traditional verification approaches, like model checking, do not scale well for larger, more complex systems. Furthermore, cyber-physical systems have both discrete and continuous behaviours, which are best captured by a hybrid automaton model. However, an algorithmic verification of hybrid models with available model checking tools is very limited, even under severe restrictions [22,23]. Second, the formal specification languages used so far lack the expressivity to formalise systems such as the European Train Control System or heterogeneous signalling systems, described in Section 1.1.2.

## 1.3 Research Questions

Cyber-physical railway signalling systems increasingly play a more important role in the transportation domain by replacing out-of-date signalling systems. A gradual replacement of older signalling systems or integration of signalling systems (e.g. mainline and metro) creates heterogeneous signalling systems, where rolling stock must safely transition from one system to another. In this research, we aim to address the problem of engineering safe heterogeneous cyber-physical railway signalling systems. In the following, we formulate the main research problem as the following research question:

**Research Question**$_1$ Can heterogeneous cyber-physical railway signalling systems be pragmatically engineered by using formal methods?

The second set of research questions arise from the selected approach to apply formal methods.

**Research Question**$_2$ Can a multifaceted methodology be developed for modelling and reasoning about heterogeneous cyber-physical railway signalling systems with existing theories and tools?

**Research Question**$_3$ Are there (or is it possible to develop) adequate and scalable verification tools to reason about hybrid, stochastic heterogeneous cyber-physical railway signalling systems properties?

## 1.4 Research Objectives

The big picture challenge we address in this research is the difficulty of modelling and verifying cyber-physical systems. Specifically, we are interested in classes of cyber-physical railway signalling systems, which are distributed and heterogeneous. In the paragraph below, we outline our five main research objectives.

**Objective**$_1$ To define modelling and verification requirements for the engineering methodology of heterogeneous cyber-physical railway signalling systems.

**Objective**$_2$ To evaluate the existing modelling and verification approaches and their integration possibility into a multifaceted methodology.

**Objective**$_3$ To develop a multifaceted methodology architecture and an engineering process using selected techniques and tools.

**Objective**$_4$ To improve (or develop) techniques, tools, simplifying heterogeneous cyber-physical railway signalling system modelling and improving verification automation.

**Objective**$_5$ Evaluating the proposed architecture and engineering methodology with railway-related case studies, which could address cyber-physical railway signalling systems properties, such as heterogeneity, unpredictability, and distributivity.

## 1.5 Thesis Structure and Contributions

The structure of the thesis (see Figure 1.1) follows a top-down approach where we begin by describing the high-level research problem of heterogeneous railway signalling safety, specifically, its origins and importance - followed by, introduction to formal methods as a system development method and our solution for guaranteeing the highest level of heterogeneous railway signalling system safety and its shortcoming for our research problem. Moreover, we elicit key research objectives, which initially provided the structure for addressing the research problem (**Chapter 1**).

The following chapter (**Chapter 2**) aims to refine the problem by constructing a high-level general heterogeneous signalling system model and extracting key requirements for the solution implementation. The second objective is to evaluate (and justify) formal methods regarding other approaches to developing cyber-physical transportation systems and select an adequate formal method based on requirements acquired from describing the high-level heterogeneous system model.

The following chapter (**Chapter 3**) discusses the first contribution of the thesis - $c_1$) a formal development methodology of heterogeneous railway signalling systems. First, the chapter provides an overall architecture of the methodology with the Event-B formal specification language forming its foundations. Second, the chapter discusses the three-step process of formally developing heterogeneous railway signalling systems using the introduced methodology. The chapter concludes by again discussing the overall methodology regarding all previous requirements.

**Chapter 4** and **Chapter 5** introduce three other thesis contributions: $c_2$) Event-B patterns for communication modelling, $c_3$) Event-B patterns for modelling hybrid signalling systems and $c_4$) a generic Event-B model of hybrid communication-based signalling systems. The methodology evaluation chapter (**Chapter 6**) presents two case studies that are used to analyse the applicability of the proposed formal methodology. The first case study focuses on formally developing distributed signalling systems using introduced communication modelling patterns. The contribution of this case study is a $c_5$) a novel and formally proved distributed protocol for a distributed reservation of railway sections. The second case study focuses on refining the generic heterogeneous signalling Event-B model with specific signalling configurations. Therefore, the following contribution is $c_6$) a formal model of a specific heterogeneous signalling system configuration with hybrid system aspects.

In the final thesis chapter **Chapter 7** we summarise the thesis, discuss methodology limitations and potential directions for future work.

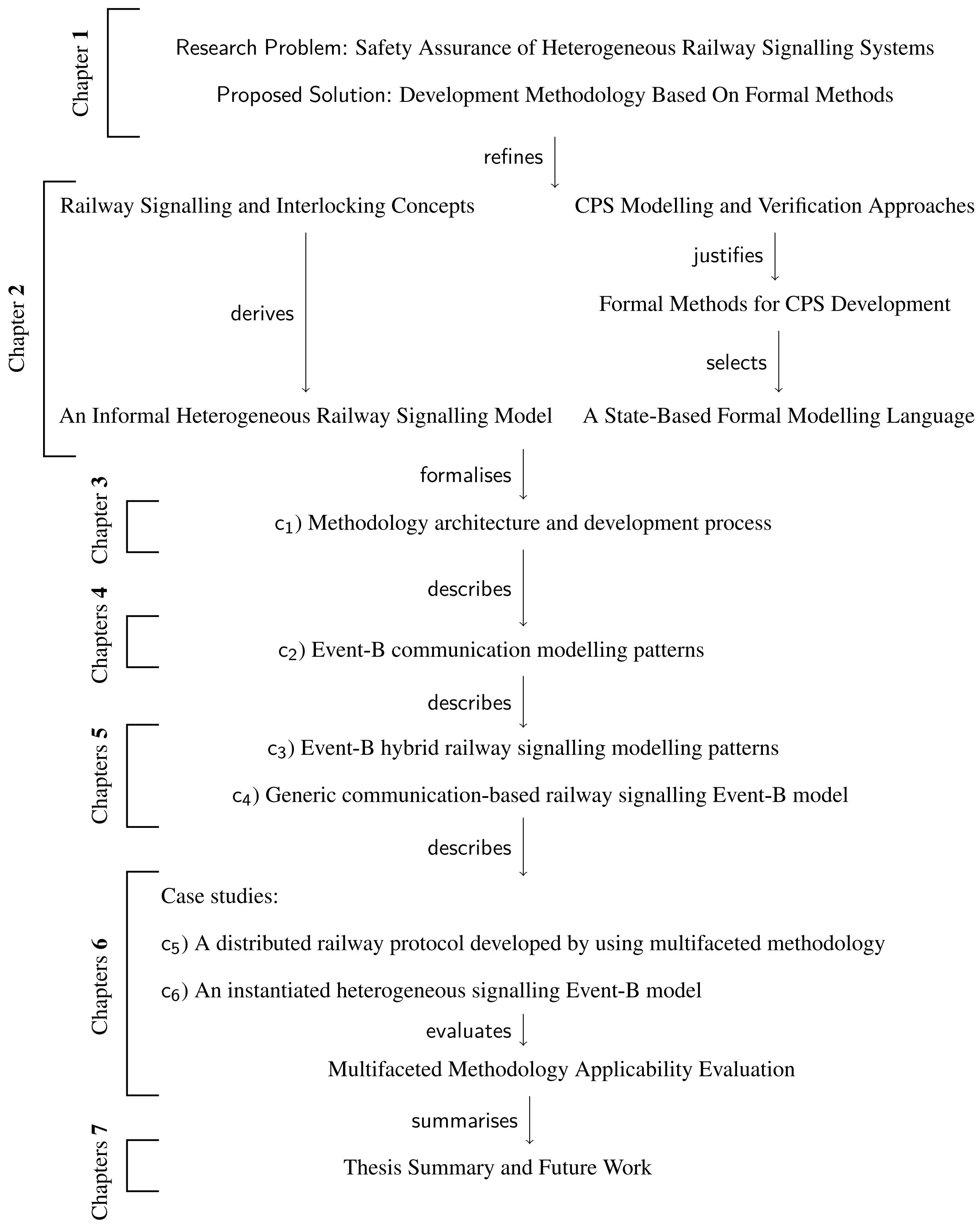

Chapter 1
Research Problem: Safety Assurance of Heterogeneous Railway Signalling Systems
Proposed Solution: Development Methodology Based On Formal Methods
refines
Chapter 2
Railway Signalling and Interlocking Concepts
CPS Modelling and Verification Approaches
justifies
derives
Formal Methods for CPS Development
selects
An Informal Heterogeneous Railway Signalling Model
A State-Based Formal Modelling Language
formalises
Chapter 3
c1) Methodology architecture and development process
describes
Chapters 4
c2) Event-B communication modelling patterns
describes
Chapters 5
c3) Event-B hybrid railway signalling modelling patterns
c4) Generic communication-based railway signalling Event-B model
describes
Chapters 6
Case studies:
c5) A distributed railway protocol developed by using multifaceted methodology
c6) An instantiated heterogeneous signalling Event-B model
evaluates
Multifaceted Methodology Applicability Evaluation
summarises
Chapters 7
Thesis Summary and Future Work

Figure 1.1: Thesis structure with objectives and contributions of each chapter

# Chapter 2

# Requirements and Formal Methods

This chapter presents a generic communication-based model of hybrid railway signalling systems with elicited requirements to facilitate their formal development and a justification for selecting the Event-B formal language as a basis for the formal engineering methodology. The chapter utilises a top-down approach to describe heterogeneous railway signalling systems by firstly introducing general railway signalling concepts, reasons for heterogeneous signalling system emergence and requirements for their model-based development. Since we have formulated heterogeneous signalling systems as a class of cyber-physical transportation systems, we provide a landscape of methods that are used in a cyber-physical system development. In the end, we attempt to justify the Event-B method as a basis for the formal engineering methodology we propose.

## 2.1 Railway Signalling and Interlocking

Railway signalling systems are paramount for safe and efficient railway operation. By means of information gathering, processing and communication, a railway signalling system authorises safe rolling stock movement and controls railway infrastructure [24].

The functional structure of the railway signalling systems can have several fine-grained levels but it is primarily made of the field elements and an interlocking level. The former are railway infrastructure elements, which make it possible to detect and direct rolling stock as well as transmit information to the train drivers. Examples of the field elements include:

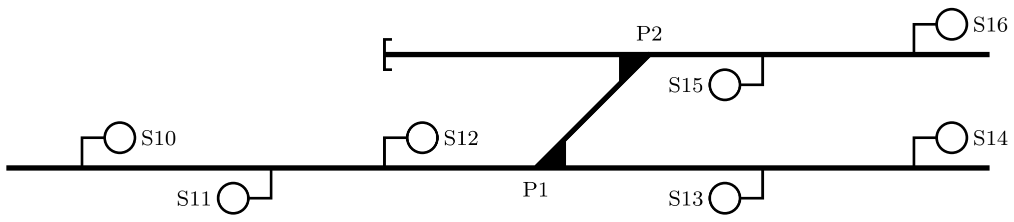


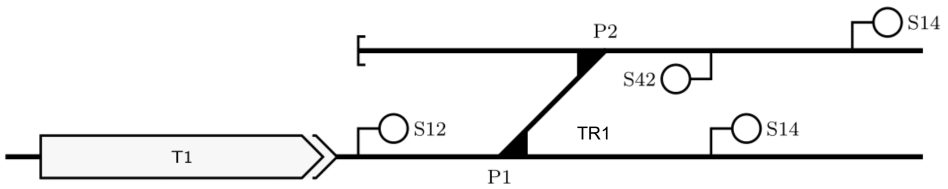
Figure 2.1: An example of a railway layout (field elements) with a fixed block operation

- Points or switches ($\mathsf{P_n}$ in Figure 2.1) are mechanical railway devices, which allow rolling stock to change tracks;
- Colour or mechanical signals ($\mathsf{S_n}$ in Figure 2.1) are devices which visually communicate a permitted travelling distance (and speed) to the rolling stock drivers;
- Track clear detection systems (track circuits) are railway devices which make it possible to detect the presence and absence of the trains;

These elements are connected to the interlocking device - in this day and age, mainly a computer-based device. An interlocking device gathers data from the field elements, ensures that conflicting trains and unsafe movable infrastructure movements are prevented and displays a signal to the train driver.

Even though there are many technological and operational differences between various signalling systems, the fundamental signalling principles are very similar. The most traditional principle to control train separation is a fixed block signalling. The fundamental principle of the fixed block signalling is to divide a railway network into block sections and protect them with an entry and exit signal. Separation of trains is ensured by only allowing block sections to be occupied by a single train. An example of the railway layout with the fixed block operation is shown in Figure 2.1.

The operational aspects of the fixed block signalling are controlled by the interlocking device. An interlocking device implements an interlocking (control) table, which specifies conditions under which a specific block can be reserved or released. An example in Figure 2.2 illustrates a situation in which an interlocking has received a request to reserve block B1 for train T1. The interlocking will check whether track circuit TR1 is clear, point P1 is in a normal position and conflicting blocks are not set. If these conditions are satisfied, the interlocking will reserve the block and set the entry signal to proceed.



| Block ID | Entry Signal | Exit Signal | Tracks Clear | Points Locked | Blocks Locked |
|---|---|---|---|---|---|
| B1 | S12 | S14 | TR1 | P1(N) | B2, B3 |

Figure 2.2: An example of a railway layout with fixed block operation (block reservation)

The modern alternative to a fixed block signalling operation is a moving block operation, where a block is dynamic and associated with a moving train (see Figure 2.3). The moving block operation technology allows reduction of spatial separation between trains, and thus increases the overall capacity of the railway network. To achieve that, a moving block system requires a continuous and precise localisation of rolling stock and safe on-board algorithms for computing speeds profiles given the permitted travelling distance. The moving block technology is currently only available for urban lines and still to be deployed on the inter-city lines.

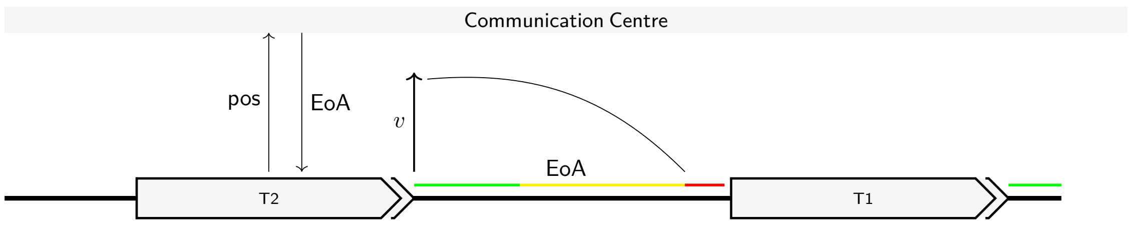


Figure 2.3: An example of a railway layout with moving block operation

In addition to an interlocking subsystem, a moving block signalling systems relies on an additional subsystem for managing rolling stock operation. The subsystem can be generally considered as a communication centre, as the main functionality of the communication centre is commanding rolling stock movement by exchanging information with trains and an interlocking (further discussed in the following section). In the moving block signalling systems, trains periodically report their position to the communication centre and receive back a distance they are allowed to travel, specifically the end of the movement authority (EoA). The main difference from a fixed block signalling operation is that a movement authority is generally issued to the rear of next train or track switch. As illustrated in Figure 2.3, a train will use received EoA information together with train-borne equipment readings to compute a safe speed profile (or braking curve) which it needs to follow. The following section discusses modern communication-based signalling systems, which have been proposed to enable moving block signalling operation.

### 2.1.1 Communication-Based Signalling Systems

For decades, railway signalling systems have relied on mechanical semaphores or coloured lights to communicate a movement authority to trains. The advances in communication and computing technologies have enabled railway engineers to develop novel signalling systems based on the continuous radio-based communication and dynamic speed supervision.

The benefits of such systems are significant. Firstly, trains can run closer together as their movement authority can be updated continuously and speed profile calculated more accurately. The reduced headways between trains can significantly increase network capacity without major infrastructure modifications. Furthermore, the radio-based signalling system reduces reliance on the trackside equipment as most of information can be wirelessly transmitted and displayed in-cab. The reduced reliance on the trackside equipment not only increases reliability but also decreases maintenance costs. Communication-Based Train Control (CBTC) and European Rail Traffic Management System (ERTMS) have been the two most widely deployed communication-based signalling systems.

The European Rail Traffic Management System is a set of standards for the operation and communication-based control of railway networks. Originally the ERTMS commenced as the European Commission [25] initiative to address the cross-border interoperability problem in Europe but since then ERTMS and its versions have been widely accepted as a mature standard and deployed world-wide. The management system is made of two crucial subsystems: Global System for Mobile Communications - Railway (GSM-R) and European Train Control System (ETCS). Depending on network specific requirements, project costs and other parameters, the latter can be implemented as one of five application levels (ETCS 0, NTC, 1, 2, or 3).

**GSM-R** The Global System for Mobile Communications provides a standardised wireless train-to-trackside communication medium customised for railway signalling systems. The communication solution provides a functionality for a driver to directly communicate with a signalling centre as well as a channel for train-related data transmission.

**ETCS** European Train Control System is an essential part of the ERTMS systems responsible for managing and protecting rolling stock. ETCS provides automatic train protection (ATP) with an on-board computer which calculates the safe speed profile to which a driver must adhere and the system will take actions if the current speed exceeds the calculated safe speed. The on-board computer (or European Vital Computer) is also responsible for controlling and interacting with other rolling stock subsystems.

ETCS also provides an in-cab signalling system with the Driver-Machine Interface (DMI) which displays information such as the current speed, speed limit and travelling distance to a target speed. The interactive DMI display also enables a driver to enter information for starting the mission, changing operation mode or overriding an end-of-the-movement authority.

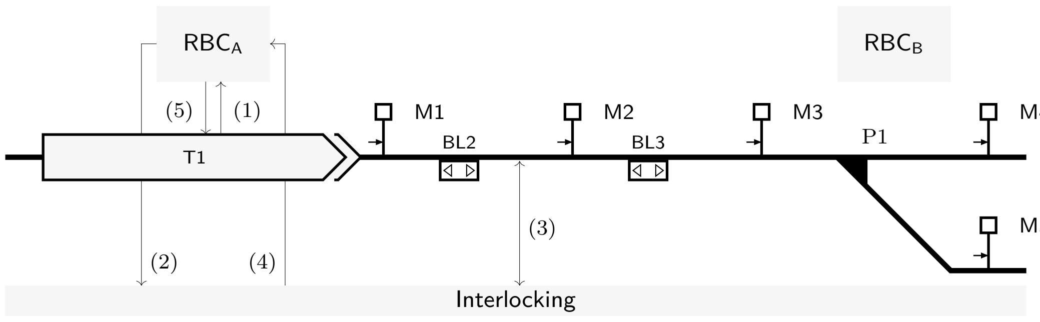


Figure 2.4: An example of ETCS Level 2 layout and operation

ETCS Level 2 application introduces two main new trackside field elements: Eurobalises and Radio Block Centres (RBC).

Eurobalises are fixed transmitters that are placed on the track and transmit the same information to all passing trains. The train passing over a balise will use an on-board device to read information stored on the balise such as its position, track gradient and speed limit. The information received from the Eurobalise together with an on-board odometry system enables a train to determine speed and distance travelled.

Radio Block Centre is another vital ETCS component responsible for issuing a movement authority - the distance train is allowed to travel. To compute a movement authority for a train RBC interacts with all major ETCS railway signalling subsystems, including interlocking, rolling stock and traffic management systems.

In the following paragraphs (with Figure 2.4) we discuss three key (simplified) ETCS Level 2 operation scenarios which are related to research problems we address in this thesis. The first procedure is linked to the process of updating the movement authority of a train while the remaining are concerned with homogeneous/heterogeneous train handovers.

**Train approaching the EoA.** The train approaching the end of its movement authority sends a movement authority extension request (1) via GSM-R to the Radio Block Centre it is currently connected $\mathsf{RBC_A}$. When the Radio Block Centre receives an extension request message, it sends a particular route lock message (2) to the interlocking. The interlocking checks the status of the requested route field elements (3); if track circuits of the requested route are free and points are correctly positioned, the interlocking locks the route and sends a confirmation message to RBC (4). RBC sends an updated movement authority (5) to the train which together with the newly calculated speed profile is displayed on the DMI.

**RBC-RBC Handover.** Because of areal coverage limitations of the Radio Block Centre, larger railway networks with ETCS Level 2 must operate with multiple Radio Block Centres. ETCS standards describe a homogeneous handover mechanism between accepting and handing-over Radio Block Centres as well as the train involved [26].

**System Level Transition.** ETCS also provides a standard for trains operating in heterogeneous signalling systems via a so-called Specific Transmission Module [27]. The Specific Transmission Module enables ETCS equipped rolling stock to operate on non-ETCS lines by providing communication interface between ETCS on-board system (European Vital Computer) and non-ETCS trackside. ETCS standards also provide a heterogeneous transition mechanism for trains switching from ETCS system to a non-ETCS system, vice versa or between ETCS lines with different application levels via System Level Transition protocol (see Annex [27]).

### 2.1.2 Heterogeneous Railway Signalling Systems

The modern communication-based signalling systems are increasingly replacing more traditional and out-of-date signalling systems. Railway signalling modernisation processes on the national level are extremely time consuming and costly while a level of railway service must be maintained. The only practical solution is to modernise the network gradually and on a smaller scale, as a result, creating heterogeneous railway signalling systems. The other type of heterogeneous signalling systems are created by integrating urban and mainline signalling systems. In Figure 2.5 we visualise and in the following paragraphs describe a section of a real-world heterogeneous signalling system deployed on the Crossrail network [28].

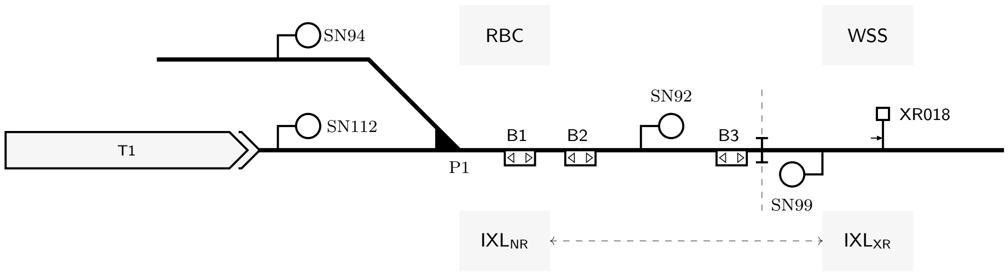


Figure 2.5: ETCS Level 2 (with signals) and CBTC Interface on Crossrail (East Direction)

The Crossrail network signalling area, visualised in Figure 2.5, illustrates an interface between ETCS Level 2 (with signals) and Communication-Based Train Control (CBTC)[1] systems. Despite the fact that both systems are functionally and architecturally very similar, their interface is not standardised, and there, customised signalling solutions must be developed for a safe and on-the-move rolling stock transition.

The interface area comprises two interlocking systems $\mathsf{IXL}_{\mathsf{NR}}$ and $\mathsf{IXL}_{\mathsf{XR}}$ which respectively manage routes on the ETCS and CBTC signalling areas. For rolling stock transition, interlocking systems jointly control a network section between signal SN92 and block marker XR018 to prevent opposing trains arriving from the CBTC area. The $\mathsf{IXL}_{\mathsf{NR}}$ will not allow a train T1 to pass signal SN112 until route from signal SN92 has been jointly reserved.

After receiving a proceed signal from SN112 train T1 will pass over the first CBTC balise group B1 called level transition announcement balises, which informs ETCS on-board computer to essentially instruct the CBTC on-board computer to prepare for controlling a train. The ETCS on-board system will also ask the driver to acknowledge the transition procedure. The train will send the first position report to CBTC trackside (WSS - Wayside System) upon passing localisation balise group B2 but will not be driven by CBTC until it passes balise group B3, which informs the ETCS on-board computer to handover train control to the CBTC on-board unit.

[1]Communication-Based Train Control is a signalling system predominantly used on urban rail lines. The system is architecturally and functionally similar to ETCS Level 2-3 systems.

## 2.1.3 Generic Heterogeneous Railway Signalling System Model

Because of the large number of existing national and international signalling systems, the number of possible heterogeneous signalling system configurations is enormous. Therefore, in the following paragraphs we describe a generic heterogeneous railway signalling model which would capture the fundamental concepts of heterogeneous signalling systems and could then be instantiated to capture a specific heterogeneous signalling configuration.

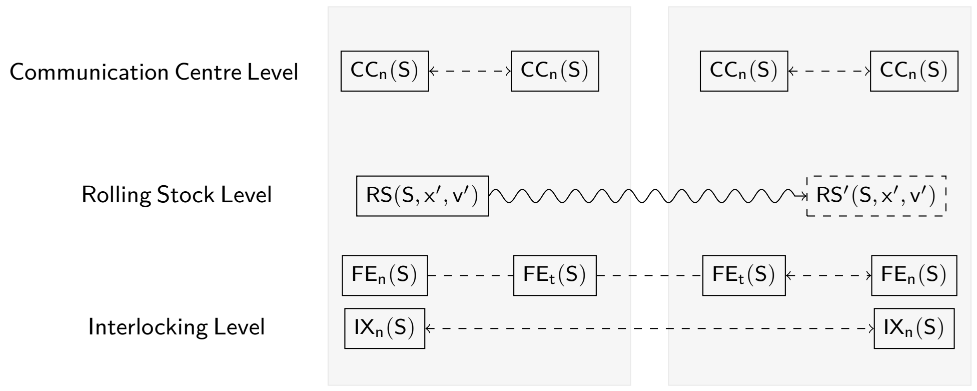


Figure 2.6: A visual representation of the generic heterogeneous railway signalling model

A heterogeneous railway signalling system is a composite system which is created by interfacing two or more distinct homogeneous signalling systems. The fundamental property of a heterogeneous signalling system is that the same signalling principles (or their implementation) are not uniformly applied for the entire railway network. In spite of that, the objective of the heterogeneous signalling system remains to ensure a safe rolling stock operation within homogeneous parts of the network and guaranteeing safe and efficient transitions between non-homogeneous parts of the network.

The generic heterogeneous railway signalling model can be considered as a quintuple which consists of the following sets of elements: field elements, interlocking systems, communication centres, rolling stock and protocols (see Figure 2.6). By nature the model we present is a hybrid model, meaning that, some subsystems of the model can exhibit both discrete and continuous aspects of the system. In the following paragraphs we briefly describe each subsystem of the generic heterogeneous signalling model.

**Rolling Stock** An element of the rolling stock set is a single train characterised by differential equations and a transition system. A train has the functionality to communicate with other signalling subsystems and operate in different signalling areas of a heterogeneous signalling system.

**Communication Centre** The abstract model generalises methods which are used by signalling systems to communicate a movement authority to trains as a communication centre. A communication centre interacts with all major signalling subsystems which include communication centres, rolling stock and interlocking systems.

**Interlocking Systems** An interlocking is a signalling subsystem responsible for preventing conflicting rolling stock movements and controlling a subset of field elements. An interlocking has the functionality to communicate with adjacent interlocking systems, communication centres and field elements.

**Field Elements** A subset of railway infrastructure, which directs, detects and protects rolling stock can be broadly categorised as field elements. Field elements can include equipment such as train detection sensors, rail switches, semaphores, level crossings and others.

**Protocols** An integral part of the heterogeneous signalling model are protocols which define communications between different subsystems of the heterogeneous signalling system. The model we present can generally have two types of protocols: homogeneous signalling area protocols and signalling handover protocols.

### 2.1.4 Railway-Based Engineering Methodology Requirements

In this section, we elicit the main formal engineering methodology requirements with respect to the listed challenges in Section 1.1.1 and specifications of the generic heterogeneous signalling model in the previous section.

**Requirement$_1$** The formal engineering methodology should support system-level modelling as the safety of heterogeneous signalling systems relies on the correctness of subsystems and their interaction.

**Requirement**$_2$ The methodology should provide a generic formal signalling model and support a model extension with specific signalling solutions.

**Requirement**$_3$ The formal engineering methodology should support the specification and verification of system-level properties.

**Requirement**$_4$ The methodology and generic formal signalling model should make it possible to capture and reason about continuous, discrete and stochastic aspects of a system.

## 2.2 Cyber-Physical System Modelling

System modelling is a powerful tool and universal practice in the world of engineering used to analyse various system aspects. System models represent a more abstract, often mathematical interpretation of the same system and makes it possible to analyse a system without constructing it. This enables engineers to detect system's problems earlier and without implementing and exhaustively testing an actual system. Of course, this comes at a cost. Firstly, there is the monetary cost of having human resources and expertise in the system modelling. Secondly, investing time in the modelling phase might result in a product launch delay. The latter cost is sometimes rejected, as the modelling and model verification phase can reduce system testing effort.

A model-based design approach is particularly crucial for developing complex cyber-physical systems. Their heterogeneous nature means that different engineering disciplines are involved in the design process and models can be used to address challenges of the interdisciplinary communication [29]. Over the decades, three main model-based cyber-physical system design approaches have been used: simulation-based, programming-based and formal methods.

**Simulation-based design** The simulation-based design of cyber-physical systems provides an economical method to prototype and analyse systems. Contrary to the manual testing of complex cyber-physical systems, a system validation via a simulation can produce a wider verification coverage at a lower cost and flexibility to validate different operational scenarios. Khaitan and McCalley [30] provide an excellent survey of design techniques and application of cyber-physical systems, while in the following paragraphs we overview the most commonly used tool-based approaches that support system simulation.

Simulink [31] is one of the most popular block-based tool suites used in modelling cyber-physical systems, or more specifically, in co-modelling digital and physical system characteristics. A Simulink model can be constructed by interconnecting discrete and continuous blocks (actors) from available pre-defined or custom-made libraries. To verify a model, Simulink provides a design verifier (SDV) to perform formal testing, which is limited to discrete Simulink models, and model verification blocks to monitor continuous aspects of linear systems during simulation. SCICOS (the SCILAB package) [32] is another popular block-based simulation tool-set which also allows generating executable C code of discrete-time system models.

Another class of simulation-based design methods can be categorised as equation-based. The block-based simulation methods require a user to manually transform a physical system model, typically defined by differential equations, to a block diagram. This can be an error-prone process [33]. The equation-based approaches makes it possible to define systems in their innate structure of differential equations. Simulation tools can then automatically provide the simulation code. OpenModelica [33, 34] and 20-sim [35] are two of the most popular tool suites for the equation-based simulation of cyber-physical systems.

Often, a single simulation suite does not provide all the functionality needed to model a system. Particularly, considering the heterogeneous nature of cyber-physical systems, integrating multiple simulation tools can be necessary. The co-simulation approach provides an algorithmic mechanism to transfer intermediate simulation results between different tools responsible for simulating separate sub-models. Functional Mock-up Interface (FMI) [36] is an open source and a tool-independent co-simulation framework which is widely used in academia and in the industry. In a FMI-based co-simulation environment, a *master* algorithm orchestrates discrete data exchanges between different sub-models, *slaves*, which are simulated with their innate solvers. In some researches, authors have attempted to utilise different cyber-physical system design approaches with co-simulation standards. For example, Savicks [37] and Larsen et al. [37] developed methodologies for integrating formal and simulation-based design approaches of cyber-physical systems. These methodologies make it possible to validate systems via simulation as well as verify system properties formally by using theorem proving or model checking techniques.

In summary, the simulation-based design of cyber-physical systems enables engineers to efficiently design, analyse and modify systems in their early development phase. The simulation tools provide engineers graphical notations which simplify virtual prototyping and make models more readable. Still, in spite of these advantages, system verification only via simulation is not sufficient as critical scenarios might be missed due to the low coverage [30]. A promising direction to improving verification quality is integrating simulation-based approaches with methods based on formal verification.

**Programming-based design** Software plays a crucial role in the majority of cyber-physical applications. Programming cyber-physical systems is a particularly challenging task which requires to consider system concurrency, timeliness and physical system characteristics. In the paper by Lee [11] the author critically discusses limitations and wide use of imperative programming languages (e.g. the C language) to developing cyber-physical systems.

Imperative programming languages have been successfully used to capture cyber aspects of cyber-physical systems. However, once programmed cyber-physical systems are deployed, they interact with a physical world and real-time processes [38]. And still, the majority of imperative programming languages have no mechanism to express or control timing behaviour. The temporal constructs have been introduced in synchronous reactive programming languages [39, 40] where the notion of time is viewed as the order of instantaneous events [41]. A time-triggered Giotto [42] programming language provides a similar synchronous approach but with non-instantaneous execution of events. The programming languages discussed have strong formal semantics and have been combined with formal verification tools [43]. More widely known languages Ada [44] and Java [45] have also been extended with real-time programming concepts and used on industrial-level applications [46].

Papers by Hnat et al. [47] and Gummadi et al. [48] present a *macroprogramming* approach to developing cyber-physical systems. Unlike a node-level programming (*microprogramming*) concept where cyber-physical sub-systems are programmed individually, macroprogramming approaches with MacroLAB and Kairos make it possible to write a single program, which is then automatically decomposed into microprograms. The macroprogramming approach makes it possible to tackle the issue of scalability and interoperability to developing cyber-physical systems, but lacks foundations to address the physical nature of these systems. Overall, the programming-based methods provide an intuitive development approach to software engineers. However, the use of languages with temporal features in the industry is still scarce and software-based system validation still relies heavily on implementation testing.

**Formal methods** Formal methods are mathematical model-driven methods, which provide a systematic approach for developing complex systems. They offer an approach to specify systems precisely via a mathematically defined syntax and semantics as well as support a formal system validation by using semi-automatic or automatic techniques.

Over the years various classical formal languages and tools have been used or extended to specify and verify cyber-physical systems. Unified Modelling Language (UML) [49] has been one of the most commonly used modelling languages for software development. Given the importance of timeliness properties for many cyber-physical systems UML was extended [50] with a real-time system modelling profile and used to model industry-level systems [51]. The semantics of Statecharts, a popular formalism for modelling reactive systems, have been extended with the real-time bound notion [52]. Formalisms, like Timed Statecharts, can provide a more hierarchical complex system view, which is particularly useful for capturing multi-layered cyber-physical systems [29]. However, languages like UML-RT or Timed Statecharts are more applicable for a formal system documentation and not their formal verification.

The integration of different formalisms has been another approach to capturing different cyber-physical system characteristics and take advantage of different development methods. Enriching block-based system development methods with a back-end formal verification has been a popular approach to improve the quality of model verification (e.g [53, 54]). Similarly, an integration of different formal specification languages has been attempted to enable the capturing of different systems characteristics formally. In work by Hoenicke et al. [55], the authors semantically integrated three well-known methods which were successfully used to model certain system aspects but not suited to cover all cyber-physical system aspects.

The popular state-based formal specification languages such as the B-method [19], Event-B [56] and Vienna Development Method (VDM) [57] have been successfully used for a formal safety-critical system development [18]. However, they have not been originally developed to capture hybrid or real-time system aspects and, therefore their application to developing cyber-physical systems has been limited. In the last few decades, there have been several works towards extending these languages. The Event-B method has been extended with time constraint [58] and hybrid modelling patterns [59–61]. The ADVANCE project [62] developed a formal cyber-physical systems modelling and verification methodology based on Event-B and UML methods. The methodology was used to develop two industrially led case studies in energy and transportation domains. The VDM modelling framework has been extended with the notion of time [63] and integrated into the cyber-physical system development framework based on the model exchange and co-simulation [37]. Several industrially driven case studies have been conducted with the framework where VDM-RT was used to model discrete elements of the cyber-physical system [64]. The other well-known state-based formal specifications languages TLA$^+$ [65] and Z [66] have been extended with real-time and hybrid modelling concepts.

In particular, due to an automated verification, formal verification approaches based on the model checking have been used to demonstrate system correctness. UPPAAL [67] is a popular tool-set for verifying hybrid models with linear dynamics which has been used to verify various cyber-physical systems [68, 69]. The model checker SPIN, which has been widely used to verify concurrent systems, was extended (RT-SPIN [70]) to capture real-time system aspects and used to verify cyber-physical energy systems [71]. The survey by Khaitan et al. [30] provides an extensive list of works where model checkers were used to verify cyber-physical systems.

Formal methods are increasingly gaining recognition in the domain of cyber-physical systems. The two main trends we have identified are: a modification of formal specification semantics to capture temporal and hybrid system aspects and integration with existing simulation/programming-based approaches. Overall, in comparison to alternatives, formal methods can provide a much higher system quality of assurance, but problems such as formal verification scalability must be addressed.

### 2.2.1 Hybrid System Modelling and Verification

One of the key aspects of the cyber-physical systems is a tight integration of digital and physical system aspects. In particular, for cyber-physical transportation systems, physical system characteristics play a crucial role in guaranteeing the safety of the system. Over the years, many hybrid model structures have been proposed for engineering safe cyber-physical and embedded systems [72] but generally a hybrid automaton structure [73] has been viewed as the most popular choice. Hybrid automata can be viewed as a finite state machine extension with real-valued states and can be mathematically defined in the following way:

| | |
|---|---|
| **Hybrid Automata** | $H = (X, G, I, In, F, J)$ |
| **Variables** | $X = \{x_1, x_2, \ldots, x_n\}$ where $x_n \in \mathbb{R}$ |
| **Control Graph** | directed graph $G = (V, E)$ where $V$ - vertices and $E$ - edges |
| **Initial Conditions** | $i \in I$ is a predicate which holds in $v \in V$ |
| **Invariant Conditions** | $i \in In$ is a predicate which holds in $v \in V$ |
| **Flow Conditions** | $f \in F$ is a predicate which holds in $v \in V$ |
| **Jump Conditions** | $j \in J$ is a predicate which holds in $e \in E$ |

In Figure 2.7 we provide a hybrid system example that is modelled as a hybrid automaton. The hybrid system has two modes - $\mathsf{Mode}_1$ and $\mathsf{Mode}_2$ - with differential equations as flow conditions, which update a state variable $\mathsf{x}$. Initially, the hybrid system is in $\mathsf{Mode}_1$, which decreases a state variable $\mathsf{x}$ according to the flow condition $\mathsf{x}' = -0.5\mathsf{x}$ until the jump condition $\mathsf{x} < 19$ or node invariant $\mathsf{x} \leq 18$ is satisfied. Similarly, a state variable $\mathsf{x}$ will evolve according to the flow condition $\mathsf{x}' = \mathsf{x}$ in $\mathsf{Mode}_2$ but return to $\mathsf{Mode}_1$ once $\mathsf{x}$ rises above 21.

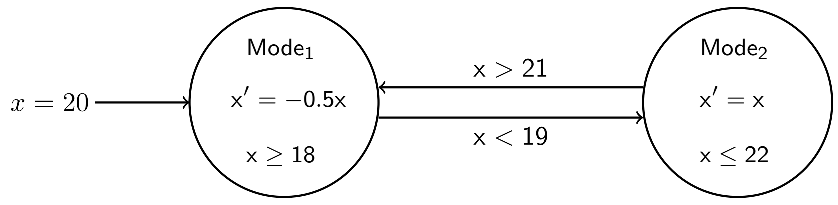


Figure 2.7: A two-mode Hybrid Automata example model

A formal verification of hybrid systems has been a major challenge and an active research area. The challenges of formally verifying hybrid systems arise mainly due to real-valued state variables and systems with non-linear dynamics [22, 23]. Even under severe system dynamic restrictions, an algorithmic verification of hybrid systems (with a single exception of classes of linear hybrid systems [74, 75]) have been very limited and undecidable. Over the years, traditional algorithmic approaches based on computing an exact state-space of hybrid systems have been replaced by computing approximate state-spaces. The so-called *lazy* representation approach makes it possible to significantly increase the number of continuous state variables an algorithmic verification tools like SpaceEx and Checkmate [76, 77] could handle.

The approximation approach has also been extended to the verification of hybrid systems with non-linear dynamics. For instance, the hybridisation method [78] partitions a non-linear system and then approximates its local regions with a simpler dynamical form. The HSolver [79] formal verification tool supports a formal safety verification of non-linear hybrid systems. Other formal verification tools for linear and non-linear are discussed extensively in a survey by Carloni et al. [80]. In spite of significant advances in the algorithmic verification of hybrid systems important challenges like the precision of approximations and capturing dynamical systems with uncertainties remain to be addressed [81].

An alternative approach to verifying hybrid systems by exploring their state-space with model checkers is mathematically proving them. The logic-based approaches, in particular, based on the first-order logic have been widely and successfully applied to proving discrete system properties. However, verifying hybrid systems requires reasoning about its real-valued state transitions over a time period which is not inherently supported by first-order logic. The logics with modal operators such as dynamic logic [82] make it possible to refer to reachable states of a system $\alpha$ with modal operators $[\alpha]$ and $\langle\alpha\rangle$. Therefore, few modal logics [83, 84] have been generalised to allow the system $\alpha$ to be a hybrid system.

In general, deductive verification methods are not limited by the state-space and combined with computer algebra systems can deal with non-linear dynamics. In spite of that, an automated deductive verification is still a major challenge [85] and for an industrial application an interactive proof effort should be dramatically reduced. Nonetheless, there have been several researches, which developed deductive verification methods for hybrid systems. For example, in a seminal work [83] Platzer developed a formalism and logic for reasoning about hybrid systems which uses a deductive verification and can be implemented in the KeYmaera X verification tool [86]. The state-based formal specification languages, which have been extended to capture hybrid system aspects [59], also support a deductive hybrid model verification.

### 2.2.2 Formal Specification Languages

Over the years, numerous formal specifications languages have been developed, often particular methods being better suited for specific problems. To avoid an increased modelling time and project costs, or even unsatisfactory results, selecting an appropriate formal method for a given project becomes crucial.

Different classes of formal methods are based on different modelling paradigms that are better suited to emphasise specific system characteristics [87]. The **history-based** specification languages such as a linear temporal logic [88] aim to model a system as a set of of all behaviours over a period of time. The intrinsic notion of time in these models has been used widely, adapted for modelling reactive real-time systems. The **state-based** formal methods such as Event-B [56], VDM [57] or Alloy [89] capture a state of the system at a single moment via state-mapping functions that define a relationship between an old and a new state. The stepwise model development, expressivity and compositional reasoning are the central features of these languages. **Transition-based** models capture all transitions a system can potentially take via transition triggering events. The Statecharts [90] diagrammatic language is a popular formalism, particularly due to its comprehensible specification format and a state decomposition option.

In our opinion, formally specifying and verifying safety properties of cyber-physical systems can be best addressed with state-based methods. The state-based methods are built upon comprehensible, yet expressive mathematical notions, which can be used to capture heterogeneity and different cyber-physical system aspect cohesion. The state-based models can be constrained by safety invariants, which can be used to express and mathematically prove system-level safety-properties. Formal methods that are based upon a state-based modelling paradigm are not new to various industry domains and have been successfully used to develop complex industrial-level systems [18].

In the following sections, we overview three formal methods and discuss their suitability for a formal engineering methodology of heterogeneous signalling systems. The methods we review support a form of stepwise development or system decomposition which is particularly useful to developing complex cyber-physical systems. To deal with the state-space explosion problem of cyber-physical systems, methods under review are primarily based on a model verification by mathematical proof. Good tool support is crucial for a formal modelling and verification, therefore, selected methods have an established and extensible tool-base.

### 2.2.3 Event-B Method

The Event-B mathematical language used in the system development and analysis is an evolution of the classical B method [19] and Action Systems [91]. The formal specification language offers a fairly high-level mathematical language based on a first-order logic and Zermelo-Fraenkel set theory as well as an economical yet expressive modelling notation. The formalism belongs to a family of state-based modelling languages where a state of a discrete system is simply a collection of variables and constants whereas the transition is a guarded variable transformation.

```
machine M
  sees Context
  variables $v$
  invariant $I(c, s, v)$
  initialisation $R(c, s, v')$
  events
    E$_1$  $=$  any $vl$ where $g(c, s, vl, v)$ then $S(c, s, vl, v, v')$ end
    ...
end
```

Figure 2.8: The structure of the Event-B model

A cornerstone of the Event-B method is the step-wise development that facilitates a gradual design of a system implementation through a number of correctness preserving refinement steps. The model development starts with the creation of a very abstract specification and the model is completed when all requirements and specifications are covered. The Event-B model is made of two key components - machines and contexts which respectively describe dynamic and static parts of the system (see Figure 2.8). The context contains modeller declared constants and associated axioms which can be made visible in machines. The dynamic part of the model contains variables which are constrained by invariants and initialised by an action. The state variables are then transformed by actions which are part of events and the modeller may use predicate guards to denote when an event is triggered. The Event-B method is a proof-driven specification language where model correctness is demonstrated by generating and discharging proof obligations - theorems in the first-order logic. Table 2.1 shows the important proof obligations of the Event-B language. The model is considered to be correct when all proof obligations are discharged.

Rodin [92] is an open source Eclipse-based integrated development environment (IDE) for Event-B model development. Rodin is a core set of plug-ins for project management, formal development, syntactic analysis, proof assistance and proof-based verification. Moreover, it also allows extension points for supporting a range of additional plugins to provide different functionalities and features related to model checking, animation, code generation, additional proof capabilities using SMTs and external theorem provers (i.e., Why3, Isabelle), UML-B, Theory plug-ins, composition and decomposition, refactoring framework, and model editors.

| | |
|---|---|
| Theorems | $\mathsf{A(s,c)} \implies \mathsf{T_c(s,c)}$ |
| Invariant Preservation | $\mathsf{A(s,c)} \wedge \mathsf{I(s,c,v)} \wedge \mathsf{G(s,c,v,x)} \wedge \mathsf{BA(s,c,v,x,v')} \implies \mathsf{I(s,c,v')}$ |
| Event Feasibility | $\mathsf{A(s,c)} \wedge \mathsf{I(s,c,v)} \wedge \mathsf{G(s,c,v,x)} \implies \exists \mathsf{v'} \cdot \mathsf{BA(s,c,v,x,v')}$ |
| Variant Progress | $\mathsf{A(s,c)} \wedge \mathsf{I(s,c,v)} \wedge \mathsf{G(s,c,v,x)} \wedge \mathsf{BA(s,c,v,x,v')} \implies \mathsf{V(s,c,v')} < \mathsf{V(s,c,v)}$ |

Table 2.1: Event-B proof obligations: $\mathsf{A}$ (axioms, $\mathsf{T}$ - theorems, $\mathsf{I}$ - invariants, $\mathsf{G}$ - guards, $\mathsf{BA}$ - before/after predicates, $\mathsf{V}$ - variants)

Even though the Event-B mathematical language is expressive enough for a lot of useful mathematical concepts, it is still desirable to allow users to extend the language. For that reason, a theory extension process has been developed and realised as a Rodin platform plug-in. With the theory extension approach, new theories which include datatypes, operators and proof rules can be defined and proved to be sound through generated proof obligations.

The paper by Banach et al. [60] has proposed an extension to Event-B - Hybrid Event-B - to allow modelling continuous system behaviours. The hybrid extension formalises semantics and syntax for accommodating continuous aspects of hybrid systems. Hybrid Event-B makes it possible to define pliant variables, which are updated via pliant events and a read-only clock. Their proposed approach also introduces additional proof obligations. In a different approach [59] Dupont et al. proposed to support hybrid system modelling by utilising the Event-B language extension tool - Theory plug-in [93]. The paper makes use of the previously developed theory of dense reals [94] and further extends it to support the theory of continuous functions and ordinary differential equation. Their work also provides a generic hybrid model with template discrete and continuous events to simplify modelling of hybrid systems. The advantage of this approach is its practicality, as the method is based on the already existing Event-B and Rodin infrastructure and it does not require further semantical or tooling extensions.

### 2.2.4 Vienna Development Method

Vienna Development Method (VDM) is one of the most established state-based formal methods, that was originally developed at the IBM Vienna laboratory [57]. The computer-based systems are modelled using the VDM-SL (Specification Language) formal specification language, which throughout the years has been extended to support a modelling of object-oriented and concurrent systems [95] and real-time systems [63].

The modelling principle of abstraction, which proposes to suppress less relevant system details is at the core of the VDM modelling approach. Theoretically modelling in VDM starts with the specification of an abstract system model, then through refinement methods based on data reitification and operation decomposition an implementable model is derived. However, the refinement mechanism has not been implemented in tools which facilitate VDM modelling. Structurally, models described in VDM-SL are made of declarations of abstract data types and operations over data which jointly make up a module.

The mission of the VDM method is to provide a rigorous system development, meaning that the properties of the model must be formally proved to ensure a correct-by-construction system implementation. The available VDM tools, such as VDMTools, Overture and others, provide a mechanism for a model validation and verification with static analysis and proof methods.

Throughout few large research projects, DESTECS [96] and INTO-CPS [97], the Overture (with VDM) tool has been integrated into a cyber-physical system development framework that relies on model-exchange and co-simulation paradigms. In particular, the co-simulation mechanism enabled VDM to formally analyse physical system aspects by co-modelling continuous systems aspects with external tools, such as OpenModelica [34] and 20-sim [35].

### 2.2.5 Differential Dynamic Logic

Differential Dynamic Logic ($\mathbf{d}\mathcal{L}$) is a rigorous formal approach for modelling and deductive verification of hybrid systems [83]. The approach proposes to address hybrid system's verification complexity and scalability challenges by utilising a verification approach, which is based on a compositional deductive system verification.

First of all, in $\mathbf{d}\mathcal{L}$ a hybrid system is described textually, as a hybrid program $\alpha$, with its state-space constrained by a predicate $[\alpha]\phi$. In contrary of hybrid systems described as a hybrid automaton, $\mathbf{d}\mathcal{L}$ approach with hybrid programs makes it possible to decompose a system and verify properties about a hybrid system $[\alpha]\phi$ by conjunctively proving subproperties of its subsystems $[\alpha_1]\phi_1 \wedge [\alpha_2]\phi_2$. In Figure 2.9 we provide a list of available operators in hybrid programs and an example model of the system depicted in Figure 2.7.

| | | |
|---|---|---|
| | | $\mathsf{x} := 10; \mathsf{q} = \mathsf{Mode}_1;$ |
| $\mathsf{x}' = \mathsf{f}(\mathsf{x})$ | (continuous evolution) | $($ |
| $\mathsf{x} := \mathsf{f}(\mathsf{x})$ | (discrete jump) | $\cup\,(?\mathsf{q} = \mathsf{Mode}_1; \mathsf{x}' = 0.5\mathsf{x} \,\&\, \mathsf{x} \geq 10)$ |
| $\mathsf{if}\ (\mathsf{X})\ \alpha\ \mathsf{else}\ \beta$ | (conditional execution) | $\cup\,(?\mathsf{q} = \mathsf{Mode}_1 \wedge \mathsf{x} > 20; \mathsf{q} := \mathsf{Mode}_2)$ |
| $\alpha;\beta$ | (seq. composition) | $\cup\,(?\mathsf{q} = \mathsf{Mode}_2; \mathsf{x}' = -\mathsf{x} \,\&\, \mathsf{x} \leq 21)$ |
| $\alpha\cup\beta$ | (nondet. choice) | $\cup\,(?\mathsf{q} = \mathsf{Mode}_2 \wedge \mathsf{x} < 12; \mathsf{q} := \mathsf{Mode}_1)$ |
| $\alpha*$ | (nondet. repetition) | $)$ |

Figure 2.9: The syntax of hybrid programs and an example model

Secondly, the fundamental idea of $d\mathcal{L}$ is to verify systems by mathematically proving them, instead of using methods, which are based on a state exploration. The $d\mathcal{L}$ is a tool-supported method with a theorem prover KeYmaera [86] that makes it possible to describe hybrid programs and prove them with $d\mathcal{L}$ calculus. The $d\mathcal{L}$ approach has already been applied for modelling and verification of railway [98], aviation [99] and automotive [100] system examples. Over the years, the logical $d\mathcal{L}$ foundations have been further extended to address stochastic [101] and distributive [102] aspects of hybrid systems.

## 2.3 Formal Methods Evaluation

The methods we reviewed have significant individual strengths and strong cases for using them for a formal cyber-physical system development. The evaluation is based upon our personal as well as external analysis [59, 103] of these formalisms.

Differential Dynamic Logic ($d\mathcal{L}$) and KeyMaera provide a strong framework for reasoning about the hybrid aspects of cyber-physical systems. The built-in logic requires minimal effort to define continuous system dynamics or no effort in specifying proof tactics, as proof rules are already part of the tool. However, we find the scalability of the framework to be the main obstacle for specifying larger systems. The Vienna Development Method is an integrated and multifaceted method, very suitable for formal cyber-physical development. The maturity of the approach and effort of integrating VDM with co-simulation-based frameworks has made the approach more deployable, as it can be used with already popular development processes. The extension of the

VDM-SL specification language, VDM-RT, improved modelling of cyber-physical systems by making it possible to specify crucially important real-time system features. The tool is open-source and based on the Eclipse IDE that allows it to be easily extensible. However, there are some technical tool-related issues that are further discussed in (§3.8.3, [103]). One of the main issues is that a stepwise system development has not been implemented practically (Event-B supports) in the Overture tool. This can significantly increase modelling and, particularly, deductive verification support, which has been identified as inadequate in comparison to Event-B [103].

Overall, our analysis indicates that Event-B [56] has some important strengths and advantages that can be used as a base for the proposed formal engineering methodology. Firstly, it has been successfully used for a formal development of various distributed protocols [104–106] and hybrid systems [107, 108]. The Event-B method provides an expressive modelling language (and extensible [109]), flexible (and practical) refinement mechanism and is also proof driven, meaning model correctness is demonstrated by generating and discharging proof obligations with many available automated theorem provers [110, 111]. The method is supported by tools such as ProB [112] which enable animating and model checking a model and co-simulation extension [113]. However, the Event-B method does not have an adequate probabilistic reasoning support, which is essential for addressing stochastic aspects of heterogeneous railway systems.

# Chapter 3

# Development Methodology

This chapter proposes a formal engineering methodology for the top-down development of heterogeneous cyber-physical signalling systems. First of all, in the chapter, we describe the overall structure of the proposed methodology, which includes two formal development phases as well as modelling and verification tools of the methodology. In the second part of the chapter, we provide more detail on the process of formally developing heterogeneous signalling systems with the proposed methodology.

## 3.1 Methodology Overview

The proposed multifaceted methodology is a two-stage formal development methodology with the Event-B formal specification language at its core (illustrated in Figure 3.1). The main outcome of the methodology is a formal Event-B model of a heterogeneous signalling system, which is developed by refining a provided generic communication-based signalling Event-B model $\mathcal{M}$ with particular signalling system specifications (defined in the first step of the methodology). Safety system aspects are of the utmost importance and are ensured by mathematically proving an instantiated model with respect to user-defined safety requirements.

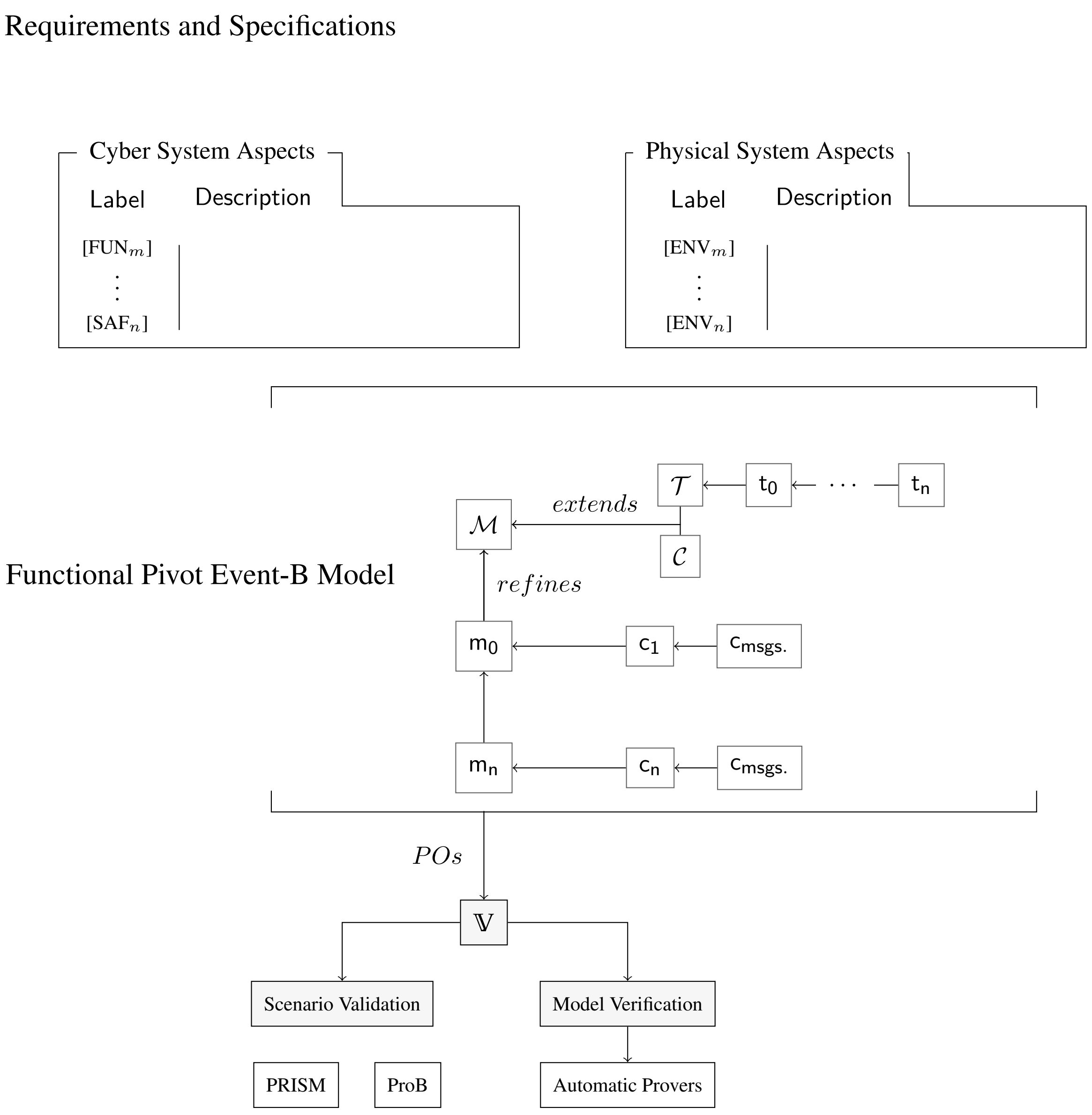


Figure 3.1: Formal modelling and verification methodology of heterogeneous signalling systems

The proposed development methodology is built upon an established Event-B formal modelling environment. The development of the functional Event-B model of a heterogeneous signalling system is facilitated by the Rodin platform, which provides a graphical user interface to develop models, automated proof obligation generation and verification tools as well as other available plug-ins. One of the essential Rodin extensions is the Theory plug-in, which enables extending the Event-B mathematical language with new mathematical theories. The developed generic communication-based hybrid signalling Event-B model depends on continuous system modelling theories and patterns introduced and implemented using the Theory plug-in by Dupont et al. [59]. As previously discussed, one of our main contributions was extending hybrid modelling theories [59] with railway specific aspects $\mathcal{T}$ using the Theory plug-in. The communication modelling patterns were developed and implemented outside of the Theory plug-in.

The second group of Rodin extensions our methodology relies upon are concerned with the verification and validation of a model. To support a deductive model verification, Rodin provides extensions to a number of automated provers (e.g. [111]) that attempt to discharge generated proof obligations automatically. Models developed in the Rodin platform can also be validated and animated using the ProB toolset, which can be particularly useful in early modelling stages where it could be too onerous to deductively verify a model. Furthermore, deductively verifying properties like deadlock-freedom or liveness in Event-B is challenging. Therefore, in some instances, ProB can provide a more pragmatic model checking based validation solution for verifying properties that are not concerned with safety. The current Rodin version does not provide a practical method for reasoning about stochastic system properties. In order to meet the requirement for our methodology, we utilised a well-known probabilistic model checker PRISM [114] and prove probabilistic properties outside of the Rodin environment.

By using the Event-B method, our methodology aims to demonstrate that the heterogeneous signalling system satisfies the formulated safety requirements.

## 3.2 Development Stages

In the previous section, we examined the overall structure of the proposed methodology which is visualised in Figure 3.1. In the following sections, we provide more detail on the methodological steps of system specification and functional Event-B model development.

### 3.2.1 System Specifications and Requirements

The generally accepted approach to any formal system development is starting the development process by informally describing system specifications and requirements. Therefore, the first development step in our methodology is defining system specifications and requirements concerned with the specific heterogeneous signalling system configuration (generic heterogeneous signalling system requirements are provided in Chapter 5).

**event_label** $\widehat{=}$ $[specification_{id}]$

**ANY**

parameters

**WHEN**

guards

**THEN**

actions

**INVARIANTS**

**invariant_label** invariant $[requirement_{id}]$

Figure 3.2: An example of an annotated Event-B model (event and invariant) in the Rodin tool

At this formal development stage, the proposed methodology requires labelling system specifications and requirements. By labelling requirements and specifications we intend to ensure traceability between informal and formal methodology artefacts. The traceability aspect guarantees the completeness of the model, meaning that all informal specifications and requirements have been captured by the formal model. On the formal Event-B artefact side, the methodology makes it necessary to annotate events and invariants of the Event-B model with labelled references to a specific informal artefact (Figure 3.2). The Rodin platform tools like ProR [115] or others [116, 117] exist for ensuring traceability between formal and informal artefacts but both are no longer supported or not realised as the Rodin tool extension.

Furthermore, the heterogeneous railway signalling model we consider is a hybrid model, meaning that system dynamics and physical aspects (e.g. rolling stock properties) must be defined. The generic theory of train dynamics we provide in a model theory $\mathcal{T}$ extension is parametrised and can be instantiated with specific constants (e.g. train mass, rail resistance).

### 3.2.2 Functional Event-B Model

The formal development of a heterogeneous signalling system Event-B model is the central activity of the methodology. For the formal modelling, the methodology provides a generic railway signalling system Event-B model, railway-related Event-B *theories* and Event-B patterns for modelling protocols. In order to derive a heterogeneous signalling system Event-B model, a developer is required to refine a provided generic Event-B railway model with specific signalling system configurations by using the Event-B refinement mechanism (depicted in Figure 3.3).

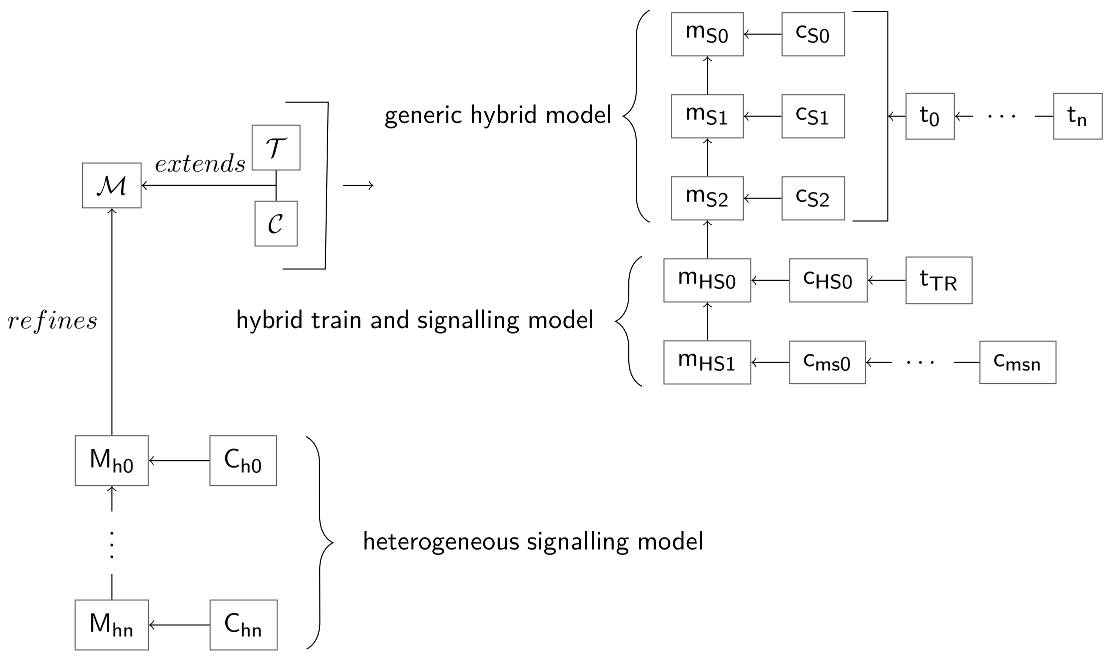


Figure 3.3: The main activity of the proposed methodology is refining a generic signalling Event-B model $\mathcal{M}$ with heterogeneous signalling system modelling machine M and context C models

The generic railway signalling system Event-B model (hybrid train and signalling model in Figure 3.3) was built using hybrid modelling patterns introduced by Dupont et al. [59]. Their method introduced an Ordinary Differential Equation (and others) Event-B theory which makes it possible to formally describe physical system properties in the Event-B model. Their method also provides Event-B events for modelling (continuous) plant and (discrete) controller behaviours, and a generic hybrid system Event-B model (generic hybrid model in Figure 3.3).

The generic railway signalling model provided by our methodology includes a new Event-B theory ($\mathcal{T}$ in Figure 3.3) which defines continuous rolling stock dynamics with a parametrised first-order non-linear differential equation. Furthermore, the railway model also provides the Event-B context model ($\mathcal{C}$ in Figure 3.3) with a pre-defined train stopping distance function, constants related to the engine traction effort and parameters for the equation of train dynamics (see Listing 3.1). The context model of the rolling stock can be further extended by specifying new parameters of the train physical model and updating train speed controller modes defined by axiom $axm_5$ (see Listing 3.1).

Listing 3.1: An Event-B context model of the hybrid train speed controller

**CONTEXT** Train
**EXTENDS** ControlledSystemCtx
**CONSTANTS**
  free_move, restricted_move
  StopDist
  a, b, c
  $f_{min}, f_{max}, f_{dec\_min}$
**AXIOMS**
  $axm_1$ $a, b, c \in \mathbb{R}^+$
  $axm_2$ $f_{min}, f_{max}, f_{dec\_min} \in \mathbb{R}$
  $axm_5$ partition(STATES, {free_move}, {restricted_move})
$\vdots$
**END**

The proposed development methodology enables extending a generic signalling system Event-B context model CommunicationCtx (see Listing 3.2) which introduces signalling sub-systems and field elements (e.g. communication centres, interlocking boxes, points) with signalling specifications defined in the first phase of the methodology. The system context model can be extended by including new axioms (e.g. constraining the number of trains) or sets. The process of extending the signalling context model with heterogeneous system aspects is visualised in Figure 3.4 as the first step.

Listing 3.2: An Event-B context model of the generic signalling system

**CONTEXT** CommunicationCtx
**SETS**
  TRN, CC, IXL, PNT, PSTATUS
**CONSTANTS**
  FREE, RESERVED
**AXIOMS**
  $axm_{1..4}$ : finite(TRN, CC, IXL, PNT)
  $axm_5$ : partition(PSTATUS, {FREE}, {RESERVED})

To model an inter-subsystem communication of heterogeneous railway signalling systems, our methodology provides Event-B modelling patterns for a systematic modelling of protocols with Event-B. For example, patterns make it possible to introduce signalling protocol messages and capture message sending/receiving events in the Event-B model (Steps 2 and 3 in Figure 3.4). Communication modelling patterns are templates for defining Event-B variables, machine events and context models. In the actual Event-B model templates are *instantiated*, or in other words, replaced with variables, events and context models, which represent the actual system and adhere to the rules and structure of the template.

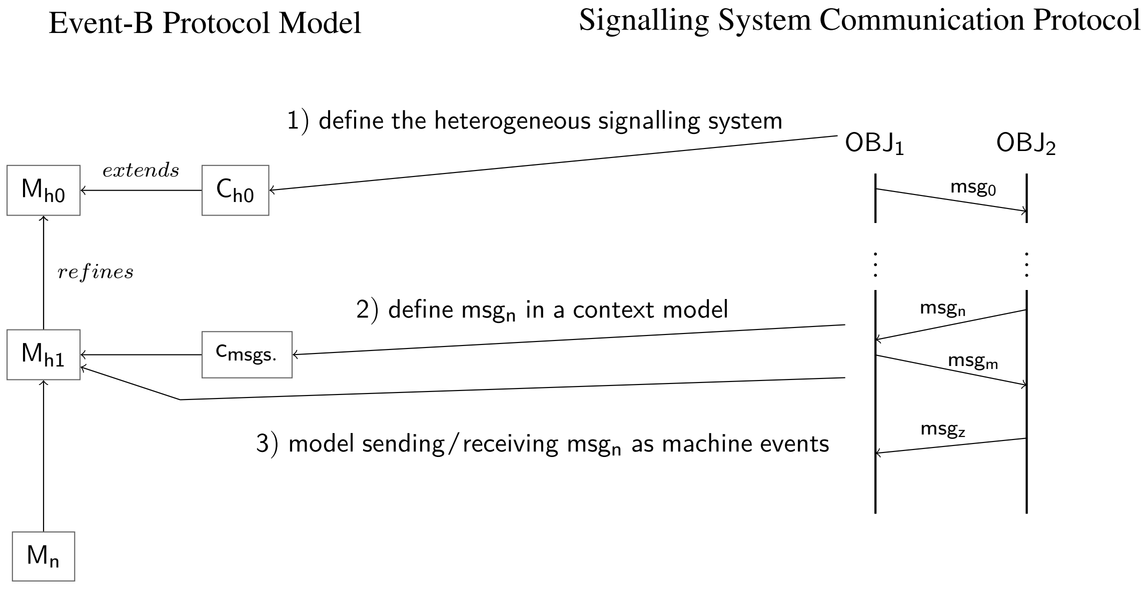

Event-B Protocol Model
Signalling System Communication Protocol
1) define the heterogeneous signalling system
$OBJ_1$
$OBJ_2$
$M_{h0}$
extends
$C_{h0}$
$msg_0$
refines
2) define $msg_n$ in a context model
$msg_n$
$M_{h1}$
$c_{msgs.}$
$msg_m$
$msg_z$
3) model sending/receiving $msg_n$ as machine events
$M_n$

Figure 3.4: The process of applying communication modelling patterns

The standard approach to modelling systems in Event-B is to start with an abstract system model and incrementally add more detail through a new Event-B machine and context files with refined events, new variables and invariants. The proposed development process follows a similar approach, where a provided abstract railway signalling model $\mathcal{M}$ shall be refined using one of the template events and communication modelling patterns. The refinement process for the communication part of the model follows a backward unfolding style, where the last protocol message exchanges are introduced into the model first. At each refinement step, a modeller is required to provide invariants about the system, which together with other proof conditions must be proved. The methodology primarily relies on deductive property verification using Rodin built-in and external automated provers. In the early model refinement phases, a deductive model verification might be too onerous, and so model checkers ProB and PRISM can provide a level of model correctness assurance.

## 3.3 Methodology Summary

In Section 2.1.4 we elicited the essential requirements for the formal development methodology of heterogeneous railway signalling systems. Two of the key requirements (**Requirement**$_1$ and **Requirement**$_3$) relate to the cyber-physical complexity of heterogeneous railway signalling systems, meaning that the system safety of these systems heavily depends on correct interactions between interlocking, communication centres and rolling stock. Therefore, these requirements necessitate the modelling of a whole system and reasoning about the system-level properties. The methodology proposes to addresses these requirements by building a formal Event-B model of a heterogeneous railway signalling system. Our evaluation indicated that the Event-B specification language can be used for system-level modelling. While, compared to simulation-based methods, Event-B requires the developer to have a mathematical background, systems formally developed using Event-B can provide a higher safety assurance, which is the main concern of this research.

The multi-aspect reasoning requirement **Requirement**$_4$ relates to the heterogeneous nature of cyber-physical railway signalling systems. The proposed methodology addresses this issue by extending Event-B language with hybrid modelling theories, which makes it possible to reason about continuous system aspects. For the stochastic property verification, our approach utilises the PRISM model checker. In Chapters 4 and 5, we provide technical contributions on modelling communication and hybrid heterogeneous signalling systems aspects.

# Chapter 4

# Communication Modelling Patterns

Distributed protocols are an essential and safety-critical element of heterogeneous railway signalling systems. Therefore, it is necessary that a formal system-level railway signalling model captures communication between different signalling sub-systems. In this chapter, we describe the communication modelling patterns of distributed protocols in Event-B. This chapter defines patterns for systematically introducing new messages into a model and modelling message exchanges between different signalling sub-systems. These modelling patterns will enable us to implement a generic heterogeneous railway signalling model (see Figure 2.6) and provide our multifaceted development methodology with a systematic communication modelling method for refining a generic signalling model. In this chapter, we provide general Event-B patterns, while the following chapters will describe how these patterns can be used for modelling communication-based and heterogeneous signalling systems.

## 4.1 Communication Modelling Patterns

The Event-B model is made of two key components: machines and contexts, which respectively describe dynamic and static parts of the system. The dynamic part of the model contains variables which are modified by machine events and invariants which constrain a model. The context contains modeller declared constants and associated axioms which can be made visible in machines. In the following sections, we describe generic Event-B modelling patterns for capturing communication aspects of a distributed protocol. The patterns will provide a systematic method for modelling distributed communication of heterogeneous signalling systems.

The communication modelling patterns consists of Event-B modelling patterns for defining communicating actors of a system, introducing protocol messages into a model through context models (and machine variables) and event patterns for modelling message exchanges. Following our modelling patterns, a communication protocol (on the right in Figure 4.1) is modelled in Event-B by first defining communicating objects ($\mathsf{OBJ}_1, \mathsf{OBJ}_2$) which, in the system, exchange messages. Then, all different protocol messages are formally defined in individual Event-B context models. In the last step message channels ($\mathsf{msg}_n$) and machine events modelling message sending/receiving are introduced. In this chapter Event-B patterns are described using generic types and sets (e.g. $\mathsf{OBJ}_1, \mathsf{msg}_n$). In the actual Event-B model of the protocol, these patterns would be *instantiated*, meaning that, generic variables, types and events would be replaced with variables, types and events, which represent an actual system and adhere to the structure of communication modelling patterns.

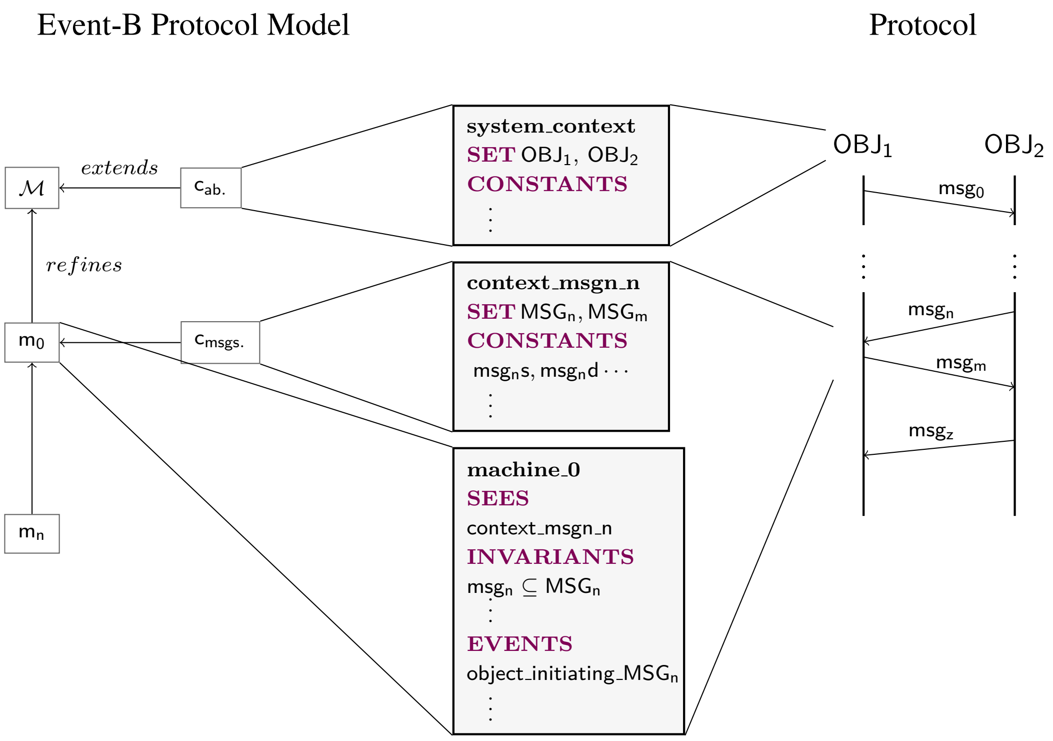


Figure 4.1: The communication modelling patterns

### 4.1.1 System Context Pattern

The first group of contexts (system_context) axiomatically defines all communicating objects present in the model (Listing 4.1). For example, communicating objects in the heterogeneous railway signalling systems are rolling stock, interlocking systems and communication centres (Section 2.1.3). An object is often required to go through a number of protocol steps to achieve its goal. Therefore, in addition to physical status (e.g. braking), some signalling objects can have a communication mode or status. In the context pattern we define a carrier set OBJ and enumerated set OBJ_STATUS to denote an abstract communicating object and a set of its statuses.

Listing 4.1: A context pattern for defining communicating objects

**CONTEXT**
  context_abstract
**SETS**
  $\mathsf{OBJ}, \mathsf{OBJ\_STATUS}$
**CONSTANTS**
  $\mathsf{STATUS}_1, \mathsf{STATUS}_2 \ldots \mathsf{STATUS}_n$
**AXIOMS**
  $\mathsf{partition}(\mathsf{OBJ\_STATUS}, \{\mathsf{STATUS}_1\}, \ldots \{\mathsf{STATUS}_n\})$

### 4.1.2 Message Modelling Patterns

The second context pattern is introduced to provide a general approach for introducing individual protocol messages into a model (Listing 4.2). In order to define the context pattern we first define a generic message type - MSG. In this context pattern, we introduce three constant functions for capturing source, destination and (optionally) value of a message type.

Depending on the message type, the source of the message in the protocol can be any of the objects defined in system_context. Therefore, in our message pattern we abstract all communicating objects to a general source - SRC. Similarly, for message destination constant function form, we give a general destination set - DST. In a protocol a message can carry a value, so the last generic set we define is VAL with associated constant function (msgv) necessary for extracting value from the message. All constant message functions are surjective ($\twoheadrightarrow$) functions meaning they are total in a domain and range. Lastly, each message type context contains an axiom, which states that there exists a suitable message between any source and destination with any possible value.

Listing 4.2: A context pattern of defining a new message

**CONTEXT**

$\mathsf{context\_message\_type}$

**EXTENDS**

$\mathsf{context\_abstract}$

**SETS**

$\mathsf{MSG}$

**CONSTANTS**

$\mathsf{msgs, msgd, msgv}$

**AXIOMS**

$\mathsf{msgs} \in \mathsf{MSG} \twoheadrightarrow \mathsf{SRC}$

$\mathsf{msgd} \in \mathsf{MSG} \twoheadrightarrow \mathsf{DST}$

$\mathsf{msgv} \in \mathsf{MSG} \twoheadrightarrow \mathsf{VAL}$

$\forall \mathsf{s, d, v} \cdot \mathsf{s} \in \mathsf{SRC} \land \mathsf{d} \in \mathsf{DST} \land \mathsf{v} \in \mathsf{VAL} \Rightarrow \exists \mathsf{m} \cdot \mathsf{msgs(m)} = \mathsf{s} \land \mathsf{msgd(m)} = \mathsf{d} \land \mathsf{msgv(m)} = \mathsf{v}$

Once a new message type context has been introduced, a message can be used in a machine model (Listing 4.3). The first step in introducing a new message into a machine is extending that machine with a message associated context. The following step is creating a three-letter variable msg (type $\mathsf{inv}_1$) to model the channel of that message. The message sending events will use channel variables to add and remove elements from a msg set to model the action of sending or receiving a message of type MSG. As messages are added and removed from the channel msg, we require a second generic local variable of type $\mathsf{inv}_2$ to locally store what messages have been sent.

Listing 4.3: A machine pattern for importing new messages and creating channels

**MACHINE**

$\mathsf{M}$

**SEES**

$\mathsf{context\_message\_type}$

**INVARIANTS**

$\mathsf{inv}_1 \quad \mathsf{msg} \subseteq \mathsf{MSG}$

$\mathsf{inv}_2 \quad \mathsf{msgo} \in \mathsf{OBJ} \rightarrow \mathbb{P}(\mathsf{MSG})$

### 4.1.3 Loop Modelling Pattern for Iterative Actions

In protocols, a communicating object is often required to send multiple messages or repeat some processes numerous times. For such scenarios we created a two-event loop pattern, which would generally be combined with message sending patterns to model burst message sending or objects' iterative actions (Listings 4.4 and 4.5).

Listing 4.4: A pattern for modelling iterative object actions (loop body part)

```
object_STATUS_b ≙
ANY
    ob
WHEN
    grd1      pctn(ob) = STATUS
    grd2..n   select_guards
THEN
    act1      (send_receive)_message
```

The first event in this pattern is the loop body event object_STATUS_b, where _b event name indicates loop body type. The STATUS and object in the event name would be modified to a corresponding communicating object and its loop starting status, for instance, train_REQUESTMA_b. The generic event pattern has a single parameter communicating object og and is enabled once the object's status variables ($grd_1$) becomes some STATUS, where STATUS is an element of OBJ_STATUS defined in context system_context. The remaining event guards ($grd_{2..n}$) would be used to further constrain the event (e.g. select a specific message to send/receive) while event action $act_1$ executes an iterative step.

Listing 4.5: A pattern for modelling iterative object actions (loop completion part)

```
object_STATUS_c ≙
ANY
    ob
WHEN
    grd1      pctn(ob) = STATUS
    grd2..n   loop_completed_guard
THEN
    act1      pctn(ob) := NEXT_STATUS
```

Another event in this pattern is a loop completion event with _c event name. As the name suggests this event detects when the iterative process has been completed. The loop_completed guard would be typically predicated on locally saved message receipt variable msgo. The action of this event simply updates objects' status to the next value.

### 4.1.4 Message Sending/Receiving Event Patterns

We identified two message sending event patterns. The most widely encountered communication situation is responding to a received message - a reply event type. We define a reply event pattern object_reply_MSG, where the main principle of this event is to take a message from one channel and create a response message in a different channel (Listing 4.6).

Listing 4.6: An event pattern for replying to a received message

```
object_reply_MSG  ≙
ANY
    $ms_1, ms_2$
WHEN
    grd1   $ms_1 \in msg_1$
    grd2   $ms_2 \in MSG_2 \setminus msg_2$
    grd3   $msg_2d(ms_2) = msg_1s(ms_1)$
    grd4   $msg_2s(ms_2) = msg_1d(ms_1)$
    grd5   $pctn(msg_1s(ms_1)) = STATUS$
THEN
    act1   $msg_2 := msg_2 \cup \{ms_2\}$
    act2   $msg_1 := msg_1 \setminus \{ms_1\}$
    act3   $msgo_2(msg_2s(ms_2)) := msgo_2(msg_2s(ms_2)) \setminus \{ms_2\}$
```

In this pattern, we use constant message functions defined in the context files to select the source and destination of the new message. The reply event pattern has two event parameters - messages $ms_1$ (received message) and $ms_2$ (reply message), where a message type must be defined. The first guard ($grd_1$) of this event states that a message $ms_1$ must have been sent or, in other words, is already an element of the channel $msg_1$. Guards $grd_{2..4}$ select an appropriate reply message $ms_2$ by using constant functions defined in the message contexts. They state that a reply message destination is the source of the received message and that the source of the new message is a destination of the received message. Guard $grd_5$ will only allow sending a reply message if the status of the source object is a specific status. Actions of this event create a new message by adding it to the channel variable ($\mathsf{act}_1$), removing a received message ($\mathsf{act}_2$) and saving the sent message's receipt locally.

Listing 4.7: An event pattern for sending an initiating message

```
object_initiating_MSG ≙
ANY
    $\mathsf{ms}_1$
WHEN
    $\mathsf{grd}_1$      $\mathsf{ms}_1 \in \mathsf{MSG}_1 \setminus \mathsf{msg}_1$
    $\mathsf{grd}_2$      $\mathsf{pctn}(\mathsf{msg}_1\mathsf{s}(\mathsf{ms}_1)) = \mathsf{STATUS}$
    $\mathsf{grd}_3$      $\mathsf{msg}_1\mathsf{d}(\mathsf{ms}_1) \in \mathsf{ran}(\mathsf{msg}_1\mathsf{d})$
    $\mathsf{grd}_{4..n}$   select_guards
THEN
    $\mathsf{act}_1$      $\mathsf{msg}_1 := \mathsf{msg}_1 \cup \{\mathsf{ms}_1\}$
    $\mathsf{act}_2$      $\mathsf{msgo}_1(\mathsf{msg}_1\mathsf{s}(\mathsf{ms}_1)) := \mathsf{msgo}_1(\mathsf{msg}_1\mathsf{s}(\mathsf{ms}_1)) \setminus \{\mathsf{ms}_1\}$
```

Another type of message sending event we define is an initiating message sending as formalised in Listing 4.7. The principle of this event is to create a new message once the status variable of an object changes to the specific status. This event pattern only has one event parameter $\mathsf{ms}_1$, denoting a new message which must not be already sent ($\mathsf{grd}_1$). The second event guard states that the program counter (pctn) of the source message must be at some STATUS defined in the abstract_context. The actions of the event add a new message to the message channel $\mathsf{msg}_1$ and save the message receipt locally.

## 4.2 Chapter Summary

In this chapter, we introduced Event-B communication modelling patterns, which was a necessary contribution towards our objective of the systematic modelling of communication-based heterogeneous signalling systems. The developed modelling patterns provide a standardised method for defining a protocol model, introducing new messages through context models and modelling message exchanges in the Event-B machine models. In addition, we defined a loop event modelling pattern which enables modelling an iterative process, such as sending multiple messages to different communicating objects.

The communication modelling patterns we introduced in this chapter were generic and could be applied for modelling any communication-based system. Nonetheless, in the following chapter we will demonstrate how communication modelling patterns could be applied for modelling communication-based railway signalling systems. In particular, we will present how communication and hybrid modelling patterns allow one to model and reason about heterogeneous railway signalling systems on a system-level.

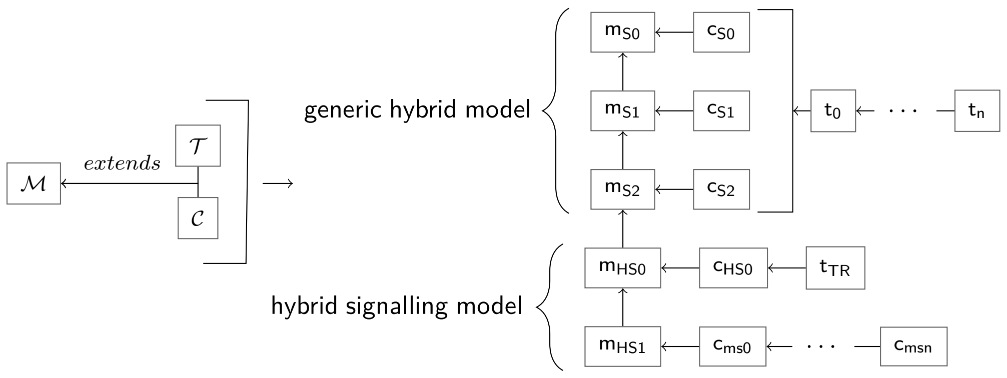
# Chapter 5

# Hybrid Signalling Modelling

Cyber-physical systems, such as railway signalling systems, exhibit both discrete and continuous behaviours, which are best captured by hybrid models. Hybrid models allow for the capturing of complex systems more accurately and thus provide a higher degree of system safety assurance. The safety of communication-based heterogeneous signalling systems heavily depends on the correctness of the rolling stock on-board system (hybrid system), which computes and controls the movement of the train. Therefore, it is important that our proposed development methodology (and the generic signalling model) provides patterns for modelling and reasoning about hybrid features of heterogeneous signalling systems.

The contributions of this chapter are Event-B modelling patterns for the formal development of railway signalling systems with discrete and continuous (hybrid) dynamics (the hybrid signalling model in Figure 5.1). The modelling patterns are then applied to defining a generic heterogeneous hybrid railway signalling model. Before we describe a railway specific modelling patterns, the chapter introduces the Event-B hybrid modelling theory [59] implemented by utilizing the Theory plug-in [93] of the Rodin platform (the generic hybrid model in Figure 5.1), which provides a formal foundation for defining hybrid signalling modelling patterns. Generic hybrid modelling patterns and a model consisting of the Event-B language extension theories ($\mathsf{t}_{0..n}$), which introduce the notion of continuous functions and ordinary differential equations, context models ($\mathsf{c}_{\mathsf{S}0..2}$) and abstract machine models ($\mathsf{m}_{\mathsf{S}0..2}$) with generic events for modelling discrete and continuous state-variable transitions. Furthermore, in order to formalise a hybrid railway signalling model we informally define a generic communication-based signalling model with a moving-block operation principle. The generalised communication-based signalling model relies on a simplified railway communication protocol between three main signalling sub-systems. Moreover, to facilitate hybrid railway signalling features of the model, we derive a rolling stock continuous dynamics model based on Davis' rolling stock resistance equation and a hybrid automata model of the train speed controller.

The hybrid signalling Event-B model refines the generic hybrid model and implements a generic communication-based signalling model. The generic hybrid signalling model includes additional machine models, which capture the speed controller of rolling stock ($\mathsf{m}_{\mathsf{HS0}}$) and system-level railway signalling system ($\mathsf{c}_{\mathsf{HS1}}$). The machine models are extended with context models ($\mathsf{c}_{\mathsf{HS0}}$, and $\mathsf{c}_{\mathsf{ms0..n}}$) and the generic train theory model ($\mathsf{t}_{\mathsf{TR}}$), which respectively define messages of the communication-based signalling system and rolling stock continuous dynamics.



extends
$\mathcal{M}$
$\mathcal{T}$
$\mathcal{C}$
generic hybrid model
$\mathsf{m}_{\mathsf{S0}}$
$\mathsf{c}_{\mathsf{S0}}$
$\mathsf{m}_{\mathsf{S1}}$
$\mathsf{c}_{\mathsf{S1}}$
$\mathsf{m}_{\mathsf{S2}}$
$\mathsf{c}_{\mathsf{S2}}$
$\mathsf{t}_0$
$\mathsf{t}_n$
hybrid signalling model
$\mathsf{m}_{\mathsf{HS0}}$
$\mathsf{c}_{\mathsf{HS0}}$
$\mathsf{t}_{\mathsf{TR}}$
$\mathsf{m}_{\mathsf{HS1}}$
$\mathsf{c}_{\mathsf{ms0}}$
$\mathsf{c}_{\mathsf{msn}}$

Figure 5.1: Structure of the hybrid railway signalling Event-B model. This figure provides a more detailed view of the functional pivot Event-B model (in Figure 3.1), which captures a generic hybrid railway signalling system.

## 5.1 Hybrid System Modelling Patterns in Event-B

Even though the Event-B mathematical language is expressive enough for a lot of useful mathematical concepts, it is still desirable to allow users to extend the language. For that reason a theory extension process has been developed and realised as a Rodin platform plug-in [93]. With the theory extension approach, new theories - which include data types, operators, and proof rules - can be defined and their correctness proved by discharging generated proof obligations.

A theory of real numbers is of particular interest for reasoning about hybrid systems properties. This work reuses the theory of dense reals. originally developed by Abrial and Butler [94] and then extended by Dupont et al. [59], to support the theory of continuous functions and ordinary differential equations. In this section, we overview the main modelling features of the [59] framework for hybrid systems in Event-B. We first describe a theory for differential equations, then expose a generic hybrid model as well as a methodology for deriving a specific controller.

To handle a hybrid system we need to employ both discrete and continuous concepts. If discrete parts are, in essence, *natively* supported by Event-B, this not really the case for continuous features. Listing 5.1 gives an excerpt of the theory defined and used throughout Event-B development.

Listing 5.1: Differential equation theory excerpt

**THEORY**
**TYPE PARAMETERS** $\mathsf{E}, \mathsf{F}$
**DATA TYPES**
$\quad \mathsf{DE}(F)$
**CONSTRUCTORS**
$\quad \mathsf{ode}(\mathsf{fun} : \mathbb{P}(\mathbb{R} \times \mathsf{F} \times \mathsf{F}),\ \mathsf{initial} : \mathsf{F},\ \mathsf{initialArg} : \mathbb{R})$
**OPERATORS**
$\quad \mathsf{solutionOf} < \mathsf{predicate} > (\mathsf{D_R} : \mathbb{P}(\mathbb{R}), \eta : \mathsf{D_R} \to \mathsf{F}, \mathsf{eq} : \mathsf{DE}(\mathsf{F}))$
$\quad \mathsf{Solvable} < \mathsf{predicate} > (\mathsf{D_R} : \mathbb{P}(\mathbb{R}),\ \mathsf{eq} : \mathsf{DE}(\mathsf{F}))$
**DIRECT DEFINITION**
$\quad \exists \mathsf{x} \cdot \mathsf{x} \in (\mathsf{D_R} \to \mathsf{F}) \wedge \mathsf{solutionOf}(\mathsf{D_R}, \mathsf{x}, \mathsf{eq})$
$\quad \vdots$
**END**

In particular, it defines the following operator and expressions:

– $\mathsf{ode}(\mathsf{F},\ \eta_0,\ \mathsf{t_0})$ is an ordinary differential equation (ODE) $\dot{\eta}(\mathsf{t}) = \mathsf{F}(\mathsf{t},\ \eta(\mathsf{t}))$ with initial condition $\eta(\mathsf{t_0}) = \eta_0$.

– $\mathsf{DE}(\mathsf{S})$ a set of differential equations value in $S$ (the continuous state space).

– $\mathsf{solutionOf}(\mathsf{D}, \eta, \mathsf{eq})$, with $\eta \in \mathsf{D_R} \to \mathsf{S}$ and $\mathsf{eq} \in \mathsf{DE}(\mathsf{S})$, predicate indicating that $\eta$ is a solution of eq (on domain D).

- $\mathsf{Solvable}(\mathsf{D}, \mathsf{eq})$, predicate indicating that there exists a solution to equation $eq$ on domain $D$.

The methodology for developing hybrid models in Event-B is based on a generic hybrid system model which is visualised in the diagram presented in Figure 5.2. This model is then refined to describe more specific systems. In the following paragraphs, we provides the features of this model, whose detailed description is given in [59].

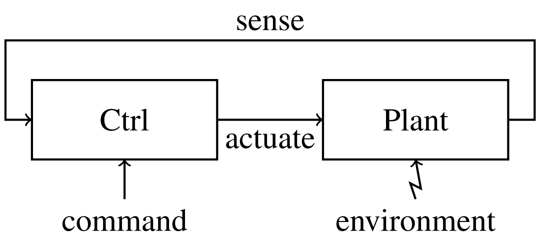


Figure 5.2: The generic hybrid system representation

**Variables and State Spaces.** A hybrid system is modelled through two variables: its discrete state $\mathsf{x_s}$ (usually corresponding to some mode in a mode automaton) and its continuous state $\mathsf{x_p}$, which is a function of time and valued in some *state space* $\mathsf{S}$ (usually a real vector space). As we will need it in later proofs or properties, we also model time with a single read-only variable, $\mathsf{t}$, which is *simulated* by the Progress event (see Listing 5.2).

Listing 5.2: A generic hybrid Event-B model: machine model structure

```
VARIABLES  t, x_p, x_s
INVARIANTS
   inv1 : t ∈ ℝ+
   inv2 : x_s ∈ STATES
   inv3 : x_p ∈ ℝ+ → S
EVENTS
   ⋮
Progress
THEN
   act1 :  t :| t < t'
```

The behaviour of hybrid systems is then modelled through four types of events. Transition events model internal decisions of the controller. This typically corresponds to discrete changes happening in the program or decisions made by the user (see Listing 5.3).

Listing 5.3: A generic hybrid Event-B model: Transition event

```
Transition
ANY s
WHERE
    grd1 : s ∈ ℙ1(STATES)
THEN
    act1 : x_s :∈ s
END
```

Sensing events represent changes in the controller induced by changes in the plant, typically detected through sensors. This event is similar to Transition except its guards generally involve $x_p$ (see Listing 5.4).

Listing 5.4: A generic hybrid Event-B model: Sense event

```
Sense
ANY s, p
WHERE
    grd1 : s ∈ ℙ1(STATES)
    grd2 : p ∈ ℙ(STATES × ℝ × S)
    grd3 : (x_s ↦ t ↦ x_p(t)) ∈ p
THEN
    act1 : x_s :∈ s
END
```

Behave events capture spurious changes in the plant, or *perturbations* in other words. This particular event modifies the continuous state ($x_p$) which is a function of time. To ensure coherence, we need to state that the *past* of the system remains the same; that is, the function does not change on $[0, t_{\mathbb{R}}[$. The *future* of the system ($[t, +\infty\mathbb{R}[$) is then set to be a solution of the given equation (see Listing 5.5).

Listing 5.5: A generic hybrid Event-B model: Behave event

```
Behave
ANY  e
WHERE
    grd1 : $e \in \mathsf{DE}(S)$
    grd2 : $\mathsf{Solvable}([t, +\infty[\,, eq)$
THEN
    act1 : $x_p :| x_p' \in \mathbb{R}^+ \to S \wedge [0, t[\, \lhd x_p' = [0, t[\, \lhd x_p \wedge \mathsf{solutionOf}([t, +\infty[\,, [t, +\infty[\, \lhd x_p', e)$
END
```

Actuation events model the action of the controller on the plant, achieved because of *actuators* (see Listing 5.6). This event is shaped on Behave from which it differs by the presence of additional guards to relate it to the controller's state.

Listing 5.6: A generic hybrid Event-B model: Actuate event

```
Actuate
ANY  e, s
WHERE
    grd1 : $e \in \mathsf{DE}(S)$
    grd2 : $\mathsf{Solvable}([t, +\infty[\,, eq)$
    grd3 : $s \subseteq \mathsf{STATES}$
    grd4 : $x_s \in s$
THEN
    act1 : $x_p :| x_p' \in \mathbb{R}^+ \to S \wedge [0, t[\, \lhd x_p' = [0, t[\, \lhd x_p \wedge \ \mathsf{solutionOf}([t, +\infty[\,, [t, +\infty[\, \lhd x_p', e)$
END
```

According to the modelling patterns, the generic model we described in the previous paragraphs is an entry point of a method for developing hybrid systems. Indeed, hybrid models should instantiate this generic model through the refinement steps.

## 5.2 Hybrid Railway Signalling Model in Event-B

In this section, we describe an abstract hybrid railway signalling Event-B model, which can be refined to capture a specific signalling configuration. First of all, this section revisits a generalised communication-based railway signalling model, including major railway signalling sub-systems and their communication relations. The section then describes continuous model aspects of the rolling stock which will be used to form a generic Event-B theory describing continuous behaviour of railway rolling stock.

### 5.2.1 Informal Communication-Based Railway Signalling Model

As previously discussed, we base our railway signalling model on the radio-based communication and in-cab signalling systems, which generally contain three sets of objects: trains, interlocking boxes and communication centres. On the infrastructure side, our railway model is made of railway tracks, which contain points (P1 in Figure 5.3) allowing trains to switch tracks and block markers ($\mathsf{M}_{1..3}$ in Figure 5.3) for marking a spatial beginning and ending of railway sections (blocks).

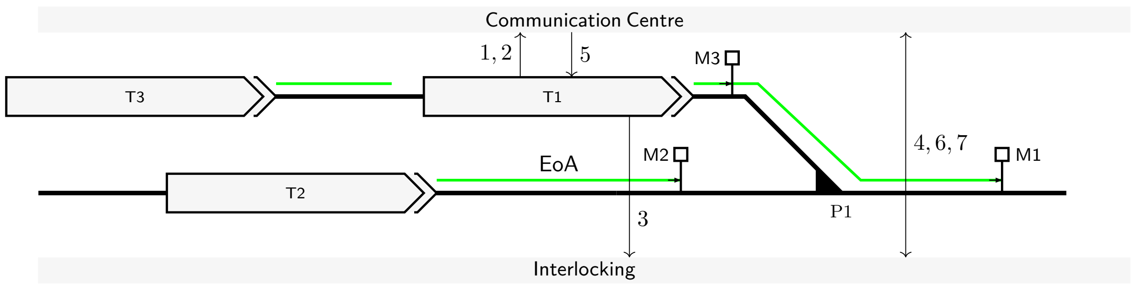


Figure 5.3: An example of the abstract railway signalling model with three trains

The objective of the abstract railway signalling model is ensuring a safe spatial separation of trains and preventing train derailment by guaranteeing only locked point crossing. Our abstract signalling model is based on a moving-block signalling principle with points protected with fixed-blocks in order to prevent a train derailment. In the following paragraphs, we specify the functionality of each object and communication relations with other objects (communication diagram of the signalling model is given in Figure 5.4).

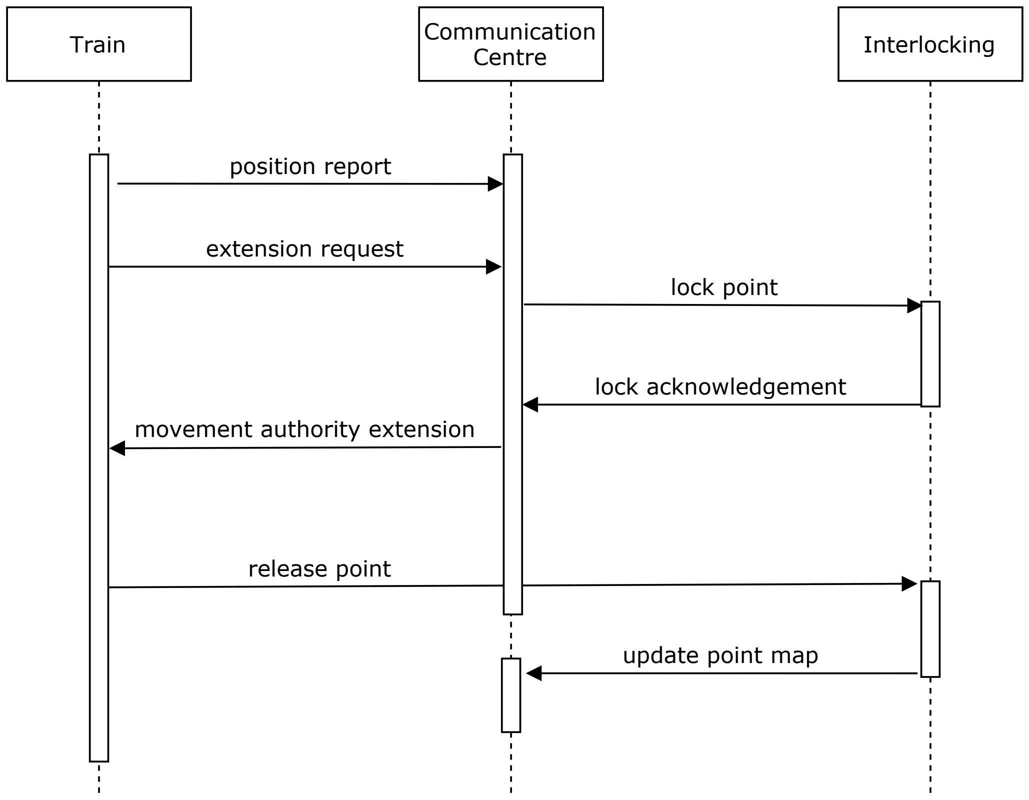


Figure 5.4: Sequence diagram of the signalling model

**Rolling Stock** In our signalling model, trains are modelled by their physical (continuous) state as well as the mode (discrete) of their on-board computer. The physical state of a train evolves according to some differential equations, with equation parameters controlled by the discrete mode of the on-board computer. The signalling model also assumes that a train is able to self-localise and communicate with other objects. In our abstract signalling model a train can send three types of messages:

1. A position report message is sent periodically to the communication centre to update its position.
2. An extension request message is sent to the communication centre when a train approaches the end of its movement authority (allowed travelling distance).
3. The train sends a release message to the interlocking informing that it has left the junction area (crossed points).

**Communication Centres** In communication-based signalling systems, a communication centre is a pivotal object which manages a part of a railway network by interacting with rolling stock and interlocking boxes. A communication centre uses information received from trains and interlocking boxes to issue allowed travelling distances to individual trains. A centre contains and continuously updates an internal railway network map with junction locations (also their status: free or locked) and rolling stock positions. The model assumes that a communication centre knows the destination of each train, so points can be locked in the correct direction. The communication centre can send the following messages:

4. When a communication centre receives an extension request message from a train with an extension path requiring locking railway points, it sends a lock point message to the interlocking (if that point status is free) to set a point to the right direction and lock it.
5. Once a point has been locked (or else it was not necessary), a communication centre sends a movement authority extension message to the train containing a permitted travelling distance which is computed by considering other trains and point positions.

**Interlocking Boxes** An interlocking is a safety-critical object responsible for authorising rolling stock and movement of infrastructure (e.g. points). In our signalling model, the function of an interlocking box is guaranteeing safety by preventing trains crossing the same junction at the same time (e.g. P1 in Figure 5.3). An interlocking can send the following messages to a communication centre:

6. An interlocking sends a lock acknowledgement message when it receives a lock point message to inform a communication centre that a point has been adjusted and locked.
7. An interlocking sends an update free map message to a communication centre when it receives a releasepoint message from a train indicating that a train left the locked junction and it can now be set to free.

### 5.2.2 Continuous Railway Signalling System Features

In this section, we describe the mathematical model of rolling stock continuous dynamics which will be used as a basis for modelling hybrid train dynamics in Event-B. The section begins by deriving a realistic acceleration differential equation from fundamental laws of physics and models of rolling bodies. In the following paragraph, we describe a hybrid train speed controller model which will be used in the generic railway signalling Event-B model.

$$\begin{cases} \mathsf{F_{net}} & = \mathsf{m} \cdot \mathsf{a} \vdash \mathsf{a} = \frac{\mathsf{F_{net}}}{\mathsf{m}} \vdash \mathsf{a} = \frac{\mathsf{f_{eng.}} - \mathsf{R_{tot.}}}{\mathsf{m}} \\ \mathsf{R_{tot.}(t)} & = \mathsf{a} + \mathsf{b} \cdot \mathsf{v(t)} + \mathsf{c} \cdot \mathsf{v(t)}^2 \\ \dot{\mathsf{v}}(\mathsf{t}) & = \frac{\mathsf{f(t)} - (\mathsf{a} + \mathsf{b} \cdot \mathsf{v(t)} + \mathsf{c} \cdot \mathsf{v(t)}^2)}{\mathsf{m}} \\ \dot{\mathsf{p}}(\mathsf{t}) & = \mathsf{v(t)} \end{cases} \tag{5.1}$$

A driver or an automated train operation system can only control the train's engine power (tractive force) which in fine yields an acceleration. From Newton's second law we know that acceleration is proportional to a net force (tractive engines force) applied to the mass of that object. The train must also overcome a resistance force, which acts in the opposite direction to the engine's traction force and thus a total engine's tractive force can be expressed as the difference between two forces. The total rolling stock resistance is comprised of the mechanical and air resistances and is commonly expressed as a second-order polynomial (Davis Resistance equation $\mathsf{R_{tot.}(t)}$ in Equation 5.1), where $\mathsf{a}, \mathsf{b}$, and $\mathsf{c}$ are fixed parameters and $\mathsf{v(t)}$ is the speed of a train at time $\mathsf{t}$ [118].

The train speed controller we consider is continuously issued with the end of movement authority (EoA), which is updated periodically by the communication centre. We assume that the speed controller is able to sense its distance to the EoA and, in particular, determine if - with the given current speed and acceleration - it can stop before EoA. The stopping distance calculus is generally done using a complex algorithm in the on-board computer, whereas in our train model, we abstract the algorithm by a stopping distance function (StopDist) which takes the current acceleration and speed as parameters and returns a distance.

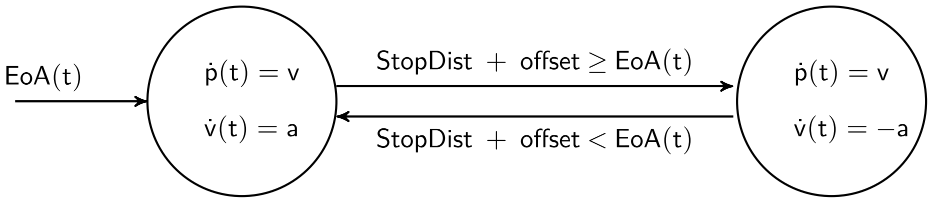


Figure 5.5: An abstract train speed controller hybrid automata model with two modes

The train speed controller has two modes: free mode and restricted mode. If the stopping distance (plus a safety offset) of the train is shorter than the EoA, then the train is said to be in a free mode and it can choose arbitrary values for $f_{eng.}$. Once, the stopping distance (+offset) of the train becomes shorter than the EoA, the train enters a restricted mode in which it is required to provide values for $f_{eng.}$ such that it can stop before the EoA. The train speed controller hybrid automata model is visualised in Figure 5.5. In the following sections, we present a formal Event-B implementation of the informally presented hybrid signalling model.

Listing 5.7: An excerpt of the signalling Event-B model: train model theory

```
THEORY Trains
OPERATORS
    DavisResistance  < expression >  (a : ℝ, b : ℝ, c : ℝ)
    DavisFunction  < expression >  (a : ℝ, b : ℝ, c : ℝ)
    DavisEquation  < expression >  (a : ℝ, b : ℝ, c : ℝ, f : ℝ)
ARGUMENTS
    a, b, c, f
WELL−DEFINEDNESS CONDITION
    0 ≤ a, b, c
DIRECT DEFINITION
    ode(DavisFunction(a, b, c, f), (v0 ↦ p0), t0)
THEOREMS
  DavisResistanceContinuity
    ∀a, b, c, n · a ∈ ℝ ∧ b ∈ ℝ ∧ c ∈ ℝ ∧ a ≥ 0 ∧ b ≥ 0 ∧ c ≥ 0 ∧ n ∈ ℕ
                                        ⇒ DavisResistance(a, b, c) ∈ Cn(n, ℝ, ℝ)
    ⋮
END
```

### 5.2.3 Hybrid Railway Signalling System Model: Theory and Context

As the behaviour of the train is generally common between several models, it was decided to create a reusable Trains theory (Listing 5.7). This theory defines the Davis equation $\mathsf{DavisEquation}(a, b, c, f, t_0, tv_0, tp_0)$ with initial conditions $tv(t_0) = tv_0$, $tp(t_0) = tp_0$. In the theory file, we also define other related theorems on the mathematical properties of this equation that will be useful for completing proofs.

In addition to a train dynamics theory model, we introduced a context model (see excerpt in Listing 5.8) which defines several constants of the system as well as constraints upon them. First of all, the Train context model axiomatically defines frictions parameters of the Davis equation, $a, b$, and $c$, as positive real number constants. Similarly, the context model also introduces the engine's traction power as real number constants: minimum power ($f_{min}$), maximum power ($f_{max}$), and minimum deceleration power ($f_{dec\_min}$ - a minimum strength with which the train can decelerate). Furthermore, we define a train's stopping distance function StopDist as a partial function of the current speed ($\mathbb{R}$) and acceleration ($\mathbb{R}$), with associated function constraining axioms. Finally, Train context defines train controller modes free_move and restricted_move, by refining the STATES set in ControlledSystemCtx with an enumerated set.

Listing 5.8: An excerpt of the signalling Event-B model: system context

```
CONTEXT Train
EXTENDS ControlledSystemCtx
CONSTANTS
    free_move, restricted_move
    StopDist
    a, b, c
    $f_{min}, f_{max}, f_{dec\_min}$
AXIOMS
    axm1  $a, b, c \in \mathbb{R}^+$
    axm2  $f_{min}, f_{max}, f_{dec\_min} \in \mathbb{R}$
    axm3  $StopDist \in (\mathbb{R} \times \mathbb{R}^+) \pfun \mathbb{R}^+$
    axm4  $StopDist(0 \mapsto 0) = 0$
    axm5  $partition(STATES, \{free\_move\}, \{restricted\_move\})$
       ⋮
END
```

### 5.2.4 Hybrid Railway Signalling System Model: Machine Events

In the first refinement of the generic hybrid model, we introduce several new events which instantiate generic events presented in Section 5.1. Because of the similarity of events, we only provide a single event for each of the generic event type. As specified in the previous subsection, in this refinement we model the speed controller where the end movement authority is continuously updated without specifying how at this level of abstraction.

Listing 5.9: Event restricting train's movement (transition type)

```
Transition_restricted_move
REFINES  Transition
WHERE
    grd1 : $x_s = \mathsf{restricted\_move}$
WITH
    s : $s = \mathsf{restricted\_move}$
THEN
    act1 : $f :\mid tp(t) + StopDist(f' \mapsto tv(t)) \leq EoA$
END
```

The Transition_restricted_move (refer to Listing 5.9) event models the change in the speed controller by adjusting the train's traction effort when the train is in the restricted move mode. The event is simply guarded by a single predicate which enables the event if and only if the status variable $x_s$ is set to restricted_move. To control the train's speed we created a variable $f$ which denotes the traction force and is modified by the action such that the stopping distance would not overshoot the end of the movement authority. One must then prove an open proof obligation that such a traction force value can be found.

Another internal controller event which changes controller's mode based on the input from the plant is sense event - Sense_to_restricted (refer to Listing 5.10). One out of the two sense events will change the train state variable $x_s$ if the end of movement authority has not been extended and the train must decelerate in order to remain within issued movement authority.

Listing 5.10: Event that captures train controller switching to a restricted_move mode

```
Sense_to_restricted
REFINES Sense
WHERE
    grd1 : $tp(t) + \mathrm{StopDist}(ta(t) \mapsto tv(t))) \geq EoA$
WITH
    s : $s = restricted\_move$
    p : $p = STATES \times \mathbb{R} \times \{v^* \mapsto p^* \mid p^* + \mathrm{StopDist}(f_{dec\_min} \mapsto v^*) \geq EoA\}$
THEN
    act1 : $x_s := restricted\_move$
END
```

As previously described, the other set of events define the evolution of the plant. The event provided in Listing 5.11 models the plant of the controller influenced by an environment. The first event modifies plant variables based on *some* solvable differential equation which describes the environment.

Listing 5.11: Event modelling environment induced changes to the train plant model

```
Behave
REFINES Behave
ANY e
WHERE
    grd1 : $e \in DE(S)$
    grd2 : $Solvable([t, +\infty], e)$
WITH
    $x_p'$ : $x_p' = bind(tv', tp')$
THEN
    act1 : $tv, tp :\mid tv' \in \mathbb{R}^+ \to S \wedge tp' \in \mathbb{R}^+ \to S \wedge [0, t[ \lhd tv' = [0, t[ \lhd tv \wedge$
        $\wedge [0, t[ \lhd tp' = [0, t[ \lhd tp \wedge solutionOf([t, +\infty[, [t, +\infty[ \lhd bind(tv', tp'), e)$

END
```

Similarly, the Actuate_move (Listing 5.12) event updates plant variables $tp, tv$. However, in this instance, variables are updated based on the Davis equation with traction force variable $f$ value as one of the parameters.

Listing 5.12: Event updating train's plant (actuate type)

Actuate_move
**REFINES** Actuate
**WHERE**
  $\mathsf{grd1} : \top$
**WITH**
  $x'_p : x'_p = \mathrm{bind}(tv', tp')$
  $e : e = \mathrm{DavisEquation}(a, b, c, f, t, tv(t), tp(t))$
  $s : s = \mathrm{STATES}$
**THEN**
  $act_1 : tv, tp :| \; tv' \in \mathbb{R}^+ \to S \wedge tp' \in \mathbb{R}^+ \to S \wedge \; [0, t[ \lhd tv' = [0, t[ \lhd tv \wedge [0, t[ \lhd tp' = [0, t[ \lhd tp \wedge$
    $\mathrm{solutionOf}([t, +\infty[, \; [t, +\infty[ \lhd \mathrm{bind}(tv', tp'), \; \mathrm{DavisEquation}(a, b, c, f, t, tv(t), tp(t)))$
**END**

### 5.2.5 Hybrid Railway Signalling System Model: Communication Aspects

To introduce communication aspects of the signalling system, the train speed controller model was further refined by introducing new events (based on patterns) to capture message exchanges between different objects. But, firstly, we created a new context (Listing 5.13) where new finite carrier sets representing trains, communication centres, interlockings, and points were introduced.

Listing 5.13: Communication signalling model context

**CONTEXT** CommunicationCtx
**SETS**
  TRN, CCS, IXL, PNT, PSTATUS
**CONSTANTS**
  FREE, RESERVED
**AXIOMS**
  $axm_{1..4} : \mathrm{finite}(\mathrm{TRN}, \mathrm{CCS}, \mathrm{IXL}, \mathrm{PNT})$
  $axm_5 : \mathrm{partition}(\mathrm{PSTATUS}, \{\mathrm{FREE}\}, \{\mathrm{RESERVED}\})$

In the following paragraphs, we provide a communication model excerpt which captures a train requesting movement authority extension and a radio-block responding. Following the pattern for each type of message, an individual context will be introduced which defines the new message type and contains constant functions, as in the context excerpt message_ma_extension (Listing 5.14). The constant message functions are used to select an appropriate message which includes a source, destination, and value of the message. As the message_ma_extension message is sent via radio-block centre to the train exts and extd function ranges are with respect to radio-block centre (source) and train (destination).

Listing 5.14: Movement authority extension message context

**CONTEXT** message_ma_extension
**SETS**
  EXT
**CONSTANTS**
  exts, extd, extv
**AXIOMS**
  $axm_1 : exts \in EXT \twoheadrightarrow CCS$
  $axm_2 : extd \in EXT \twoheadrightarrow TRN$
  $axm_3 : extv \in EXT \nrightarrow \mathbb{R}^+$
  $axm_4 : \forall s, d, v \cdot s \in CCS \land d \in TRN \land v \in \mathbb{R}^+ \;\Rightarrow\; \exists m \cdot exts(m) = s \land extd(m) = d \land extv(m) = v$

In the dynamic model (machine) part, we introduce a communication channel for the new messages which includes one for the extension message ext. The model excerpt below also contains another channel req for the movement authority extension messages and a variable reqt. The latter variable is simply used to track sent messages *locally* as messages are added and removed from the channel. Having communication channels introduced, events can be refined to incorporate message sending. We also, introduced new continuous variables, replacing $\{f, tp, ta, tv\}$, to capture multiple trains in the system ntp, nta, ntv.

Listing 5.15: Movement authority channel variables for messages

**MACHINE** SignallingModel
**SEES** CommunicationCtx
**VARIABLES**
  ext, req, reqt  ...
**INVARIANTS**
  $inv_1 : ext \subseteq EXT$
  $inv_2 : req \subseteq REQ$
  $inv_3 : reqt \in TRN \rightarrow \mathbb{P}(REQ)$
  ⋮
**END**

The Trains_sense_to_restricted event (Listing 5.16) models the controller state change once the train enters the area where it must decelerate to stay within the movement authority. In order for the train to continue the travel, it must request the movement authority extension by sending a message to the radio-block centre. We model that by refining the event, first adding additional event parameters denoting a train tr, radio-block centre rb, and a request message rq. Then we include new guards $grd_{2..6}$ which define variable types and state that rq must be a new message with a destination to specific radio-block centre rb and source of the train of interest rq. In the message we also include the current position of the train which, in this refinement, can be accessed from position function $ntp(tr)$ (due to multiple trains). New actions of the events $act_{2..3}$ add the new message to the request message channel req and saves a sent message receipt locally. This event was extended based on the *initiating* message sending event pattern.

Listing 5.16: Refined event for requesting *EoA* extension

```
Trains_sense_to_restricted
REFINES Sense_to_restricted
ANY
    tr, cc, rq
WHERE
    $grd_1 : tr \in TRN$
    $grd_2 : ntp(tr)(t) + StopDist(nta(tr)(t) \mapsto ntv(tr)(t)) \geq nEoA(tr)$
    $grd_3 : cc \in CCS$
    $grd_4 : rq \in (REQ \setminus req)$
    $grd_5 : reqd(rq) = cc$
    $grd_6 : reqs(rq) = tr$
    $grd_7 : reqv(rq) = ntp(tr)(t)$
THEN
    $act_1 : nx\_s(tr) := restricted\_move$
    $act_2 : req := req \cup \{rq\}$
    $act_3 : reqt(tr) := reqt(tr) \cup \{rq\}$
END
```

To model movement_authority_extension message sending we apply reply message modelling pattern which essentially adds a new message to the channel and removes the replied message. In order to avoid having a single *super-event*, it was decided to split message sending into two events which respectively model movement authority extension over a line and junction. In Listing 5.17 we provide an event for movement extension over a line. Firstly, the guards ($grd_{1..6}$) check if an adequate request rq message has been received and a new extension message ex will be sent to the source train $\mathsf{grd}_5$. The value of the reply message contains a distance value which is shorter than the distance to the next train and point ($\mathsf{grd}_{8..9}$), but greater than the current end of the movement authority ($\mathsf{grd}_7$). The event actions simply create a new extension message and remove the responded message. A similar event was created to capture the second branch when a point must be locked by communicating with an interlocking.

Listing 5.17: Event for modelling radio-block centre reply to the extension_request message (line part)

```
Radio_block_centre_extension_1
ANY
    rb, rq, ex
WHERE
    $grd_1 : rb \in RBC$
    $grd_2 : rq \in req$
    $grd_3 : ex \in (EXT \setminus ext)$
    $grd_4 : reqd(rq) = rb$
    $grd_5 : extd(ex) = reqs(rq)$
    $grd_6 : exts(ex) = reqd(rq)$
    $grd_7 : extv(ex) > nEoA(reqs(rq))$
    $grd_8 : extv(ex) < ppos(pmap(ntp(tr)))$
    $grd_9 : \forall tr \cdot tr \in TRN \wedge tr \neq reqs(rq) \Rightarrow extv(ex) < ntp(tr)(t)$
THEN
    $act_1 : ext := ext \cup \{ex\}$
    $act_2 : req := req \setminus \{rq\}$
END
```

The final event in the message exchange cycle for requesting movement authority extension is the refined event Trains_transition_eoa (Listing 5.18). Originally this event was introduced in modelling train speed controller to abstractly represent the internally updated movement authority. To introduce a communication aspect to this event we extend it with additional guards to check if an extension message ex has been received to the specific train tr ($\mathsf{grd}_{3..5}$). The new end of the movement authority value (new_EoA) is constrained by the value of extension message ($\mathsf{grd}_6$). The action of the event simply update trains end of authority variable $\mathsf{EoA(tr)}$ and deletes the extension message ex.

Listing 5.18: Refined transition event with updated *EoA*

```
Trains_transition_eoa
REFINES Transition_eoa
ANY
    tr, ex
WHERE
    grd_1 : tr ∈ TRN
    grd_2 : ex ∈ ext
    grd_3 : extd(ex) = tr
THEN
    act_1 : nEoA(tr) := extv(ex)
    act_2 : ext := ext \ {ex}
END
```

### 5.2.6 Proving Generic Hybrid Signalling Model: Proof Statistics

The generic model is proved once and for all and, in this work, we refined the abstract hybrid controller model. The hybrid railway signalling model is itself a reusable artefact, which could be further refined to capture a specific signalling configuration (e.g. by defining a specific railway schema) or modelling railway protocols (e.g. signalling handover).

As discussed by Alur in [22], the verification of hybrid systems remains a challenge. The verification approaches based on the reachability analysis aim to provide a fully automatic verification approach, but these approaches are limited to linear systems due to the real-valued variables. In this and other related works, authors have tried to address the verification scalability problem by developing alternative proof-based verification approaches. Still, as our verification results (Table 5.1) and also similar works [59, 61, 119] suggest, the proof automation of hybrid Event-B models is still low and must be improved for more practical applications. In spite of improved verification tools, a refinement plan for hybrid models should be reconsidered. Perhaps a top-down approach (particularly for this model), where continuous system's aspects are introduced in later refinement steps, is more suitable.

An important feature of this hybrid system development approach is the requirement of explicitly stating the system's dynamic properties. In our opinion, this problem is often overlooked and could lead to miscommunication between, for instance, control engineers and software engineers. With our proposed approach, a formal hybrid artefact is created, which

| Model | \|POs\| | Auto. | Inter. |
|---|---|---|---|
| hybrid_model (4m. + 1c.) | 61 | 23 | 38 |
| communication_model (1m. + 8c.) | 49 | 18 | 31 |
| Total | 110 | 41 | 69 |

Table 5.1: Proof statistics of the communication-based signalling Event-B model

can be used between different types of engineers. On the other hand, the approach currently requires some understanding of formal methods (e.g. mathematical syntax) and could benefit with connection to more widely used visual tools like Simulink or other similar tools via Functional Mock-Up Interface [36].

## 5.3 Chapter Summary

In this chapter, we introduced a generic hybrid railway signalling Event-B model, which was conceptually based on the in-cab railway signalling systems with a moving-block operation and implemented using the generic hybrid modelling patterns developed by Dupont et al. [59]. The chapter then described the method for extending the generic signalling model using previously introduced communication modelling patterns in order to capture heterogeneous signalling systems. In order to demonstrate safety of the generic signalling model, we formally proved collision freedom and plant constraint invariants by using available automated theorem provers and solvers of the Rodin platform. The generic hybrid signalling model is the key element of the proposed formal development methodology and the basis for the functional pivot Event-B model of the hybrid railway systems presented in Figure 3.1. The model is extensible and can be further refined to model a specific heterogeneous (or homogeneous) signalling configuration with communication modelling patterns.

In the following chapter (case study 2), we will demonstrate how a generic railway signalling model can be used to implement a specific heterogeneous signalling configuration. The case study captures a scenario, where multiple following trains are transitioning from one signalling area to another and safe train separation must be maintained during the transition.

# Chapter 6

# Methodology Evaluation

In this chapter, we present two railway signalling systems formally developed by using the proposed multifaceted methodology. The main objective of this chapter is to evaluate the new methodology and its applicability to formally developing cyber-physical railway signalling systems concerning the methodology requirements discussed in Section 2.1.4. It is important to note that even though the general formal development process in both studies follows the development process described in our methodology, due to the nature of a signalling system some divergence was required in the first case study. This formal development divergence in the first case study is further explained in the following paragraph.

In the first case study, we present a novel distributed resource allocation protocol developed to facilitate a decentralised railway signalling concept. The main objective of this case study is to evaluate the proposed formal development methodology when a signalling system is in the early (prototyping) development stage with incomplete requirements and specifications. Furthermore, this case study will also enable us to evaluate particular elements of the methodology, specifically, Event-B communication modelling patterns. On the other hand, the decentralised railway signalling case study did not model rolling stock and focused on applying the methodology to develop a new type of signalling (protocol level) system. Therefore, a generic hybrid communication-based signalling system model was not used in this study, as otherwise required by the proposed methodology. The proposed methodology was applied to developing a decentralised railway signalling concept in the following way:

1. According to the proposed methodology, initial specifications and requirements of the decentralised railway signalling concept are specified. (Section 6.1.1)
2. According to the proposed methodology, the initial functional pivot Event-B model of the decentralised railway signalling system is developed.
   (a) The initial functional pivot Event-B model was animated and attempted to be formally proved but problems with the initial protocol have been discovered (Section 6.1.2).
   (b) The initial decentralised railway signalling concept is refined with a new resource locking mechanism (specifications) and additional requirements (Section 6.1.3).
   (c) The refined decentralised railway signalling concept is specified in a semi-formal notation for a better signalling system concept translation to a formal Event-B model.
3. According to the proposed methodology, the refined (2c) functional pivot Event-B model of the decentralised railway signalling system is developed and formally proved.
4. According to the proposed methodology, the final decentralised railway signalling system was stochastically analysed using the PRISM model checker.

In the second case study, we model a train signalling transition (refer to Section 2.1.1) in a heterogeneous signalling system with interfacing moving and fixed block signalling systems. The overall objective of the second study is to evaluate the proposed methodology and its applicability to formally developing and verifying the safety of heterogeneous signalling systems. In this case study, we are particularly interested in evaluating our proposed process of extending the generic communication-based signalling system model with specific heterogeneous signalling system configurations. Unlike in the first case, the requirements and signalling specifications in this case study are fully known. The proposed methodology was applied to developing a heterogeneous railway signalling system in the following way:

1. According to the proposed methodology, specifications and requirements of the heterogeneous signalling system are detailed (Section 6.2.1).
   (a) System Specifications: heterogeneous system model with assumptions and system-level transition protocols are described informally (transition protocol semi-formally).

   (b) System Requirements: A heterogeneous system's safety requirements are expressed in a format suggested by the methodology.

2. According to the proposed methodology, the functional pivot Event-B model of the heterogeneous railway signalling system is developed (Section 6.2.2).

   (a) A context model HetModelContex, which extends context model CommunicationCtx of the generic communication-based signalling model, is introduced. The context model defines static elements of the heterogeneous signalling system described in the previous development step.

   (b) As according to the communication modelling patterns, each communication message described in the system specification step is introduced into the functional pivot Event-B model via a context model.

   (c) The communication-based signalling Event-B model is extended with an additional machine model, which models all communications described in the system specification step.

   (d) The safety requirements are expressed in the new machine model as invariants and formally proved.

In the last section of this chapter, we summarise both case studies, discuss the applicability and the limitations of the proposed methodology to formally developing heterogeneous railway signalling systems with respect to the evaluation criteria.

## 6.1 Case Study 1: Distributed Resource Allocation Protocol

In recent years, there have been proposals to utilise distributed system concepts (e.g. [120, 121]) for railway signalling as an approach to addressing network capacity and maintenance cost problems. These emerging distributed railway signalling concepts have proposed to use a radio-based communication technology to decentralise today's signalling systems. Nonetheless, because of the complex concurrent behaviour, distributed systems are notoriously difficult to validate that could curtail the development and deployment of novel distributed signalling solutions. In this case study, we present work that uses the proposed multifaceted methodology to formally develop and verify a distributed railway signalling protocol which could deliver decentralised signalling benefits while meeting high safety requirements. The developed distributed signalling protocol is based on serialisability and inspired from the one defined from transaction processing [122–124] in centralised and distributed database systems. The main objective of the protocol is guaranteeing mutual exclusion of railway sections (safety) while ensuring liveness of a system.

The model of our distributed railway signalling system relaxed some assumptions on message ordering, or in other words, allowed message delays in the model (contrary to other related work [121,125]). However, relaxed assumptions on the communication environment complicated the verification of safety and liveness properties of the distributed railway section allocation protocol. Nonetheless, the final protocol model was proved to guarantee the safety of agents while ensuring the probabilistic liveness of the system. In the following sections, we first describe a system model, assumptions and key safety and liveness requirements. Then, we demonstrate how safety and liveness properties can be violated (due to message delays) without a more complicated resource allocation protocol, which is then described in a semi-formal notation in Section 6.1.3. After describing resource allocation protocol semi-formally, we describe the developed functional pivot Event-B model of the protocol and its verification. In the final sections of this case study, we describe the verification of protocol probabilistic termination.

### 6.1.1 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 (resource 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 the order of agent allocation 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. In Table 6.1, high-level distributed system specifications are elicited in the format, which is suggested by the proposed methodology.

| | |
|---|---|
| $\text{SYS}_1$ | The distributed railway system is made of agents and resources (respectively representing trains and railway subsections). |
| $\text{SYS}_2$ | Agents can only communicate with resources and not other agents. |
| $\text{SYS}_3$ | An agent has an objective, which is a set of resources an agent has to reserve before proceeding with the next objective. |
| $\text{SYS}_4$ | Agents and resources have a memory in which sent and received messages are stored. |
| $\text{SYS}_5$ | A resource has a queue-like memory which contains the order in which agents will be allocated that resource. |
| $\text{SYS}_6$ | A resource has a promised pointer (ppt) variable, which indicates a currently available resource allocation slot number. |
| $\text{SYS}_7$ | A resource has a read pointer (rpt) variable, which specifies an agent which is currently using that resource. |
| $\text{ENV}_1$ | Message delays are permitted in the distributed signalling system model. |

Table 6.1: High-level distributed signalling system specifications

The main objective of the protocol is to enable safe and deadlock-free distributed atomic allocation of collection of resources. When using the term *safe resource allocation*, we mean that no two different agents have been allocated 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 Table 6.2. The following section attempts to justify the need for an adequate distributed protocol by discussing problematic resource allocation scenarios.

| | |
|---|---|
| $\text{SAF}_1$ | A resource will not be allocated to multiple agents at the same time. |
| $\text{SAF}_2$ | An agent will not use a resource until all requested resources are allocated. |
| $\text{LIV}_1$ | An agent must be eventually allocated a set of resources it has requested. |

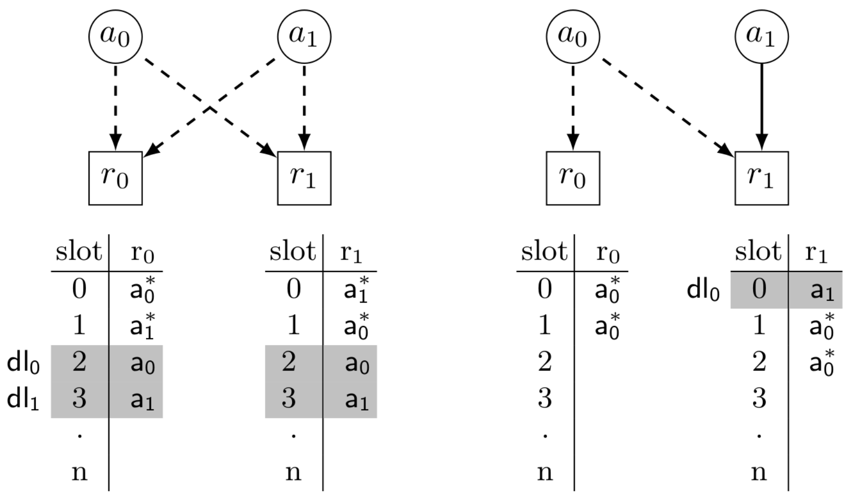
Table 6.2: High-level system safety and liveness requirements

### 6.1.2 Problematic Distributed Resource Allocation Scenarios

Let us consider Scenarios 1-2 (visualised in Figure 6.1) to see how requirement $\mathsf{LIV}_1$ could not be guaranteed (while ensuring $\mathsf{SAF}_2$) without an adequate distributed resource allocation protocol.

**Scenario 1** In this scenario, agents $\mathsf{a}_0$ and $\mathsf{a}_1$ 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 ($\mathsf{ppt}(\mathsf{r}_k)$) and then increments the $\mathsf{ppt}(\mathsf{r}_k)$. For instance, in this scenario, resource $\mathsf{r}_0$ firstly received a request message from agent $\mathsf{a}_0$ and thus replied with the value $\mathsf{ppt}(\mathsf{r}_0)$ = 0, which was then followed by a message to $\mathsf{a}_1$ with an incremented $\mathsf{ppt}(\mathsf{r}_0)$ value of 1. In Figure, we denote $\mathsf{a}_n^*$ as the $\mathsf{ppt}(\mathsf{r}_k)$ value sent to $\mathsf{a}_n$. Request messages at resource $\mathsf{r}_1$ 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 $\mathsf{a}_0$ was promised queue-slots $\{(\mathsf{r}_0, 0), (\mathsf{r}_1, 1)\}$ while $\mathsf{a}_1$ queue-slots $\{(\mathsf{r}_0, 1), (\mathsf{r}_1, 0)\}$. If agents would lock these exact queue-slots, resource $\mathsf{r}_0$ would allow agent $\mathsf{a}_0$ to *use* it first, while resource $\mathsf{r}_1$ would concurrently allow agent $\mathsf{a}_1$. The distributed system would deadlock and fail to satisfy $\mathsf{LIV}_2$ requirement as both agents would wait for the second *use* message to ensure $\mathsf{SAF}_2$.



| | slot | $\mathsf{r}_0$ |
|---|---|---|
| | 0 | $\mathsf{a}_0^*$ |
| | 1 | $\mathsf{a}_1^*$ |
| $\mathsf{dl}_0$ | 2 | $\mathsf{a}_0$ |
| $\mathsf{dl}_1$ | 3 | $\mathsf{a}_1$ |
| | . | |
| | n | |

| slot | $\mathsf{r}_1$ |
|---|---|
| 0 | $\mathsf{a}_1^*$ |
| 1 | $\mathsf{a}_0^*$ |
| 2 | $\mathsf{a}_0$ |
| 3 | $\mathsf{a}_1$ |
| . | |
| n | |

| slot | $\mathsf{r}_0$ |
|---|---|
| 0 | $\mathsf{a}_0^*$ |
| 1 | $\mathsf{a}_0^*$ |
| 2 | |
| 3 | |
| . | |
| n | |

| | slot | $\mathsf{r}_1$ |
|---|---|---|
| $\mathsf{dl}_0$ | 0 | $\mathsf{a}_1$ |
| | 1 | $\mathsf{a}_0^*$ |
| | 2 | $\mathsf{a}_0^*$ |
| | 3 | |
| | . | |
| | n | |

Figure 6.1: 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 an agent $a_n$ is $dl(a_n) = \{(r_k, s), (r_{k+1}, s), \dots, (r_{k+m}, s)\}$, where $r_k$ is a resource requested by agent $a_n$ 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 $\{a_0, a_1\}$ initially receiving $\{(r_0, 0), (r_1, 1)\}$ and $\{(r_0, 1), (r_1, 0)\}$ slots, mutually re-request resources again. This time they receive $\{(r_0, 2), (r_1, 2)\}$ and $\{(r_0, 3), (r_1, 3)\}$ slots, and are able to form distributed lanes $dl_0(a_0)$ and $dl_1(a_1)$.

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

### 6.1.3 Semi-Formal Protocol Description

In order to address the issues described in the previous section, we developed a two-stage protocol, where the $stage_1$ of the distributed protocol specifies how an agent forms a distributed lane and $Stage_2$ of the protocol addresses deadlock scenarios, which can occur after agents form distributed lanes. In the following paragraphs we semi-formally describe both stages of the protocol, which then will be formally modelled in Event-B specification language .

$Stage_1$ An agent, which intends to reserve a set of resources starts by sending request messages to resources. The messages are sent to those resources which are part of agent's current objective. In the provided pseudocode excerpt, we first denote relations $sent\_requests$ and $objective$ which are mappings from agents to resources (ln. 1 Algorithm 1). The messages $request$ are sent by an agent $a_n$ to a resource $r_k$ ($r_k \in objective[a_n]$) until $sent\_requests[a_n] = objective[a_n]$ (ln. 4-7 in Algorithm 1). When a resource $r_k$ receives a request message from an agent $a_n$ it responds with a reply message which contains the current promised pointer value $\mathsf{ppt}(\mathsf{r_k})$ and then increments the promised pointer (ln. 6-8 in Algorithm 2). After sending all request messages an agent waits until all reply messages are received from requested resources (ln. 8 in Algorithm 1).

**ALGORITHM 1** Agent $\text{stage}_1$ communication algorithm

```
1: variables sent_requests, received_replies, sent_srequests, sent_write typeof $AGT \leftrightarrow RES$ init $\varnothing$
2: variables objective typeof $AGT \leftrightarrow REL$ init objective $:\in AGT \twoheadleftrightarrow RES$
3: variables replies typeof $AGT \leftrightarrow \mathbb{N}$ init $\varnothing$
4: while sent_requests[$a_n$] $\neq$ objective[$a_n$] do        ▷ requesting resources which belong to the objective
5:     request($a_n$) $\rightarrow r_k$
6:     sent_requests := sent_requests $\cup \{(a_n, r_k)\}$
7: end
8: wait until received_replies[$a_n$] $=$ objective[$a_n$]
9: while |replies[$a_n$]| $\neq 1$ do        ▷ enter while loop if all received indices are not the same
10:        sent_srequests[$a_n$] := $\varnothing$
11:        received_replies[$a_n$] := $\varnothing$
12:        replies[$a_n$] := $\varnothing$
13:        while sent_srequests[$a_n$] $\neq$ objective[$a_n$] do
14:            m := max(replies[$a_n$]) $+ 1$
15:            srequest($a_n$, m) $\rightarrow r_k$        ▷ send a special request message with a desired slot index m
16:            sent_srequests := sent_srequests $\cup \{(a_n, r_k)\}$
17:        end
18:        wait until received_replies[$a_n$] $=$ objective[$a_n$]
19: end
20: while sent_write[$a_n$] $\neq$ objective[$a_n$] do
21:     m := max(replies[$a_n$])
22:     write($a_n$, m) $\rightarrow r_k$
23:     sent_write := sent_write $\cup \{(a_n, r_k)\}$
24: end
```

If all received promised pointer values are the same (a distributed lane can be formed) an agent will complete $\text{stage}_1$ by sending write messages which contain the negotiated index to all requested resources (ln. 20-24 Algorithm 1). But if one of the received promised pointer values is different an agent will start a renegotiation cycle (ln 9-19 Algorithm 1). An agent will now send 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. 14 in Algorithm 1. A resource will reply to srequest message with the higher value of the current $\mathsf{ppt}(\mathsf{r_k})$ or received srequest message value and will update the promised pointer (ln. 9-11 in Algorithm 2). After sending all srequest messages, an agent will wait for reply messages and will restart the loop if received slot indices are not the same.

**ALGORITHM 2** Resource communication algorithm

```
1: variables ppt typeof AGT $\rightarrow \mathbb{N}$ init RES $\rightarrow \{0\}$
2: variables rpt typeof AGT $\nrightarrow \mathbb{N}$ init $\varnothing$
3: variables rlock typeof RES $\nrightarrow$ AGT init $\varnothing$
4: variables mem typeof RES $\rightarrow (\mathbb{N} \nrightarrow \mathsf{AGT})$ init RES $\leftrightarrow \{\varnothing\}$
5: switch received_message do
6:     case request($a_n$)    ▷ a resource replies to a request message with a slot index ppt($r_k$)
7:         reply(ppt($r_k$), $r_k$) $\rightarrow a_n$
8:         ppt($r_k$) := ppt($r_k$) + 1
9:     case srequest($a_n$, n) ▷ a resource replies to a special request message with a slot index ppt($r_k$) or n
10:        reply(max(ppt($r_k$), n), $r_k$) $\rightarrow a_n$
11:        ppt($r_k$) := max(ppt($r_k$), n) + 1
12:    case write($a_n$, m)    ▷ a resource replies to a write message with a pready message if
13:        if rlock($r_k$) = $\varnothing \wedge$ m = rpt($r_k$)    ▷ a resource is not locked and read pointer is at slot m
14:            mem($r_k$, m) := $a_n$
15:            pready($r_k$) $\rightarrow a_n$
16:    case lock($a_n$)
17:        if rlock($r_k$) = $\varnothing$    ▷ a resource replies with a ready message if a resource is not locked
18:            rlock($r_k$) := $a_n$    ▷ a resource is locked and ready message is sent
19:            response($r_k$, READY) $\rightarrow a_n$
20:        if rlock($r_k$) $\neq \varnothing$
21:            response($r_k$, DENY) $\rightarrow a_n$    ▷ resource replies with a deny message if a resource is locked
22:    case release($a_n$, m)    ▷ a resource will unlock itself and remove an agent from memory slot m
23:        mem($r_k$, m) := $\varnothing$
24:        rlock($r_k$) := $\varnothing$
25:        if mem($r_k$) $\neq \varnothing$    ▷ if memory is not empty a resource will
26:            rpt($r_k$) := min(dom(mem($r_k$)))    ▷ update a read pointer to the next reserved slot
27:            $a_{next}$ := mem($r_k$)(rpt($r_k$))    ▷ and send pready message to that agent
28:            pready($r_k$) $\rightarrow a_{next}$
29:
```

It is important to note that described deadlock scenarios and the protocol have a stochastic nature and one needs to guarantee that a desirable state is reachable. In Table 6.3 we elicit additional safety and liveness requirements for the $\text{stage}_1$ of the protocol, which will need to be proved in the formal model.

| | |
|---|---|
| $\text{SAF}_3$ | An agent will not send write (form a distributed lane) messages until all received promised pointer values are identical. |
| $\text{SAF}_4$ | Agents with overlapping objectives will negotiate distributed lanes with different indices. |
| $\text{LIV}_2$ | An agent will eventually negotiate a distributed lane. |

Table 6.3: Low-level protocol $\text{stage}_1$ safety and liveness requirements

**ALGORITHM 3** Agent stage$_2$ communication algorithm

```
1: variables received_response, received_pready, sent_lock, sent_release, consumed, released typeof
   AGT ↔ RES init ∅
2: variables response typeof RES ↔ (AGT ↔ {DENY, READY}) init ∅
3: while received_response[a_n] ≠ objective[a_n] do                                        ▷ asda
4:    wait until received_pready[a_n] = objective[a_n]       ▷ wait until all pre-ready messages are received
5:    while sent_lock[a_n] ≠ objective[a_n] do                      ▷ attempt to lock all requested resources
6:       lock(a_n) → r_k
7:       sent_lock := sent_lock ∪ {(a_n, r_k)}
8:    end
9:    wait until received_response[a_n] = objective[a_n]          ▷ wait until all ready messages are received
10:   if DENY ∈ ran(response)[a_n] do
11:      while sent_release[a_n] ≠ response^{-1}[(a_n, DENY)] do    ▷ resources which sent DENY message
12:         release(a_n) → r_k
13:         sent_release := sent_release ∪ {(a_n, r_k)}
14:      end
15:      received_response[a_n] := ∅
16:      received_pready[a_n] := ∅
17:      sent_lock[a_n] := ∅
18:      sent_release[a_n] := ∅
19:   end
20: end
21: while consumed[a_n] ≠ objective[a_n] do        ▷ all locked resources (stage_2 completed) are consumed
22:    consumed := consumed ∪ {(a_n, r_k)}
23: end
24: while released[a_n] ≠ objective[a_n] do                  ▷ all consumed resources are eventually released
25:    release(a_n) → r_k
26:    released := released ∪ {r_k, a_n}
27: end
```

Stage$_2$ Once an agent completes sending all write messages it will wait for all pready messages from resources (ln. 4 in Algorithm 3). A pready message is sent by a resource firstly if it has received a write message and no other agent is using that resource at that moment - resource is not locked (ln. 12 - 15 in Algorithm 2). Secondly, a pready message will only be sent to an agent if a distributed lane is the new minimum. In our protocol resources read pointer always take a new minimum value in the queue, once an agent sends a release message and allocation is removed from the queue.

When an agent receives all pready messages it will send lock messages to requested resources (ln. 3 - 5 Algorithm 3). If a resource is unlocked upon receiving lock message it will reply with ready message and lock itself, meaning, that it will stop sending pready messages (even to agent with smaller distributed lanes) until a release message is received from that agent (ln. 12 - 15

Algorithm 2). However, if, a resource is locked upon receiving a lock message, it replies with deny message (ln. 16 - 21 in Algorithm 2). If for every lock message an agent received a ready message, it can proceed to use resources and eventually release them. If, one of the messages is a deny message an agent will send release messages to resources (ln. 11 - 14 in Algorithm 3) which sent ready messages to unlock them and will wait for pready messages again to repeat the process.

### 6.1.4 Protocol Model Refinement Strategy

The model development approach we utilise is a top-down approach which starts with the abstract model that formally specifies the objective of the protocol. In fact, distributed aspects of the system are ignored at this model level and the abstract model considers a centralised configuration. The abstract resource allocation protocol model was captured by two machines ($\mathsf{m}_0$ and $\mathsf{m}_1$). The former model essentially summarises the high-level objective of the protocol which is agents safely capturing and releasing collection of resources (objectives). This abstract model contains individual events for capturing and releasing objectives. The next refinement step introduces resources into the model and decomposes two previously introduced events according to the loop pattern defined in Section 4.1.3.

The following group of refinement steps introduces more details about the model by primarily modelling communication aspects. For protocol modelling, we propose to use the backward unfolding style where the next refinement step introduces the preceding protocol step. The abstract models were firstly refined with $\mathsf{stage}_2$ segment of the protocol. In the refinement, $\mathsf{m}_2$, we introduced lock, response and release messages and associated events into the model. In this step we also demonstrated that the protocol's $\mathsf{stage}_2$ ensures safe distributed resource reservation by proving an invariant. The invariant states that no two agents will both be at the resource consuming stage if both requested intersecting collections of resources. The following refinement, $\mathsf{m}_3$, is the bridge between protocol stages $\mathsf{stage}_1$ and $\mathsf{stage}_2$ and introduces two new messages write and pready into the model. In the final refinement step ($\mathsf{m}_4$) we model $\mathsf{stage}_1$ of the distributed protocol which is responsible for creating distributed lanes. The 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.

**Abstract Model Context** The abstract model context introduces agents, resources and objectives into the model as finite carrier sets. The context also contains the enumerated set for the agent's status.

**Message Context** All messages in the protocol were defined in individual context models according to the communication pattern presented in [126].

**Abstract Model** The initial machine (abstract model) summarises the purpose of the protocol. The high-level objective of the protocol is to facilitate the reservation of collection of resources (objectives) by agents. The abstract model captures an agent reserving and eventually releasing an objective.

**Refinement 1 (Abstract ext.)** In this refinement, we introduce resources into the model and now an agent tries to fulfil the objective by locking individual resources (and releasing). To model the iterative process of locking/releasing resources we use a loop modelling pattern.

**Refinement 2** Due to the backward unfolding style the model is then refined with $\mathsf{stage}_2$ part of the protocol. In the refinement lock, response and release messages are introduced. With this refinement step, we also demonstrate that the protocol's $\mathsf{stage}_2$ ensures safe, distributed resource reservation by proving an invariant. The invariant states that no two agents will both be at the resource consuming stage if both requested intersecting collections of resources.

**Refinement 3** This refinement can be considered as a bridge between the protocol's stages $\mathsf{stage}_1$ and $\mathsf{stage}_2$. Here, two new messages - write and pready - are introduced into the model.

**Refinement 4** The final refinement step captures the $\mathsf{stage}_1$ of the distributed protocol which is responsible for creating distributed lanes. The 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. $\mathsf{SAF}_{3\text{-}4}$).

### 6.1.5 Event-B: Abstract Context

The formal protocol modelling was started by defining static model information such as carrier sets, constants and axioms. First of all we create a context for the abstract model which contains three finite size carrier sets representing agents (AGT), resources (RES) and objectives (OBJ) as shown in Listing 6.1. The latter carrier set is used as an extractor operator for groups of resources in a constant function (objr).

Listing 6.1: A context of the abstract resource allocation model

```
context_abstract
SETS
   AGT, RES, OBJ
CONSTANTS
   objr
AXIOMS
   axm1   finite(AGT)
   axm2   finite(RES)
   axm3   ∅ ⊂ RES
   axm4   objr ∈ OBJ → ℙ1(RES)
   axm5   ∃o · o ∈ dom(objr) ⇒ card(objr(o)) ≥ 1
```

We also introduce an enumerated set (AST) to denote agent status or in other words agents program counter values in a separate context model context_agent_state (Listing 6.2). For the abstract model only (CONSUME) and (RELEASE) elements are needed whereas remaining elements will be introduced in the following subsections.

Listing 6.2: A context model defining program counter values of an agent

```
context_agent_state
SETS
   AST
CONSTANTS
   REQUEST, WRITE, RENEGOTIATE, CONFIRMR, CONSUME, RELEASE, CONFIRMW,
   LOCK, CONFIRMC, UNLOCK, CONFIRMP
AXIOMS
   axm1   partition(AST, {REQUEST}, {CONFIRMW}, {WRITE}, {RENEGOTIATE},
{CONFIRMR}, {CONFIRMP}, {LOCK}, {UNLOCK}, {CONFIRMC}, {CONSUME}, {RELEASE})
```

### 6.1.6 Event-B: Machine $m_0$

In modelling the distributed resource allocation protocol we follow a standard Event-B modelling approach where the abstract model summarises the protocol with a centralised view of the system. As previously discussed, the objective of the distributed protocol is to enable safe resource locking. This can be abstracted as agents consuming and releasing objectives. In the abstract model we want to prevent agents consuming identical objectives.

To begin with, we introduce two variables for storing consumed objectives (cons) and agents status (pct0) as shown in Listing 6.3. The agent consume event updates cons variable with a new pair ($act_1$ in Listing 6.4) if an objective has not been consumed (guard $g_1$) and an agent is not consuming any other objectives (guard $g_2$). In addition the event updates agents program counter - variable which helps to track the steps of the agent and discharge proof obligations (guard $a_2$).

Listing 6.3: The invariants of the abstract distributed resource allocation model

**VARIABLES**

$\mathsf{cons}, \mathsf{pct0}$

**INVARIANTS**

$\mathsf{inv}_1 \quad \mathsf{cons} \in \mathsf{AGT} \pfun \mathsf{OBJ}$

$\mathsf{inv}_2 \quad \mathsf{pct0} \in \mathsf{AGT} \rightarrow \mathsf{AST}$

$\mathsf{inv}_3 \quad \mathsf{cons}^{-1} \in \mathsf{OBJ} \pfun \mathsf{AGT}$

Listing 6.4: The event modelling an agent capturing an objective

$\mathsf{agent\_consume} \;\widehat{=}$

**ANY**

$\mathsf{ag}, \mathsf{ob}$

**WHERE**

$\mathsf{grd}_1 \quad \mathsf{ob} \in \mathsf{OBJ} \setminus \mathsf{ran}(\mathsf{cons})$

$\mathsf{grd}_2 \quad \mathsf{ag} \in \mathsf{AGT} \setminus \mathsf{dom}(\mathsf{cons})$

$\mathsf{grd}_3 \quad \mathsf{pct0}(\mathsf{ag}) = \mathsf{CONSUME}$

**THEN**

$\mathsf{act}_1 \quad \mathsf{cons}(\mathsf{ag}) := \mathsf{ob}$

$\mathsf{act}_2 \quad \mathsf{pct0}(\mathsf{ag}) := \mathsf{RELEASE}$

**END**

The second event (Listing 6.5) models the release of an objective by removing a pair which belonged to the cons variable and updating the program counter. The correctness of the abstract model can be verified by proving invariant ($\mathsf{inv}_3$ in Listing 6.3) which asks to prove that an objective is only consumed by a single agent.

Listing 6.5: The event modelling an agent releasing an objective

$\mathsf{agent\_release} \;\widehat{=}$

**ANY**

$\mathsf{ag}, \mathsf{ob}$

**WHERE**

$\mathsf{grd}_1 \quad \mathsf{ag} \in \mathsf{dom}(\mathsf{cons})$

$\mathsf{grd}_2 \quad \mathsf{ob} \in \mathsf{cons}(\mathsf{ag})$

$\mathsf{grd}_3 \quad \mathsf{pct0}(\mathsf{ag}) = \mathsf{RELEASE}$

**THEN**

$\mathsf{act}_1 \quad \mathsf{cons} := \mathsf{cons} \setminus \{\mathsf{ag} \mapsto \mathsf{ob}\}$

$\mathsf{act}_2 \quad \mathsf{pct0}(\mathsf{ag}) := \mathsf{CONSUME}$

**END**

### 6.1.7 Event-B: Machine $\mathsf{m}_1$

The refinement step $\mathsf{m}_1$ expands on the previous refinement by introducing resources into the model. Instead of simply consuming an objective, an agent captures resources until an objective is fulfilled. Captured resources are stored in newly created variable capt whereas objective an agent will try to complete in the function objt (shown in Listing 6.6).

Listing 6.6: Variables and invariants of the refinement step $\mathsf{m}_1$

```
VARIABLES
    capt, objt, pct1
INVARIANTS
    inv_1    $capt \in AGT \to \mathbb{P}(RES)$
    inv_2    $objt \in AGT \to OBJ$
    inv_saf  $\forall a_1, a_2 \cdot a_1 \in dom(capt) \land a_2 \in dom(capt) \land a1 \neq a2 \implies capt(a1) \cap capt(a2) = \varnothing$
```

In contrary to capturing a single objective, an agent might need to consume multiple resources in order to fulfill its objective. For the iterative process, we previously introduced a two event pattern which we instantiate in this refinement step for capturing and releasing events. The loop body event agent_consume_b (Listing 6.7) takes any agent with previously initialised objective and assign a new resource rs to the variable capt (action $\mathsf{act}_1$ in Listing 6.7). The agent must be at CONSUME state (guard $\mathsf{grd}_4$) and the resources must be within agent's objective (guard $\mathsf{grd}_2$) and not yet be captured by any agent (guard $\mathsf{grd}_3$).

Listing 6.7: The event modelling an agent consuming a free resource: loop body event

```
agent_consume_b  $\widehat{=}$
ANY
    ag, rs
WHERE
    grd_1   $ag \in dom(capt)$
    grd_2   $rs \in objr(objt(ag))$
    grd_3   $rs \notin union(ran(capt))$
    grd_4   $pct1(ag) = CONSUME$
THEN
    act_1   $capt(ag) := capt(ag) \cup \{rs\}$
END
```

The loop completion event agent_consume_c (Listing 6.8) would be triggered as soon as the objective has been fulfilled (guard $\mathsf{grd}_1$ in Listing 6.8) and program counter would be updated to new state - RELEASE (action $\mathsf{act}_1$ in Listing 6.8). Similarly, in this refinement we transform agent_release event according based on the event pattern presented. To show correctness of the extended model, we prove an invariant (inv_saf in Listing 6.6) which states that no two agents can have the same resource captured (still this system model is not deadlock free).

Listing 6.8: The event modelling an agent consuming a free resource: loop completion event

```
agent_consume_c ≙
ANY
   ag
WHERE
   grd1   capt(ag) = objr(objt(ag))
   grd2   pct1(ag) = CONSUME
THEN
   act2   pct0(ab) := RELEASE
END
```

### 6.1.8 Event-B: Machine $\mathsf{m}_2$

In this refinement step, we start considering communication between agents and resources. Because of backward unfolding style we introduce $\mathsf{stage}_2$ part of the protocol first. At this stage of the protocol three types of messages can be sent: lock and release by an agent and response message by a resource. In this refinement, we apply previously introduced communication modelling patterns.

Listing 6.9: Variables and invariants of the refinement step $\mathsf{m}_2$

```
VARIABLES
   rdpt, lck, lcke, rsp, rel, pct2
INVARIANTS
   inv1   rdpt ∈ RES ⇸ AGT
   inv2   lck ⊆ LCK
   inv3   lcke ∈ AGT → ℙ(RES)
   inv4   rsp ⊆ RSP
   inv5   rel ⊆ REL
   inv6   pct2 ∈ AGT → AST
```

First of all, machine $\mathsf{m_2}$ is extended with three context models - separately defining each message type according to the generic message context pattern (response message context model shown in Listing 6.10). Then, by following machine communication modelling pattern we create variables $\mathsf{lck}$, $\mathsf{rsp}$ and $\mathsf{rel}$ which represent communication channels of each message. In addition an agent's variable $\mathsf{lcke}$ is created to store already sent $\mathsf{lck}$ messages (shown in Listing 6.9).

Listing 6.10: The context model of the resource response type message

```
resource_response_message
SETS
    RSP, CNF
CONSTANTS
    rsps, rspd, rspv, READY, DENY
AXIOMS
    axm1   rsps ∈ RSP ↠ RES
    axm2   rspd ∈ RSP ↠ AGT
    axm3   rspv ∈ RSP ↠ CNF
    axm4   partition(CNF, {READY}, {DENY})
    axm5   finite(RSP)
    axm6   ∀s, d, v · s ∈ RES ∧ d ∈ AGT ∧ v ∈ CNF ⇒ ∃m · rsps(m) = s ∧ rspd(m) = d ∧ rspv(m) = v
```

In the $\mathsf{stage_2}$ an agent tries to lock resources associated with negotiated distributed lane by sending lock messages. To model that, we apply loop modelling patterns and create two new events: $\mathsf{agent\_lock\_b}$ and $\mathsf{agent\_lock\_c}$ (Listings 6.11 and 6.12). Since, preceding protocol messages are modelled in the next refinements, we model lock message as a initiating message at this stage but later convert it to a reply event type. The first event is the body of a message sending loop which sends a new lock message (action $\mathsf{act_1}$ in Listing 6.11) if a message $\mathsf{lc}$ has not been sent before (guard $\mathsf{grd_1}$) and destination (resource) is within agent's objective (guard $\mathsf{grd_3}$). The program counter of the agent must be in $\mathsf{LOCK}$ protocol phase (guard $\mathsf{grd_4}$). An agent also saves a local copy of the sent message with the event action $\mathsf{act_2}$.

The second event in the loop pattern is a loop completion event $\mathsf{agent\_lock\_c}$ which detects the end of the loop and updates the program counter (Listings 6.12). For the $\mathsf{lock}$ message sending event - an agent must detect when all messages have been sent or in other words the objective has been fulfilled (guard $\mathsf{grd_1}$). This event simply updates the program counter to $\mathsf{CONFIRMC}$ state with the action (action $\mathsf{act_1}$). According to the protocol, after sending all $\mathsf{lock}$ messages an agent will wait for replies before it proceeds to consuming resources.

Listing 6.11: The event modelling an agent locking resources: loop body event

```
agent_lock_b  ≙
ANY
    lc
WHERE
    grd1   $lc \in LCK \setminus lck$
    grd2   $lckd(lc) \notin lcke(lcks(lc))$
    grd3   $lckd(lc) \in objr(objt(lcks(lc)))$
    grd4   $pct2(lcks(lc)) = LOCK$
THEN
    act1   $lck := lck \cup \{lc\}$
    act2   $lcke(lcks(lc)) := lcke(lcks(lc)) \cup lckd(lc)$
END
```

Listing 6.12: The event modelling an agent locking resources: loop completion event

```
agent_lock_c  ≙
ANY
    ag
WHERE
    grd1   $ag \in dom(lcke)$
    grd2   $lcke(ag) = objr(objt(ag))$
    grd3   $pct2(ag) = LOCK$
THEN
    act1   $pct2(ab) := CONFIRMC$
END
```

In order to model a resource's response to the lock message first we created a read pointer variable rdpt to work as a resource lock for an agent. The resource lock is released once a resource receives a release message from an agent who locked the resource (see Listing 6.9). A resource sending a response message is a reply type message therefore we used a reply event modelling pattern (Listings 6.13 and 6.14). A resource sends response message when lock message has been received - guard $grd_1$ in both events. The following two guards ($grd_4$ and $grd_5$) define the source and destination of the new message which are respectively destination and source of received lock message.

Lastly, the response message also carries a value and guards $grd_6$ define the value in both events. The message carries READY value if the resource is not locked by other agent ($grd_3$ in Listing 6.13) otherwise a DENY message is sent ($grd_3$ in Listing 6.14). In addition to sending a message, the resource_response_ready event also removes answered message and if READY message was sent resource locks itself for that agent with action $act_3$.

Once an agent sends all lock messages it must receive all response messages before it can progress. In the model, we created a new event agent_decide (Listing 6.15) which checks the values of received response messages and selects the new program counter value. In principle this event is a loop completion event but in the model we decided to model *decision* events separately to reduce events complexity. The main difference from the basic loop completion event is that depending on conditions different program counter values can selected. In agent_decide event if all messages contained READY value (guard $grd_2$) then the event will update program counter to CONSUME. But if a DENY message (guard $grd_3$) was received program counter is changed to UNLOCK state. In the latter scenario an agent must then send release messages to resources, which sent READY messages and locked their resources, so protocol can progress

Listing 6.13: The event modelling a resource sending a response message: READY type

```
resource_response_ready $\widehat{=}$
ANY
    lc, rs, rp
WHERE
    $grd_1$   $lc \in lck$
    $grd_2$   $rs \in RSP \setminus rsp$
    $grd_3$   $rsps(rs) \notin dom(rdpt)$
    $grd_4$   $rsps(rs) = lckd(lc)$
    $grd_5$   $rspd(rs) = lcks(lc)$
    $grd_6$   $rspn(rs) = READY$
THEN
    $act_1$   $rps := rsp \cup \{rs\}$
    $act_2$   $lck := lck \setminus \{lc\}$
    $act_3$   $rdpt := rdpt \lhd\!\!\!- \{rsps(re) \mapsto rspd(re)\}$
END
```

For selecting correct message from the channel we use messages constant functions and a channel variable - $[rsp \cap rspd^{-1}[\{ag\}]$ selects all rsp messages which were sent to agent ag. Inserting the result to the rsps constant function we get sources (resources) of these messages. This message extraction guard pattern is used widely when relevant messages need to be selected. Events of the unlocking phase are modelled using reply communication pattern and hence not covered in this subsection. Once relevant resources are unlocked agent tries to lock them again in this refinement step.

Listing 6.14: The event modelling a resource sending a response message: DENY type

```
resource_response_deny ≙
ANY
    lc, rs, rp
WHERE
    grd1   lc ∈ lck
    grd2   rs ∈ RSP \ rsp
    grd3   rsps(rs) ∈ dom(rdpt)
    grd4   rsps(rs) = lckd(lc)
    grd5   rspd(rs) = lcks(lc)
    grd6   rspn(rs) = DENY
THEN
    act1   rps := rsp ∪ {rs}
    act2   lck := lck \ {lc}
END
```

Listing 6.15: The event modelling agents decision between restarting $\text{stage}_2$ or consuming resources

```
agent_decide ≙
ANY
    ag, pc
WHERE
    grd1   $rsps[rsp \cap rspd^{-1}[\{ag\}]] = lcke(ag)$
    grd2   $rspn[rsp \cap rspd^{-1}[\{ag\}]] = \{READY\} \Rightarrow pc = CONSUME$
    grd3   $DENY \in rspn[rsp \cap rspd^{-1}[\{ag\}]] \Rightarrow pc = UNLOCK$
    grd4   pct2(ag) = CONFIRMC)
THEN
    act1   pct2(ag) := pc
END
```

Listing 6.16: The safety invariant for prohibiting mutual resource locking by different agents

```
INVARIANTS
    SAF1   $\forall a1, a2 \cdot pct2(a1) = CONSUME \wedge pct2(a2) = CONSUME \wedge a1 \neq a2$
               $\Rightarrow objr(objt(a1)) \cap objr(objt(a2)) = \varnothing$
```

In this refinement step, which specifies $\text{stage}_2$ of the distributed protocol, we were required to prove the invariant expressed in Listing 6.16. The safety invariant relates to the mutual exclusion safety protocol property $\text{SAF}_1$ specified in Figure 6.2. The safety invariant was proved interactively.

### 6.1.9 Event-B: Machine $\mathsf{m}_3$

In this refinement step we continue to unfold protocol backwards and introduce two new messages write and pready. In fact this refinement step can be thought as an intermediate step gluing protocol stages $\mathsf{stage}_1$ and $\mathsf{stage}_2$. Events for capturing write message sending were developed using the initiating message pattern. Since they were structurally identical to the previous refinement events agent_lock_b and agent_lock_c events we do not discuss them here. The more interesting challenge of this refinement step was correctly capturing the pready message. A variable of type $\mathsf{mem} \in \mathsf{RES} \rightarrow \mathbb{P}(\mathsf{AGT})$ was created to abstract queue like resource memories (see Listing 6.17) which are introduced in the next refinement step.

Listing 6.17: The event modelling an agent capturing an objective

**VARIABLES**

wrt, prd, mem, wrte, pct3

**INVARIANTS**

| | |
|---|---|
| inv_1 | $\mathsf{wrt} \subseteq \mathsf{WRT}$ |
| inv_2 | $\mathsf{prd} \subseteq \mathsf{PRD}$ |
| inv_3 | $\mathsf{mem} \in \mathsf{RES} \rightarrow \mathbb{P}(\mathsf{AGT})$ |

A resource sends pready message to inform an agent of its availability for consumption and that it can be locked now. There are two different cases when this message should be sent and instead of constructing a single event to cover all scenarios it was decided to create more trivial events for each of the cases (Listing 6.18 and 6.19). In the model we created a reply type resource_write_preready event which creates a new message and removes the answered message from the wrt channel if a resource is not locked. A pready message will be sent if a write message has been received (guard $\mathsf{grd}_1$ in Listing 6.18) and resource is not locked by an agent ($\mathsf{grd}_3$).

The second event resource_release_pready was necessary to cover a scenario where pready message was sent to an agent in response to its write message but in the end the agent was not able to lock all resources. This would mean that write messages have been removed and resource_write_preready guards would never be satisfied for that agent. Yet an agent would be still waiting to receive pready messages eventually. Therefore we introduced initiating message resource_release_pready event which sends a new pready message to any agent which is interested in that resource if a resource is not locked.

Listing 6.18: The event modelling a resource replying to write message with a pready message

```
resource_pready_write ≙
ANY
    wr
    pr
WHERE
    grd1   $wr \in wrt$
    grd2   $pr \in PRD\ prd$
    grd3   $wrtd(wr) \notin dom(rdpt)$
    grd4   $wrts(wr) \in mem(wrtd(wr))$
    grd5   $prdd(pr) = wrts(wr)$
    grd5   $prds(pr) = wrtd(wr)$
THEN
    act1   $prd := prd \cup \{pr\}$
    act2   $wrt := wrt \setminus \{wr\}$
END
```

Listing 6.19: The event modelling a resource sending a pready message to an agent in a memory

```
resource_pready_release ≙
ANY
    wr
    pr
WHERE
    grd1   $pr \in PRD \setminus prd$
    grd2   $prds(pr) \notin dom(rdpt)$
    grd3   $prdd(pr) \in mem(prds(pr))$
THEN
    act1 $prd := prd \cup \{pr\}$
END
```

### 6.1.10 Event-B: Machine $m_4$

The final refinement step $m_4$ introduces the remaining part of the protocol - $stage_1$. This stage starts with an agent requesting a set of resources and finishes with agent forming a distributed lane by sending a write messages to the same resources. In between, an agent might need to send a special request message srequest if a distributed lane was not negotiated in the first attempt. To allow distributed lane forming - a resource memory introduced in the previous refinement now must be refined to queue-like data structure with corresponding variables (e.g. read pointer, promised pointer).

Listing 6.20: Variables and invariants of the machine model $m_4$

```
VARIABLES
    req, rqst, rqs, rqts, rep...
INVARIANTS
    inv_1   $req \subseteq REQ$
    inv_2   $rqst \in AGT \rightarrow \mathbb{P}(RES)$
    inv_3   $rqs \subseteq REQ$
    inv_4   $rqts \in AGT \rightarrow \mathbb{P}(RES)$
    inv_5   $rep \subseteq REP$
    inv_6   $ppt \in RES \rightarrow \mathbb{N}$
    inv_7   $rpt \in RES \nrightarrow \mathbb{N}$
    inv_8   $lan \in RES \rightarrow (\mathbb{N} \nrightarrow AGT)$
    inv_9   $pct4 \in AGT \rightarrow AST$
```

To begin with we start with introducing communication channel variables for request, reply and srequest messages. In addition two local variables rqst and rqts were created to store messages on the agent side (shown in Listing 6.20). Following the loop and initiating message modelling patterns we introduced two events agent_request_b and agent_request_c for requesting resources (Listings 6.21 and 6.22).

Listing 6.21: The event modelling an agent requesting a resource: loop body event

```
agent_request_b $\widehat{=}$
ANY
    rq
WHERE
    grd_1   $rq \in REQ \setminus req$
    grd_2   $reqd(rq) \notin rqst(reqs(rq))$
    grd_3   $reqd(rq) \in objr(objt(reqs(rq)))$
    grd_4   $pct4(reqs(rq)) = REQUEST$
THEN
    act_1   $wrt := wrt \cup \{wr\}$
    act_2   $wrte(wrts(wr)) := wrte(wrts(wr)) \cup \{wrtd(wr)\}$
    act_3   $mem(wrtd(wr)) := mem(wrtd(wr)) \cup wrts(wr)$
END
```

After all request messages have been sent event agent_request_c changes agents program counter to CONFIRMW state ($act_1$ in Listing 6.22) which informs an agent to wait until all reply messages have been received.

Listing 6.22: The event modelling an agent requesting a resource: loop completion event

```
agent_request_c $\widehat{=}$
ANY
    ag
WHERE
    grd1   $ag \in dom(rqst)$
    grd2   $rqst(ag) = objr(objt(ag))$
    grd3   $pct4(ag) = REQUEST$
THEN
    act1   $pct4(ag) := CONFIRMW$
END
```

On the resource side, a reply type event resource_reply_general (Listing 6.23) was created which simply sends a new reply message (action $act_1$) containing the current promised pointer value (guard $grd_5$), removes request message ($act_2$) and increments the promised pointer ($act_3$). Once an agent has sent all request and respectively received all reply messages a decision must be made whether to form a distributed lane or renegotiate new indexes.

Listing 6.23: The event modelling a resource replying to a request message

```
resource_reply_general $\widehat{=}$
ANY
    rp
    rq
WHERE
    grd1   $rq \in req$
    grd2   $rp \in REP \setminus rep$
    grd3   $repd(rp) = reqs(rq)$
    grd4   $reps(rp) = reqd(rq)$
    grd5   $repn(rp) = ppt(reqs(rq))$
THEN
    act1   $rep := rep \cup \{rp\}$
    act2   $req := req \setminus \{rq\}$
    act3   $ppt(reps(rp)) := ppt(reps(rp)) + 1$
END
```

The distributed lane will be formed if all reply messages contained the same promised pointer value otherwise an agent must renegotiate a distributed lane. In the model, we created a new *decision* agent_decide_write_renegotiate event which changes the program counter according to the named conditions (Listing 6.24). To check whether all reply messages contained the same index it is sufficient to check the set size (cardinality) of the $repn[rep \cap repd^{-1}[ag]]$ values set. The guard $grd_1$ must have both local variables because this event can be enabled after general request or srequest message sending.

Listing 6.24: The event modelling agent's decision on whether to restart $\text{stage}_1$ or start $\text{stage}_2$

```
agent_confirm_write_renegotiate  ≙
ANY
    ag
    pc
WHERE
    grd1   $ag \in repd[rep]$
    grd2   $card(rqst(ag) \cup rqts(ag)) = card(rep \cap repd^{-1}[\{ag\}])$
    grd3   $reps[rep \cap repd^{-1}[\{ag\}]] = objr(objt(ag))$
    grd4   $pct4(ag) = CONFIRMW$
    grd5   $card(repn[rep \cap repd^{-1}[\{ag\}]]) > 1 \Rightarrow pc = RENEGOTIATE$
    grd6   $card(repn[rep \cap repd^{-1}[ag]]) = 1 \Rightarrow pc = WRITE$
THEN
    act1   $pct4(ag) := pc$
END
```

If an agent was not able to negotiate a distributed lane it must send a new, special message - srequest. This time instead of requesting an arbitrary index a new srequest message contains a desired index which is computed by adding a non-zero constant to the highest reply index. To send srequest messages we created two reply type events: agent_renegotiate_b and agent_renegotiate_c (Listings 6.25).

Listing 6.25: The event modelling an agent renegotiating a resource: loop body event

```
agent_renegotiate_b  ≙
ANY
    ag, rp, rq
WHERE
    grd1   $ag \in repd[rep]$
    grd2   $pct4(ag) = RENEGOTIATE$
    grd3   $rp \in rep \cap repd^{-1}[\{ag\}]$
    grd4   $rq \in RQS \setminus rqs$
    grd5   $rqss(rq) = ag$
    grd6   $rqsd(rq) = reps(rp)$
    grd7   $rqsn(rq) = max(repn[rep \cap repd^{-1}[\{ag\}]]) + 1$
    grd8   $rqsd(rq) \notin rqts(rqss(rq))$
    grd9   $rqsd(rq) \notin rqst(rqss(rq))$
THEN
    act1   $rqs := rqs \cup \{rq\}$
    act2   $rep := rep \setminus \{rp\}$
    act3   $rqts(rqss(rq)) := rqts(rqss(rq)) \cup \{rqsd(rq)\}$
    act4   $rqst(rqss(rq)) := rqst(rqss(rq)) \setminus \{rqsd(rq)\}$
END
```

Listing 6.26: The event modelling an agent renegotiating a resource: loop completion event

```
agent_renegotiate_c ≙
ANY
    ag
WHERE
    grd2   rqts(ag) = objr(objt(ag))
    grd3   pct4(ag) = RENEGOTIATE
THEN
    act1   pct4(ag) := CONFIRMW
END
```

As a standard reply type event agent_renegotiate_b creates a new message and removes the answered reply message from the channel. Whereas agent_renegotiate_c event updates the program counter again to CONFIRMW state - this cycle repeats until a distributed lane is negotiated.

Lastly in this refinement we introduce the distributed lane data structure and accordingly update relevant events. Two variables ppt and rpt are created to respectively represent promise pointer and read pointer of a resource. For each resource the promised pointer is initialised to zero and only two reply events resource_reply_general and resource_reply_special (Listing 6.27) modify a promised pointer variable. The first event simply increments the current $\mathsf{ppt}(\mathsf{r})$ value after reply message has been sent whereas the latter event updates the $\mathsf{ppt}(\mathsf{r})$ by computing action $\mathsf{act}_3$, where maximum function parameters are the current $\mathsf{ppt}(\mathsf{r})$ value and received srequest message value.

Listing 6.27: The event modelling a resource replying to a srequest message

```
resource_reply_special ≙
ANY
    rp
    rq
WHERE
    grd1   rq ∈ rqs
    grd2   rp ∈ REP \ rep
    grd3   repd(rp) = rqss(rq)
    grd4   reps(rp) = rqsd(rq)
    grd5   repn(rp) = max({ppt(reps(rp)), rqsn(rq)})
THEN
    act1   rep := rep ∪ {rp}
    act2   rqs := rqs \ {rq}
    act3   ppt(reps(rp)) := max({ppt(reps(rp)), rqsn(rq)}) + 1
END
```

Read pointers are updated by two events which send pready messages. In constrast to the previous variable the read pointer $\mathsf{rpt}(\mathsf{r})$ is always set to the minimum value of the request pool. This is necessary as some agent might negotiate a distributed lane with lower index than others but its write messages are delayed (or even lost) so the protocol would halt. Allowing agents with higher distributed lane indexes but sooner write message arriving to consume resources introduces fault-tolerance into the protocol. Guards of these two events were also strengthened to only send pready messages to newly incoming agents if their index is the lowest in the request pool.

### 6.1.11 Proving Functional Protocol Correctness

As shown in Section 6.1.3 (Scenarios 1 - 2) high-level system requirements can only be met if an agent invariably and correctly forms a distributed lane. The probabilistic lane forming eventuality ($\mathsf{LIV}_2$) is discussed separately, while in the following paragraphs, we focus on the proof regarding requirements $\mathsf{SAF}_{3\text{-}4}$.

$\mathsf{SAF}_3$ is required to ensure that agent's resource objectives are not satisfied or satisfied in 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 $\mathsf{SAF}_3$ is maintained in the model. For illustrative purposes we focus on details of verifying a slightly more interesting case of $\mathsf{SAF}_4$ which assumes that $\mathsf{SAF}_3$ property is proven.

Listing 6.28: The event modelling a resource replying to a request message with history variable extension

```
resource_reply_general  $\widehat{=}$
ANY
    rq, rp
WHERE
    grd$_1$   $\mathsf{rq} \in \mathsf{req}$
    grd$_2$   $\mathsf{rp} \in \mathsf{REQ} \setminus \mathsf{rep}$
    grd$_3$   $\mathsf{repd}(\mathsf{rp}) = \mathsf{reqs}(\mathsf{rq})$
    grd$_4$   $\mathsf{reps}(\mathsf{rp}) = \mathsf{reqd}(\mathsf{rq})$
    grd$_5$   $\mathsf{repn}(\mathsf{rp}) = \mathsf{ppt}(\mathsf{reps}(\mathsf{rp}))$
THEN
    act$_1$   $\mathsf{rep} := \mathsf{rep} \cup \{\mathsf{rp}\}$
    act$_1$   $\mathsf{req} := \mathsf{req} \setminus \{\mathsf{rq}\}$
    act$_3$   $\mathsf{ppt}(\mathsf{res}) := \mathsf{ppt}(\mathsf{res}) + 1$
    act$_4$   $\mathbf{his_{ppt}(res) := his_{ppt}(res)}$
    act$_5$   $\mathbf{his_{wr}(res) := his_{wr}(res) + 1}$
END
```

Requirement $\mathsf{SAF_4}$ addresses potential cross-blocking deadlocks or resource double locking due to distributed lane overriding. The strategy is to prove the requirement by showing 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 $\mathsf{SAF_3}$). 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 $\mathsf{SAF_4}$ it is enough to show that each resource replies to a request or special request message with a unique promised pointer value.

Listing 6.29: Additional variables and invariants added for providing safety property $\mathsf{SAF_4}$

**VARIABLES**

$\mathsf{his_{ppt}}, \mathsf{his_{wr}}$

**INVARIANTS**

$\mathsf{his_{ppt}} \in \mathsf{RES} \to (\mathbb{N} \nrightarrow \mathbb{N})$

$\mathsf{his_{wr}} \in \mathsf{RES} \to \mathbb{N}$

$\mathsf{inv_{saf4}}$ $\quad \forall \mathsf{r}, \mathsf{n_1}, \mathsf{n_2} \cdot \mathsf{r} \in \mathsf{RES} \land \mathsf{n_1}, \mathsf{n_2} \in \mathsf{dom}(\mathsf{his_{ppt}}(\mathsf{r})) \land \mathsf{n_1} < \mathsf{n_2} \Rightarrow \mathsf{his_{ppt}}(\mathsf{r})(\mathsf{n1}) < \mathsf{his_{ppt}}(\mathsf{r})(\mathsf{n2})$

$\mathsf{inv\_his_{ppt}}$ $\quad \forall \mathsf{res} \cdot (\mathsf{his_{wr}}(\mathsf{res}) = 0 \land \mathsf{his_{ppt}}(\mathsf{res}) = \varnothing) \lor$
$\quad (\mathsf{dom}(\mathsf{his_{ppt}}(\mathsf{res})) = 0 \,..\, \mathsf{his_{wr}}(\mathsf{res}) - 1 \land \mathsf{his_{ppt}}(\mathsf{res})(\mathsf{his_{wr}}(\mathsf{res}) - 1) = \mathsf{ppt}(\mathsf{res}) - 1)$

To prove this property we first extend our model with a variable $\mathsf{his_{ppt}}$ and an entry variable $\mathsf{his_{wr}}$ or *time-stamp* as shown in Listing 6.29. The former variable stores each resource's chronological promised pointer updates and latter works as a write pointer for history variable. After introducing history variables, we updated two events $\mathsf{resource\_reply\_\{general, special\}}$ which update promised pointer variables (after sending reply messages) by adding two new actions $\mathsf{act_4}$ and $\mathsf{act_5}$ as shown in Listing 6.28 (identical extension for $\mathsf{resource\_reply\_special}$ event).

Action $\mathsf{act_4}$ updates a history variable for a resource $\mathsf{res}$ with the current write stamp and promised pointer ($\mathsf{ppt}(\mathsf{res})$) value sent. The next action $\mathsf{act_5}$ simply updates the resource's write stamp. We can then add the main invariant to prove ($\mathsf{inv_{saf4}}$) 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.

To prove that $\mathsf{resource\_reply\_\{general, special\}}$ preserve $\mathsf{inv_{saf4}}$, 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 ($\mathsf{inv\_his_{ppt}}$) and proved they are preserved by the events which helped to prove $\mathsf{inv_{saf_4}}$.

**Proof statistics.** In Table 5.1 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 [126] use and the SMT-based verification support [110, 111].

| 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 |

Table 6.4: Event-B protocol model proof statistics

### 6.1.12 Proving Protocol Probabilistic Termination 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 and liveness properties are hard to formalise and deductively prove in the Event-B method. Alternatively, a model checking based approach could be used to find if a satisfying state can be reached. However, the ProB model checker is often not able to automatically find a bounded model of a system, when expressions of the Event-B model are too complex for the ProB constraint solver. Furthermore, the ProB model checker is not a probabilistic model checker, so, probabilistic behaviours cannot be captured or reasoned about.

Therefore, it was decided to prove the termination of the protocol outside of the Event-B model by redeveloping and proving part of the protocol ($\mathsf{stage_1}$) in the PRISM probabilistic model checker. The first advantage of using the PRISM model checker is that it allows developers assigning probabilities to events, which can be used to model message delays. The second advantage of PRISM is that it supports quantitative properties. For example, one can express a question in PRISM like ”What is the probability of event X occurring?”. In this work, we used a quantitative reasoning to demonstrate that the probability of an agent negotiating a distributed lane ($\mathsf{LIV_2}$) approaches one as a number of completed renegotiation cycles increases.

However, 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 an abstraction for a protocols $\mathsf{stage_1}$, we created a skeleton model, which then could be instantiated to model specific scenarios of $\mathsf{stage_1}$ with $\mathsf{n}$ agents, $\mathsf{m}$ 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.

To model the distributed lane negotiation protocol we use the Discrete Time Markov Chains framework from the PRISM model checker. The generic model can be partitioned into three main parts: global variable declaration, resource modules and agent modules.

**Global Variables** There are two types of global variables $\mathsf{a_{nm}}$ and $\mathsf{aon_{nm}}$ and both types are associated with agents. The first variable $\mathsf{a_{nm}}$ is used to model $\mathsf{stage_1}$ messages exchanges between an agent $\mathsf{n}$ and resource $\mathsf{m}$. For example, if one wants to model a scenario where an agent $\mathsf{a_0}$ wants to reserve resources $\mathsf{r_0, r_1}$ two global variables $\mathsf{a_{00}}$ and $\mathsf{a_{01}}$ (initially set to 0) are needed. The second variable $\mathsf{aon_{nm}}$ is a status variable and similarly to the previous one, a separate variable is created for every agent-resource relation in the scenario. The status variable can take one of two possible values ([0..1] init 1) and are used to update probabilities as well as control models flow. The following paragraphs explain how these variables are modified.

**Resource Modules** For each resource an individual resource module is created which contain a local promised pointer variables and single command. Two parameters which can be varied for the $\mathsf{ppt_m}$: initial value (or the promised pointer offset) and the upper bound of the promised pointer. The module also only requires a single command which is expressed below. The additional guards are mainly added to protect from variable overflows and terminate model once all distributed lanes are negotiated.

The main guard uses agent status variables $\mathsf{aon_{nm}}$ to activate module only if there is at least one agent which is interested in reserving that resource but has not received a promised pointer value. Then for each interested agent, there is an associated probabilistic state transition. A probability of specific state transition is inversely proportional to the number of remaining agents which have not yet received a promised pointer value from that resource. The specific state transition will update variables $\mathsf{a_{am}}$, $\mathsf{ptt_m}$ and will deactivate status variable $\mathsf{aon_{am}}$ by setting it to zero. All interested agents have structurally identical expression in the command with modified variable identifiers (after +).

**ALGORITHM 4** Resource module skeleton model

```
1: module $\mathsf{r_m}$
2: $\mathsf{ppt_m\ :\ [0\,..\,T]\ init\ T_n;}$
3: $[]\quad \mathsf{additional\_guards}\quad \&\quad (\mathsf{aon_{am}} + \mathsf{aon_{bm}} + \mathsf{aon_{cm}} + \cdots > 0) \rightarrow$
4: $\mathsf{aon_{am}} * (\mathsf{aon_{am}} + \mathsf{aon_{bm}} + \mathsf{aon_{cm}} ...)^{-1} : (\mathsf{a'_{am}} = \max(\mathsf{a_{am}}, \mathsf{ppt_m}))\quad \&$
5: $(\mathsf{ppt'_m} = \max(\mathsf{a_{am}}, \mathsf{ppt_m}) + 1)\quad \&\quad (\mathsf{aon'_{am}} = 0) + \cdots ;$
6: endmodule
```

**Agent Modules** Similarly each agent also has an associated agent module with a local variable $\mathsf{dist_n}$ for storing distributed lane index. The module contains two events for two scenarios: a distributed lane has been negotiated (distributed lane variable is updated) and distributed lane has not been negotiated. The first command is firstly enabled if all agents status variables have been set to zero i.e. all promised pointer values have been received. The second guard ensures that all promised pointer received values are the same and other remaining guards protect from variable value overflow. The action of the first command, will update $\mathsf{dist_n}$ variable and the module is never enabled again.

**ALGORITHM 5** Agent module skeleton model

```
1: module $\mathsf{a_n}$
2: $\mathsf{dist_n\ :\ [\text{-}1\,..\,100]\ init\ \text{-}1;}$
3: $[]\quad (\mathsf{aon_{na}} + \mathsf{aon_{nb}} + \cdots = 0)\quad \&\quad (\mathsf{a_{na}} = \mathsf{a_{nb}} = \cdots)\ \&\ \mathsf{other\_guards} \rightarrow$
4: $\mathsf{dist'_n} = \mathsf{a_{na}};$
5: $[]\quad (\mathsf{aon_{na}} + \mathsf{aon_{nb}} + \cdots = 0)\quad \&\quad (\exists\, \mathsf{x, y} \cdot \mathsf{a_{nx}} \neq \mathsf{a_{ny}})\ \&\ \mathsf{other\_guards} \rightarrow$
6: $\mathsf{a'_{na}} = \max(\mathsf{a_{na}}, \mathsf{a_{nb}}, \cdots) + 1\quad \&\quad \mathsf{a'_{nb}} = \max(\mathsf{a_{na}}, \mathsf{a_{nb}}, \cdots) + 1\quad \&\quad \cdots$
7: $\&\quad \mathsf{aon'_{na}} = 1\quad \&\quad \mathsf{aon'_{nb}} = 1\quad \&\quad \cdots ;$
8: endmodule
```

The second command models the second scenario where distributed lane was not negotiated a special request messages would be sent in the protocol. In this command we replace a guard which now enables command if there is a pair of $\mathsf{a_{nm}}$ variables which are not equal. The actions of the command will update variables $\mathsf{a_{nm}}$ with a value of desired distributed lane as well as status variables $\mathsf{aon_{nm}}$ to allow getting new in resource modules.

### 6.1.13 Proving Protocol Probabilistic Termination with PRISM: Results

In this subsection, we discuss stochastic model checking results which proves that $\mathsf{LIV}_2$ requirement is preserved. In particular, we focus on showing that $\mathsf{LIV}_2$ requirement is ensured in Scenario 2 (Section 2.2).

In order to demonstrate that $\mathsf{LIV}_2$ requirement holds in Scenario 2 (Section 2.2) we used $\text{stage}_1$ 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 the probability of an agent forming a distributed lane while the upper limit of the promised pointer is increased ($\mathsf{n}$ 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.

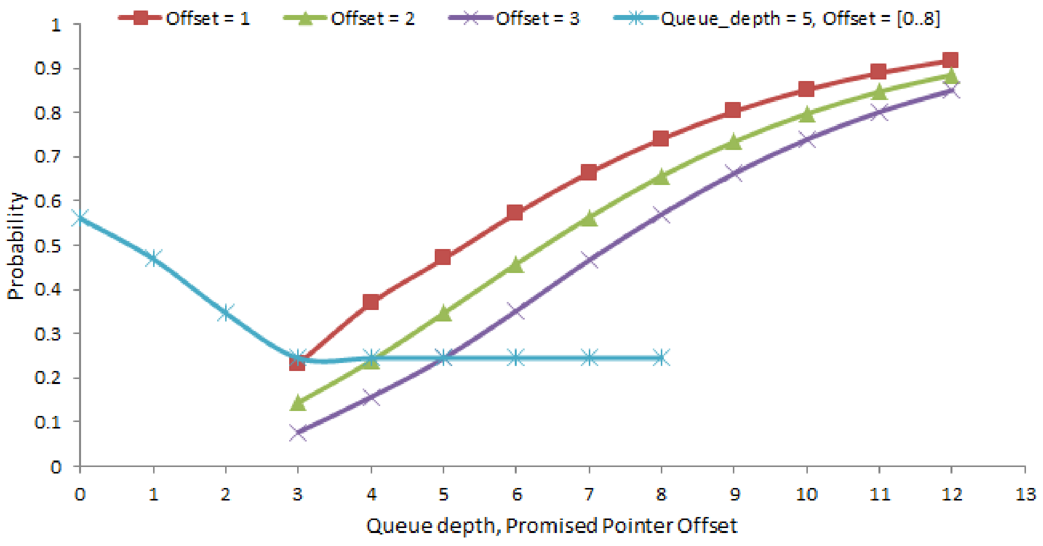


Figure 6.2: Scenario 2 with varied resource promised pointer offset and queue depth.

For each scenario, we would run a quantitative property: $\mathsf{P} = ?\,[\mathsf{F}\;\mathsf{dist}_0 > \text{-}1]$ 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 Figure 3 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.

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). 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 $\mathsf{LIV}_2$ requirement as distributed lanes can be negotiated.

## 6.2 Case Study 2: Heterogeneous Railway Signalling System

Integrating modern railway signalling systems within outdated national railway networks is currently one of the major challenges in the railway domain. A gradual railway network modernisation process means that heterogeneous railway signalling networks will be inevitable due to practical constraints to upgrade the whole network at once.

In some situations, mainline services must be integrated with urban networks which mostly operate with different signalling solutions. For example, Crossrail is a major ongoing railway project where mainline services will be integrated with a high -performance urban railway system. This particular network will operate with three different signalling systems. On the western and eastern branches of the network fixed block signalling systems will operate whereas the central area will be operated with a moving block principle. Novel signalling interfaces will be developed to ensure a smooth and safe rolling stock signalling transition.

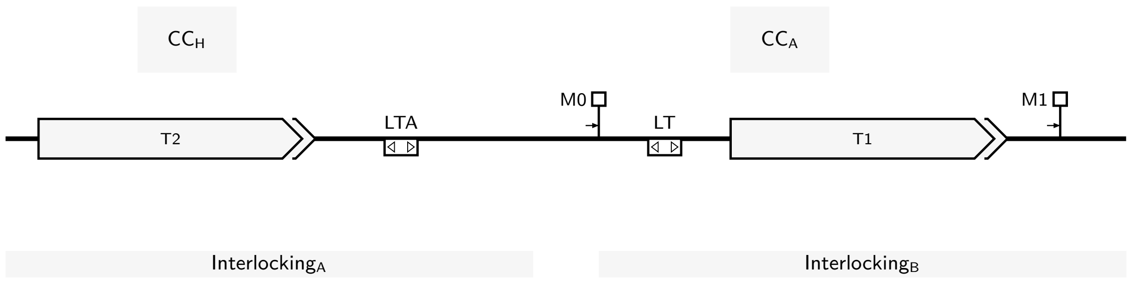


Figure 6.3: An example of the heterogeneous railway signalling model with two transitioning trains

In this case study, we model the signalling transition of a train in a heterogeneous signalling system with interfacing fixed and moving block signalling systems. The model we consider in this case study is based on the simplified Crossrail western network section signalling interface between ETCS Level 2 and CBTC [28] (discussed in Section 2.1.2). However, in this model, we only consider a one-way transition scenario, where all trains in the model are initially in the moving-block signalling system and transition to the fixed-block part of the network. The model is visualised in Figure 6.3 with two trains transitioning from a moving-block to a fixed-block signalling system. In the following sections, we further describe the heterogeneous model, a handover communication protocol and the main safety requirements of the system. Then, we describe the formal Event-B of the system, which refined the generic communication-based hybrid signalling model we introduced in Chapter 5.

### 6.2.1 Heterogeneous Signalling Model Description and Requirements

The heterogeneous signalling model we consider in this case study builds upon the previously described communication-based hybrid signalling model in Section 5.2. This extension will be reflected in a formal Event-B model by refining a generic hybrid signalling model according to the proposed multifaceted formal development methodology. In the following paragraphs, we introduce the key extensions of the generic signalling model, including additional field elements, system-level transition protocol and system requirements.

**System Model** In order to model a heterogeneous signalling system and rolling stock signalling transition, we require extending the generic hybrid signalling model described in Section 5.2. The first modification we make is refining a position reference balises with a type parameter. The two new balise types for a system-level transition protocol are $\mathsf{Level\ Transition\ Announcement\ (LTA)}$ and $\mathsf{Level\ Transition(LT)}$ which once passed over by a train will respectively inform a communication-centre to start a handover procedure and that a train has crossed a signalling boundary. The handover procedure is defined by a system-level transition protocol, which extends the protocol of the generic communication-based signalling system (see Figure 5.4) with functionality to issue trains with a movement authority that crosses into the next signalling area and handover train management to the accepting communication centre. In Table 6.5, we elicit specifications and assumptions of the heterogeneous signalling model we consider in this case study.

**System-Level Transition Protocol** A system-level transition protocol describes message exchanges between adjacent signalling systems and their communicating agents. The objective of the protocol is to safely and efficiently transition rolling stock from one signalling system to another. In communication-based signalling systems, this generally involves transferring management of a transitioning train from handing-over communication centre $\mathsf{CC_H}$ to accepting communication centre $\mathsf{CC_A}$.

$\mathsf{SPE}_1$ | All trains in the system model have identical physical and cyber behaviour (as described in Section 5.2).

$\mathsf{SPE}_2$ | A single train on-board computer can operate it in both signalling areas.

$\mathsf{SPE}_3$ | There exists a communication interface between the adjacent communication centres.

$\mathsf{SPE}_4$ | All trains are initially located and safely separated in the moving-block signalling area.

$\mathsf{SPE}_5$ | The model only allows system-level transition from a moving-block signalling area to a fixed-block.

Table 6.5: Specifications of the heterogeneous signalling model

1. A position report (with LTA) message is sent to the handing-over communication centre once a transitioning train passes over a level-transition announcement (LTA) balise. This message informs a communication centre that a train is approaching the end of the signalling area and for a movement authority extension has to be requested from an adjacent signalling area.

2. A route request message is sent from $\mathsf{CC}_\mathsf{H}$ to $\mathsf{CC}_\mathsf{A}$ requesting extending movement authority into $\mathsf{CC}_\mathsf{A}$ signalling area. This models assumes $\mathsf{CC}_\mathsf{A}$ operates with a fixed-block signalling, and thus, a route (block) is requested.

3. A movement authority message is sent to $\mathsf{CC}_\mathsf{H}$ by $\mathsf{CC}_\mathsf{A}$ with an extended movement authority (if route is available) into an accepting signalling area. If route is not available a message does not contain an extension distance and train must not be allowed to transition. The movement authority is forwarded to the transition train.

4. A position report (with LT) message is sent to $\mathsf{CC}_\mathsf{H}$ once a train passes over a level transition balise indicating that it has transitioned to the adjacent signalling area. The message is then forwarded to $\mathsf{CC}_\mathsf{A}$ indicating that the management of the train can be taken over by $\mathsf{CC}_\mathsf{A}$

5. Once $\mathsf{CC}_\mathsf{A}$ is informed that a train has passed over a level transition balise with message (4) a take over acknowledgement message is sent to $\mathsf{CC}_\mathsf{H}$ to finalise train management transfer.

6. A connect message is sent by $\mathsf{CC}_\mathsf{H}$ to a transitioning train informing that it is now registered with $\mathsf{CC}_\mathsf{A}$.

7. A register message is sent by a train to $\mathsf{CC}_\mathsf{A}$ indicating that it will now act on a movement authority received from $\mathsf{CC}_\mathsf{A}$.

8. A take over completed to $\mathsf{CC_H}$ orders communication centre to remove transitioned train from its map.

The transitioning of the train is completed when an updated movement authority is received from $\mathsf{CC_A}$ indicating to a train that it can disconnect from $\mathsf{CC_H}$ and act on the new movement authority.

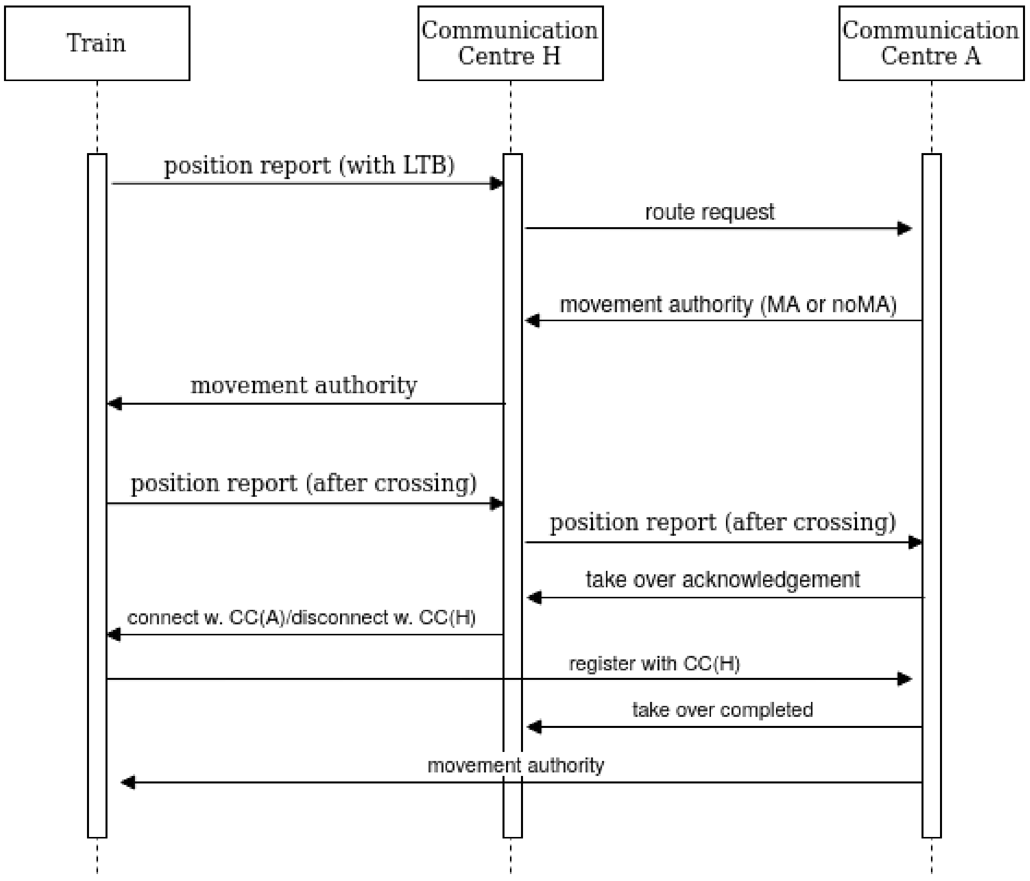

Train
Communication Centre H
Communication Centre A
position report (with LTB)
route request
movement authority (MA or noMA)
movement authority
position report (after crossing)
position report (after crossing)
take over acknowledgement
connect w. CC(A)/disconnect w. CC(H)
register with CC(H)
take over completed
movement authority

Figure 6.4: Sequence diagram of the heterogeneous signalling model

**Safety Requirements** In this case study, we are interested in modelling a heterogeneous signalling system, and more importantly, formally ensuring that the system's safety requirements are preserved by mathematically proving the Event-B model. In safety requirement Table 6.6 we describe safety requirements we intend to prove in the heterogeneous signalling system formal model.

The safety requirements $\mathsf{SAF}_{1,2}$ generally apply to any communication-based signalling system and have been proved in a generic hybrid communication-based signalling model. In this case study, we are focusing on a safe movement authority extension to the adjacent signalling system and a safe train handover between the accepting and handing-over communication centres. These requirements are expressed in the Requirement table below as safety requirements $\mathsf{SAF}_{3,4}$.

| | |
|---|---|
| $\mathsf{SAF}_1$ | A communication-centre will issue a movement authority that ensures a safe separation of trains. |
| $\mathsf{SAF}_2$ | At all times the train must remain within the issued movement authority. |
| $\mathsf{SAF}_3$ | A train will only be allowed to transition to the adjacent signalling system if the accepting communication centre has provided a movement authority extension. |
| $\mathsf{SAF}_4$ | A handing over communication centre can only remove a train from its map, if it is confirmed that an accepting communication centre has completed a take-over procedure. |

Table 6.6: Safety requirements of the heterogeneous signalling system model

### 6.2.2 Formal Event-B Model of a Heterogeneous Railway Signalling System

In this section, we overview the formal Event-B model of the heterogeneous signalling system specified in the previous section. The model refines the generic communication-based model with an additional machine model and several context models, which capture a system-level transition protocol.

**Context Model.** In the heterogeneous signalling model, we extend the context model of the generic hybrid signalling model by including additional field elements and defining communication centre types (show in Listing 6.30). Firstly, we introduce a communication centre type function $\mathsf{csst}$, which maps all communication centres to a type, represented as an enumerated set of two elements $\mathsf{ACC}$, $\mathsf{HNO}$ representing accepting and handing-over types. Furthermore, we define a level-transition announcement balise as a new object ($\mathsf{LTA}$), which has an associated position function $\mathsf{ltap}$, which maps balises to a one-dimensional position.

**Machine Model.** The heterogeneous signalling model refines generic hybrid machine models by introducing new events, which model message exchanges described in a sequence diagram depicted in Figure 6.7 (Listing 6.31). According to the proposed communication modelling patterns, for every new message we introduce a channel variable in the machine model and associated message context model.

Listing 6.30: Communication signalling model context

**CONTEXT** HetModelContex
**EXTENDS**
  CommunicationCtx
**SETS**
  CTYPE, STATUS, RTYPE, LTA, RTS
**CONSTANTS**
  FREE, RESERVED, NRM, INT, CTYPE, csst, ltap, rtyp
**AXIOMS**
  $axm_1 : finite(LTA)$
  $axm_2 : finite(RTS)$
  $axm_3 : partition(STATUS, \{FREE\}, \{RESERVED\})$
  $axm_4 : partition(CTYPE, \{ACC\}, \{HNO\})$
  $axm_5 : partition(RTYPE, \{NRM\}, \{INT\})$
  $axm_6 : csst \in CCS \rightarrow CTYPE$
  $axm_7 : ltap \in LTA \rightarrow \mathbb{R}^+$
  $axm_8 : rtyp \in RTS \rightarrow RTYPE$
  $axm_9 : \forall r1, r2 \cdot r1 \in RTS \wedge r2 \in RTS \wedge r1 \neq r2 \Rightarrow rtyp(r1) = INT \wedge rtyp(r2) = INT$
  $axm_{10} : INT \in ran(rtyp)$
  $\vdots$

Listing 6.31: The machine model of the heterogeneous signalling system

**MACHINE** HetModelMachine
**REFINES** SignallingModel
**VARIABLES**
  pos, rrs, rmem, rte, rrt $\cdots$
**INVARIANTS**
  $inv_1$ $pos \subseteq POS$
  $inv_2$ $rrs \subseteq RRS$
  $inv_3$ $rmem \in CCS \leftrightarrow TRN$
  $inv_4$ $rte \in RTS \rightarrow STATUS$
  $inv_5$ $rrt \in RTS \rightarrowtail TRN$

For example, a train_position_message_LTA message is shown in Listing 6.32, which introduces a new message type POS which is sent to a communication centre (message destination function posd) from a train passing over LTA balise.

Listing 6.32: The context model of the position_report message

```
train_position_message_LTA
SETS
    POS
CONSTANTS
    poss, posd
AXIOMS
    axm1    poss ∈ POS ↠ TRN
    axm2    posd ∈ POS ↠ CCS
    axm3    ∀s, d · s ∈ TRN ∧ d ∈ CCS ∧ v ∈ POS ⇒ ∃m · poss(m) = s ∧ posd(m) = d
```

In Listings 6.33 and 6.34, we provided an excerpt of the heterogeneous Event-B model. Events demonstrate how a generic hybrid signalling model can be extended with new communication modelling events. With event Train_Sense_LTA we model a train passing over a LTA balise and starting a handover procedure. The event considers a train tr whose position at time ntp(tr)(t) equals position of an level-transition balise ran(ltap) specified by guard $grd_2$. The train tr will send a message ps to a communication centre it is connected $grd_3$. The position report with a level transition announcement message was defined according to the communication modelling pattern and is described in Listing 6.32.

Listing 6.33: An event modelling a train passing over a level announcement balise

```
Train_Sense_LTA
ANY
   tr, cc, ps
WHERE
   grd1 : tr ∈  TRN
   grd2 : ntp(tr)(t) = ran(ltap)
   grd3 : tr ∈ rmem[{cc}]
   grd4 : ps ∉ POS
   grd5 : poss(ps) = tr
   grd6 : posd(ps) = cc
THEN
   act1 : pos := pos ∪ {ps}
END
```

Listing 6.34: An event modelling communication centre requesting a route locking upon receiving level transition announcement

```
CC_Request_Route
ANY
  cch, cca, ps, rr
WHERE
  grd$_1$ : $cch \in CCH$
  grd$_2$ : $ps \in pos$
  grd$_3$ : $rr \notin rrs$
  grd$_4$ : $rrss(rr) = posd(cch)$
  grd$_5$ : $rrsd(rr) = cca$
THEN
  act$_1$ : $rrs := rrs \cup \{rr\}$
  act$_2$ : $pos := pos \setminus \{ps\}$
END
```

According to the sequence diagram, once a handing-over communication centre receives a position report message with level transition announcement it will send route request message to an accepting communication centre. In the model, we capture this with a CC_Request_Route event, where a new message rr is sent to an accepting cca and the received position report message ps is removed from the channel with action act$_2$.

**Proof Statistics** The model contained only a single new machine model and 11 additional context models, which defined messages of the handover protocol. The completed heterogeneous signalling model generated 32 new proof obligations, which were mostly discharged interactively.

| Model | \|POs\| | Auto. | Inter. |
|---|---|---|---|
| heterogeneos_model (1m. + 11c.) | 32 | 10 | 22 |
| hybrid_model (4m. + 1c.) | 61 | 23 | 38 |
| communication_model (1m. + 8c.) | 49 | 18 | 31 |
| Total | 142 | 51 | 91 |

Table 6.7: Proof statistics of the Event-B heterogeneous railway signalling model

The main safety properties of the heterogeneous signalling system (Table 6.6) were formally defined as three Event-B invariants. The safety requirement $\mathsf{SAF}_1$ was split into two invariants. Firstly, we demonstrated that $\mathsf{SAF}_1$ holds by proving that the first railway block in the accepting signalling area is reserved, before a movement authority message is sent by the accepting communication-centre (invariant $\mathsf{SAF}_3$ in Listing 6.35). The second invariant was concerned with a train safe separation in the fixed-block signalling area and was proved by expressing it as a railway block mutual exclusion property ($\mathsf{inv}_5$ in Listing 6.31). We did not require a third invariant, which related to the handing-over signalling area, as it was already proved in the generic communication-based signalling model. Similarly, as the speed controller of the train was not changed, ($\mathsf{SAF}_2$ in Listing 6.35) was preserved in the heterogeneous signalling model.

Listing 6.35: The safety invariants of the heterogeneous signalling system

**MACHINE** HetModelMachine

**INVARIANTS**

$\mathsf{SAF}_2$ $\forall t, tr \cdot t \in \mathbb{R}^+ \land tr \in \mathsf{TRN} \Rightarrow \mathsf{ntp}(tr)(t) \leq \mathsf{nEoA}(tr)$

$\mathsf{SAF}_3$ $\forall ps, rr \cdot ps \in \mathsf{pos\_lt} \land \mathsf{rtyp}(rr) = \mathsf{INT} \Rightarrow \mathsf{rte}(rr) = \mathsf{RESERVED}$

$\mathsf{SAF}_4$ $\forall cc, tc \cdot \mathsf{ccst}(cc) = \mathsf{HNO} \land tc \in \mathsf{toc} \land \mathsf{tocd}(tc) = cc \Rightarrow \mathsf{tocv}(cc) \in \mathsf{rmem}(cc)$

The safety invariant $\mathsf{SAF}_3$ (Listing 6.35) was expressed as a formal property which states that if a position message ps (with a level-transition announcement) has been sent, a route rr, which interfaces adjacent signalling system has to be reserved.

The final safety property ($\mathsf{SAF}_4$) was also proved interactively by proving Event-B invariant $\mathsf{SAF}_4$ (Listing 6.35) which formally states that if a take-over completion (tc) message has been sent to a handing-over communication centre ($\mathsf{tc} \in \mathsf{toc}$), a train ($\mathsf{tocv}(\mathsf{cc})$) must still be in the memory (map) of the handing-over communication centre ($\mathsf{rmem}(\mathsf{cc})$).

## 6.3 Evaluation Summary and Discussion

In this chapter, we presented two case studies, which have been formally developed according to the proposed development methodology. In the first study, we developed a novel distributed resource allocation protocol, which can be used in decentralised railway signalling systems to safely allocate railway sections to trains. The methodology in this case study was specifically used to prototype a new signalling system starting with initial system specifications and requirements. The initial functional pivot Event-B model of the protocol, which is developed in the second methodology step, played a crucial role in enabling us to identify problematic scenarios where a liveness property could not be ensured. In particular, the ProB animation and model checking Rodin tool provided a pragmatic method to discovering protocol issues without requiring us to complete formal proofs at this development stage. The problematic scenarios have been resolved by refining a resource allocation protocol with a two-stage resource locking mechanism. And once we had the confidence in the protocol correctness, the final distributed protocol version was formally modelled and formally proved by discharging proof obligations. However, this case study has also demonstrated that probabilistic system properties cannot easily be expressed in Event-B modelling language, and therefore, one of the protocol liveness properties was verified externally using the PRISM stochastic model checker.

The focus of the second case study was the evaluation of the proposed methodology when it is applied to formally developing heterogeneous signalling systems. In the section about the second study, we first described the heterogeneous signalling system we considered in the case study, which included the specification of the signalling handover protocol and safety requirements (as suggested by the methodology). Then, we described the functional pivot Event-B model of the heterogeneous signalling system, which was developed by extending the generic communication-based signalling model and utilizing communication modelling patterns. Lastly, the safety requirements were formally expressed as Event-B invariants and interactively proved in the Rodin platform.

In overall, we think that the two case studies presented in this chapter are sufficient to discuss and demonstrate the applicability of the methodology to formally developing signalling systems. First of all, the case studies were selected to capture two dominant formal system development approaches, namely, an iterative (prototyping) development method and formal development of a fully specified system. Secondly, both case studies extensively evaluated communication modelling patterns, specifically, if the introduced patterns are complete and could capture a majority of communication scenarios. Similarly, in the second study, we evaluated the refinement process of the generic hybrid communication-based signalling model with heterogeneous signalling specifications. The second case study has demonstrated that the generic communication-based signalling model could be easily refined into a fixed-block signalling system, thus enabling modelling different configurations of heterogeneous signalling systems. Lastly, we emphasised that it is crucial that the methodology enables formal verification of discrete, hybrid, and stochastic system-level properties. In order to evaluate the formal verification aspect of the methodology, case studies were selected so it was required to formally verify safety properties on hybrid, probabilistic and discrete system aspects.

In Section 1.3 and Section 2.1.4 we elicited key research questions and requirements for the multifaceted methodology. In the following paragraphs, we revisit and discuss research questions and methodology requirements we raised in this thesis based on results obtained from the evaluation study.

**Requirement**$_1$ The formal engineering methodology should support a system-level modelling as the safety of heterogeneous signalling systems relies on the correctness of subsystems and their interaction.

**Requirement**$_3$ The formal engineering methodology should make it possible to specify and verify system-level properties.

The importance of verifying cyber-physical systems at the system-level was expressed in the seminal Baheti and Gill paper [13] in which the authors expressed great concerns about the *divide* and *conquer* application of formal methods to the cyber-physical system development. Therefore, two of the key methodology requirements, Requirement$_1$ and Requirement$_3$, are both concerned with the modelling and verification of cyber-physical systems at the system-level.

We think that the methodology's evaluation, in particular, a heterogeneous signalling system modelling study has demonstrated that the Event-B modelling language and developed modelling patterns provide a unified system-level signalling modelling framework. In the functional pivot Event-B model, different signalling subsystems can be expressed as set-based objects in the context models, whereas dynamic sub-system behaviour can be expressed as parametrised machine model events. Crucially, a system abstraction and stepwise modelling principles of Event-B enable specifying parts of the signalling system at the chosen level of detail, which helps to address the trade-off between formal development productivity and the need to consider all sub-systems. An example of this was modelling a heterogeneous signalling system, in which communication centres have been modelled in much greater detail compared to interlocking systems. The sub-system refinement decision was based on the safety properties we were interested in proving, which in that case study were essentially concerned with the interactions between rolling stock and communication-centres.

**Requirement**$_2$ The methodology should provide a generic formal signalling model and support a model extension with specific signalling solutions.

The evaluation demonstrates that the proposed systematic formal development method is beneficial in both formal development scenarios (iterative and *linear*). The stepwise development principle of the methodology was particularly useful in the iterative development scenario (case study 1) where a number of protocol issues were discovered. Due to the stepwise modelling principle, re-modelling efforts were significantly reduced as only a part of the complete functional pivot Event-B model needed to be reworked. For example, issues discovered in the $\mathsf{stage}_1$ of the distributed protocol only affected final refinement step $\mathsf{M}_4$ and so machine models $\mathsf{M}_{0..3}$ and associated (completed) proofs were preserved.

Similarly, in the second case study, a stepwise development principle of the methodology and generic communication-based signalling system model helped to reduce the formal development effort of the heterogeneous signalling system. The heterogeneous signalling system was modelled by extending existing context and machine Event-B models with signalling specifications particular to the modelled scenario. On the other hand, the second case study has demonstrated that the new signalling objects might need to be introduced or be parametrised, for example, level-transition balises.

We believe that evaluation results have demonstrated (and addressed Requirement$_2$) that: 1) the methodology provides a mechanism to extend a generic communication-based signalling model via Event-B refinement and 2) a generic model is sufficiently generic to be refined into the signalling system with different operation principles.

**Requirement**$_4$ The methodology and generic formal signalling model should make it possible to capture and reason about continuous, discrete and stochastic aspects of a system.

The multi-aspect reasoning requirement relates to the cyber-physical nature of heterogeneous signalling systems. This methodology requirement was addressed by developing a generic hybrid signalling model with hybrid Event-B modelling patterns introduced by Dupont et al. [59]. In the second case study (and also Chapter 5) we demonstrated that the methodology does support reasoning about hybrid system aspects, specifically, that Event-B invariants can be expressed so they refer to discrete and continuous system variables. In addition to specified invariants, one is also required to prove well-definedness and feasibility (ODE solution existence) proof obligations related to continuous system aspects. The evaluation has demonstrated that proving properties about hybrid system aspects is a major challenge (Tables 5.1 and 6.7). Nonetheless, because of the stepwise Event-B modelling principle, the generic hybrid signalling model is only required to be proved once and only proof obligations of its refinements steps (e.g. case study 2) are required to be discharged.

In the first case study, we were also required to prove a protocol termination property. However, we found it difficult to encode stochastic elements into the Event-B protocol model or specify a convergence variant (often used to prove termination). Therefore, it was decided to model and verify part of the protocol $\mathtt{stage}_1$ in a PRISM probabilistic model checker. We believe that this is an important methodology and the Event-B modelling limitation, which should be addressed in future work.

**Methodology Limitations** Even though we believe that the proposed methodology is a step forward towards a practical formal methods based engineering methodology of cyber-physical railway signalling systems, we identified a few key methodology limitations. In the following paragraphs, we discuss these limitations, whereas the next chapter will discuss future work, which would address issues raised in this section.

**Research Question**$_3$ Are there (or is it possible to develop) adequate and scalable verification tools to reason about hybrid, stochastic heterogeneous cyber-physical railway signalling systems properties?

**Requirement**$_4$ The methodology and generic formal signalling model should make it possible to capture and reason about continuous, discrete, and stochastic aspects of a system.

Automating formal verification has been one of the major challenges in the field of formal methods, particularly automatically verifying hybrid models. One of the main research questions (**Research Question**$_3$) of this research was exactly concerned with the scalability and adequacy of verification tools. Furthermore, one of the requirements (**Requirement**$_4$) for the methodology was the ability to reason about continuous, discrete and stochastic aspects of a system. Over the years, a number of verification tools have been developed and integrated into the Rodin platform (e.g. [110, 111]) which have reduced the number of interactive proofs.

However, the verification of a heterogeneous signalling system and generic hybrid signalling model (see Chapter 5.2.6) have demonstrated that existing Event-B verification tools do not achieve adequate verification automation (see Table 5.1 and Table 6.7). Similar verification results have been reported in other studies, which attempted to verify hybrid systems in Event-B [61,119]. Although the methodology does enable reasoning about continuous system properties as imposed by **Requirement**$_4$, a poor verification automation of hybrid signalling models would hinder the methodology's usability, particularly for verifying larger hybrid signalling models.

**Requirement**$_2$ The methodology should provide a generic formal signalling model and support a model extension with specific signalling solutions.

The Event-B method, which is at the core of the proposed methodology, provides a refinement-based model development method. The formal development methodology provides a generic (hybrid) communication-based signalling model, which can then be refined in order to capture a specific heterogeneous or homogeneous signalling system. However, the proposed methodology does not currently provide refinement patterns or specify how a generic signalling model should be extended. Similarly, the communication modelling patterns we introduced in this research also do not impose a specific refinement process of the distributed system (distributed system refinement we briefly discussed in [126]). In the recent work [127], the hybrid Event-B modelling patterns we used to develop a generic hybrid signalling model have been extended with refinement patterns for modelling distributed-hybrid systems.

**Research Question**$_1$ Can heterogeneous cyber-physical railway signalling systems be pragmatically engineered by using formal methods?

Although, some contributions of this research (e.g. communication modelling patterns) can be used to formally model a variety of systems, the methodology we proposed is railway domain specific. However, to a certain extent, the practicality of the method depends on the capability to *communicate* useful feedback (e.g. discovered signalling issues) to (railway signalling) system engineers in the form they can interpret. Furthermore, it is generally accepted that only a small percentage of systems engineers have a background in formal methods. Therefore, we believe that one of the methodology's limitations is inadequate feedback, especially when a problem is discovered in the model. For example, a feedback issue has been addressed in work by Iliasov et al. [128] in which a deductive verification method is used to verify signalling systems. The results (e.g. undischarged proof obligations) are then used to generate a verification report th highlights problems on the considered railway layout and the excerpt of signalling software.

# Chapter 7

# Conclusions

This chapter summarises the key findings of this research. In particular, this chapter focuses on summing up the main insights acquired from developing and evaluating our proposed methodology. In this chapter, we also present the potential research directions for extending the methodology for it to be more applicable in a model-based development of real-world cyber-physical railway signalling systems.

## 7.1 Summing Up

In this research, we attempted to address the challenges of applying formal methods for a model-based development of cyber-physical systems. In particular, the research described in this thesis focused on a formal development and the safety verification of heterogeneous cyber-physical railway signalling systems. The difficulty of a model-based cyber-physical system development stems from the complex nature of cyber-physical systems which have a deeply intertwined physical processes, computation and networking system aspects that have to be captured by a system-level formal model. Furthermore, the heterogeneity of the modern communication-based railway signalling systems and their safety-critical nature further complicate the adoption of formal methods in developing these systems.

The overall objective of this research was to address the challenge of applying formal methods to designing railway signalling systems by formulating a formal development methodology of cyber-physical railway signalling systems. It was essential that the methodology not only reduces the effort of applying formal methods, but also facilitates a rigorous and systematic model-based signalling system development. The main contribution of this research is summarised in the following:

c1) A model-driven development methodology of the heterogeneous cyber-physical railway signalling systems

At the core of the proposed methodology, a formal specification language facilitates a rigorous formal development of signalling systems. In this thesis, we presented our evaluation of different classes of formal methods and formal specification languages, and discussed their suitability to modelling cyber-physical railway signalling systems. The analysis concluded that the Event-B method (and the Rodin tool-set) with an expressive modelling language, stepwise and proof-based modelling approach would be the most suitable for a formal modelling and verification of cyber-physical railway signalling systems.

Based on the requirements for the methodology and formal specification language evaluation, the formal development methodology was introduced with the Event-B formalism being at the core of the formal railway signalling modelling process. In order to provide a more systematic and practical formal modelling process of heterogeneous railway signalling systems, we described three technical methodology contributions:

c2) Event-B communication modelling patterns

c3) Event-B hybrid railway signalling modelling patterns

c4) Generic communication-based railway signalling Event-B model

The proposed formal development method and modelling patterns were rigorously evaluated with the two case studies, which captured different system development principles (iterative and sequential), distinct signalling concepts and safety requirements. The results have demonstrated that this methodology with the generic communication-based railway signalling Event-B model and patterns reduce formal modelling and verification effort by:

1) providing a once and for all proved formal generic signalling model, which is refined to capture a specific signalling system

2) providing communication modelling patterns, which implement a systematic method to modelling diverse message sending and receiving scenarios

In particular, the heterogeneous signalling system modelling study has demonstrated that the generic communication-based signalling model can be easily extended to capture different heterogeneous signalling configurations by instantiating communication modelling pattern events and refining system context model. The rolling stock plant dynamics model can also be modified by further constraining plant model parameters in the Train context model. Overall, we think that the generic Event-B signalling model provides a reusable formal development framework, which increases the productivity of formally developing systems.

In this thesis, we also demonstrated that the methodology can be effectively used for an early system prototyping (case study 1) where system specifications and requirements are incomplete, and for a formal system development with fully defined requirements (case study 2). In the first case study, the Rodin tools such as the ProB plug-in [112] were pragmatically used for discovering system or modelling issues in the early modelling stages where discharging proof obligations might be too cumbersome. The first case study has also demonstrated that communication modelling patterns make it possible to capture different message sending/receiving scenarios in a systematic way, and therefore, applied to modelling a variety of distributed communicating systems.

The proposed development methodology and associated contributions of this research facilitate an application of formal methods to developing and verifying the safety of heterogeneous (and homogeneous) hybrid signalling systems. We believe that, the introduced modelling principles, patterns and a generic hybrid signalling model, reduce formal modelling effort by providing a more systematic and reusable model-based development of heterogeneous signalling systems. Despite the limitations of the proposed methodology, which were discussed in the previous chapter, we think that overall our methodology provides a more practical application of formal methods to the railway signalling domain.

More generally, research results presented in this thesis show that formal methods can provide an adequate and unified theoretical foundation to capture physical processes, computation and networking system aspects of cyber-physical systems. However, even with the available modelling patterns and generic domain models formal modelling and verification remain a major obstacle in further adoption of formal methods. In particular, an automatic formal verification of formal models, which combine continuous and discrete behaviour remains a challenge. The problem is particularly relevant for a safety verification of cyber-physical transportation systems for which safety requirements are often associated with a physical behaviour of a plant, which is controlled by a discrete controller.

In the following section, we discuss future works that could be undertaken to further advance this research.

## 7.2 Future Work

In this thesis, we identified a few potential future research directions, which could address the limitations of the methodology (discussed in Chapter 6) and increase its applicability to formally developing real-world heterogeneous signalling systems.

As discussed by Kim And Kumar [3] one of the fundamental research challenges in a model-based development of cyber-physical systems is the integration of different theoretical frameworks and modelling/verification tools. The evidence of this research can lead one to conclude that the Event-B theoretical framework together with the Rodin Theory plug-in can provide a unified formal modelling and verification framework. Nonetheless, a further work is still necessary, especially a development of (and integrating into a proposed methodology) Event-B theories that capture stochastic and heterogeneous system aspects. In particular, stochastic theory and modelling patterns could address issues raised in the first case study where an external PRISM stochastic modelling framework had to be used.

The proposed formal development methodology does not provide a systematic process to refine the generic communication-based signalling model with specific signalling configurations. Formalising refinement steps would fully utilise the benefits of the Event-B stepwise modelling and reduce modelling and verification effort. In the paper [126], we briefly discussed initial ideas for a systematic refinement of distributed Event-B system models. We believe that the future work of defining refinement patterns can be built around refinement ideas in our paper [126] and refinement patterns of distributed hybrid systems described in a work by Dupont et al. [127]. Furthermore, as we demonstrated in the second case study, the refinement of the generic signalling model is mostly concerned with refining communication part of the system. The modelling effort with our methodology could be significantly reduced by providing an automatic refinement of communication models from a high-level protocol specification, for example, sequence diagrams, which are a widely used method to specify protocols in the railway industry.

In the work by Savicks [113], the Event-B modelling framework was extended with a functionality to export Event-B models as a Functional Mock-up Unit which then could be co-simulated with other models using the Functional Mock-Up Interface standard. We believe that by integrating the Event-B co-simulation and co-modelling plug-in within our proposed methodology we could address an inadequate feedback issue by providing seamless model translation to domain-specific tools, such as railway modelling and simulation tool SafeCap [129]. In order to integrate the Event-B co-simulation plug-in within our methodology, the co-simulation plug-in would have to be extended with a functionality to export Event-B models which include new theories developed via the Theory plug-in. Furthermore, an extension of the SafeCap tool with a Functional Mock-Up Interface standard support would also be required. Nonetheless, because of the standard popularity in various industries, we believe that this methodology extension would increase the prospects of deploying methodology in a real industrial setting.

Overall, we believe that the main barrier for a practical deployment of our proposed development methodology is the formal verification of hybrid signalling system properties. We believe that the challenge of automatically verifying hybrid Event-B models can be addressed by:

1) Improving the existing Rodin verification plug-ins

   The verification could be improved by extending an existing Rodin verification plug-in [111] (based on the Why3 [130] umbrella prover) with additional libraries which include Reals and continuous system operators defined in Section 5.1.

2) Modifying the continuous system Event-B theory described in Section 5.1

   The Rodin Theory plug-in not only enables users defining new operators, expressions and datatypes but also allows to specify rewrite and inference proof rules. We believe that by including new proof rules into the continuous system Event-B theory, the number of interactive proofs, which relate to the continuous model variables, would be reduced.

3) Integrating automatic verification tools, which are based on the reachability analysis into the Rodin platform.

   Over the last few decades, formal verification tools, which are based on a system reachability analysis have been developed. The advantage of these tools is their fully automatic verification characteristic, which often comes at the cost of the scalability and accuracy. However, in the last several years, verification algorithms and tools such as SpaceEX [76] based on the reachability analysis have been significantly improved and can verify hybrid systems with tens of variables in minutes.

Lastly, the future work should further evaluate our formal development methodology with more complex heterogeneous systems.

# Bibliography

[1] J. Potocki de Montalk, “Computer software in civil aircraft,” *Microprocessors and Microsystems*, vol. 17, no. 1, pp. 17 – 23, 1993.

[2] T. A. Henzinger and J. Sifakis, “The discipline of embedded systems design,” *Computer*, vol. 40, no. 10, pp. 32–40, 2007.

[3] K. Kim and P. R. Kumar, “Cyber–Physical Systems: A Perspective at the Centennial,” *Proceedings of the IEEE*, vol. 100, pp. 1287–1308, May 2012.

[4] A. H. Cribbens, “Solid-state interlocking (SSI): an integrated electronic signalling system for mainline railways,” *IEE Proceedings B - Electric Power Applications*, vol. 134, no. 3, pp. 148–158, 1987.

[5] E. A. Lee, “Cyber physical systems: Design challenges,” in *11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pp. 363–369, 2008.

[6] J. C. Knight, “Safety critical systems: challenges and directions,” in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pp. 547–550, May 2002.

[7] V. Gunes, S. Peter, T. Givargis, and F. Vahid, “A survey on concepts, applications, and challenges in cyber-physical systems,” *KSII Transactions on Internet and Information Systems*, vol. 8, pp. 4242–4268, December 2014.

[8] M. Broy and A. Schmidt, “Challenges in engineering cyber-physical systems,” *Computer*, vol. 47, pp. 70–72, February 2014.

[9] H. K. Bruce, E. Lee, I. Lee, A. Mok, G. Pappas, R. Rajkumar, L. S. Raymond, S. A. Vincentelli, K. Shin, J. Stankovic, J. Sztipanovits, W. Wolf, and W. Zhao, "Cyber-physical systems: Executive summary," *CPS Steering Group, Arlington*, 2018.

[10] T. A. Henzinger and J. Sifakis, "The embedded systems design challenge," in *FM 2006: Formal Methods* (J. Misra, T. Nipkow, and E. Sekerinski, eds.), pp. 1–15, Springer Berlin Heidelberg, 2006.

[11] E. Lee, "The past, present and future of cyber-physical systems: A focus on models," *Sensors (Basel, Switzerland)*, vol. 15, pp. 4837–4869, March 2015.

[12] J. Lygeros and M. Prandini, "Stochastic hybrid systems: A powerful framework for complex, large scale applications," *European Journal of Control*, vol. 16, no. 6, pp. 583 – 594, 2010.

[13] R. Baheti and H. Gill, "Cyber-physical systems," *The Impact of Control Technology*, pp. 161 – 166, 2011.

[14] ERTMS User Group, "UNISIG: ERTMS/ETCS: System requirements specification v. 3.4.0," April 2002.

[15] IEEE Std 1474.1-2004, "IEEE standard for Communications-Based Train Control (CBTC) performance and functional requirements," 2005.

[16] Crossrail Project Website. `https://www.crossrail.co.uk/`.

[17] National Audit Office, "Completing Crossrail." `https://www.nao.org.uk/report/crossrail/`, May 2019.

[18] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," *ACM Comput. Surv.*, vol. 41, no. 4, October 2009.

[19] J.-R. Abrial, *The B-book: Assigning Programs to Meanings*. New York, USA: Cambridge University Press, 1996.

[20] European Committee for Electrotechnical Standardization, "EN50128 railway applications - software for railway control and protection systems," 1997.

[21] A. Fantechi, "Twenty-five years of formal methods and railways: What next?," in *Software Engineering and Formal Methods* (S. Counsell and M. Núñez, eds.), pp. 167–183, Springer International Publishing, 2014.

[22] R. Alur, "Formal verification of hybrid systems," in *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pp. 273 – 278, ACM, 2011.

[23] B. Silva, O. Stursberg, B. Krogh, and S. Engell, "An assessment of the current status of algorithmic approaches to the verification of hybrid systems," in *Proceedings of the 40th IEEE Conference on Decision and Control*, vol. 3, pp. 2867–2874, IEEE, 2001.

[24] G. Theeg and S. Vlasenko, *Railway Signalling & Interlocking: International Compendium*. Edition EURAIL press, PMC Media House GmbH, 2017.

[25] European Commission, "Commission Decision 2001/260/EC of 21 March 2001 on the basic parameters of the command-control and signalling sub-system of the trans-European high-speed rail system referred to as ERTMS characteristicsin Annex II(3)to Directive96/48/EC," *Official Journal of the European Union L.93*, pp. 53 – 56, 2011.

[26] ERTMS User Group, "UNSIG: RBC-RBC Safe Communication Interface Subset-098 v. 3.0.0," February 2012.

[27] ERTMS User Group, "UNISIG: Specific Transmission Module FFFIS Subset-035 v. 3.2.0," December 2015.

[28] D. Milburn, "ETCS & CBTC interfaces - Crossrail signalling." Presentation Available Online: `www.networkrailconsulting.com/news-and-publications/publications`, December 2015.

[29] D. Broman, E. A. Lee, S. Tripakis, and M. Törngren, "Viewpoints, formalisms, languages, and tools for cyber-physical systems," in *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, MPM 12, pp. 49–54, ACM, 2012.

[30] S. K. Khaitan and J. D. McCalley, "Design techniques and applications of cyber–physical systems: A survey," *IEEE Systems Journal*, vol. 9, no. 2, pp. 350–365, 2015.

[31] X. Zheng, C. Julien, M. Kim, and S. Khurshid, "Perceptions on the state of the art in verification and validation in cyber-physical systems," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2614–2627, 2017.

[32] R. Nikoukhah and S. Steer, "SCICOS A Dynamic System Builder and Simulator User's Guide - Version 1.0," Research Report RT-0207, INRIA, June 1997.

[33] H. Elmqvist, S. E. Mattsson, and M. Otter, "Modelica - a language for physical system modeling, visualization and interaction," in *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, pp. 630–639, 1999.

[34] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nystrom, L. Saldamli, D. Broman, and A. Sandholm, "Openmodelica - a free open-source environment for system modeling, simulation, and teaching," in *2006 IEEE Conference on Computer Aided Control System Design*, pp. 1588–1595, 2006.

[35] J. F. Broenink, "20-sim software for hierarchical bond-graph/block-diagram models," *Simulation Practice and Theory*, vol. 7, no. 5, pp. 481 – 492, 1999.

[36] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clau, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. v. Peetz, S. Wolf, A. S. Gmbh, Q. Berlin, F. Scai, and S. Augustin, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th International Modelica Conference*, pp. 105 – 114, March 2011.

[37] P. G. Larsen, J. Fitzgerald, J. Woodcock, R. Nilsson, C. Gamble, and S. Foster, "Towards semantically integrated models and tools for cyber-physical systems design," in *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications* (T. Margaria and B. Steffen, eds.), pp. 171–186, Springer International Publishing, 2016.

[38] A. Sorensen and H. Gardner, "Programming with time: Cyber-physical programming with impromptu," *SIGPLAN Not.*, vol. 45, no. 10, pp. 822 – 834, 2010.

[39] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language lustre," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, 1991.

[40] G. Berry, *The Foundations of Esterel*, pp. 425–454. Cambridge, MA, USA: MIT Press, 2000.

[41] N. Halbwachs, *Synchronous Programming of Reactive Systems*. Berlin, Heidelberg: Springer-Verlag, 2010.

[42] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: a time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 84–99, 2003.

[43] N. Halbwachs, F. Lagnier, and C. Ratel, "Programming and verifying real-time systems by means of the synchronous data-flow language lustre," *IEEE Trans. Softw. Eng.*, vol. 18, pp. 785 – 793, Sept. 1992.

[44] A. Burns and A. Wellings, *Concurrent and Real-Time Programming in Ada*. USA: Cambridge University Press, 3rd ed., 2007.

[45] G. Bollella and J. Gosling, "The real-time specification for java," *Computer*, vol. 33, no. 6, pp. 47–54, 2000.

[46] B. Bouyssounouse and J. Sifakis, *Programming Languages for Real-Time Systems*, pp. 338–351. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[47] T. W. Hnat, T. I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse, "Macrolab: A vector-based macroprogramming framework for cyber-physical systems," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys 08, pp. 225 – 238, Association for Computing Machinery, 2008.

[48] R. Gummadi, O. Gnawali, and R. Govindan, "Macro-programming wireless sensor networks using kairos," in *Distributed Computing in Sensor Systems* (V. K. Prasanna, S. S. Iyengar, P. G. Spirakis, and M. Welsh, eds.), (Berlin, Heidelberg), pp. 126–140, Springer Berlin Heidelberg, 2005.

[49] OMG, "Unified Modeling Language Superstructure, Version 2.5.1.," 2017.

[50] OMG, "Modeling and Analysis of Real-time and Embedded systems (MARTE), Version 1.1.," 2019.

[51] M. Z. Iqbal, S. Ali, T. Yue, and L. Briand, “Applying UML/MARTE on industrial projects: Challenges, experiences, and guidelines,” *Softw. Syst. Model.*, vol. 14, pp. 1367 – 1385, Oct. 2015.

[52] Y. Kesten and A. Pnueli, “Timed and hybrid statecharts and their textual representation,” in *Formal Techniques in Real-Time and Fault-Tolerant Systems* (J. Vytopil, ed.), (Berlin, Heidelberg), pp. 591–620, Springer Berlin Heidelberg, 1991.

[53] K. Bae, P. C. lveczky, T. H. Feng, E. A. Lee, and S. Tripakis, “Verifying hierarchical Ptolemy II discrete-event models using Real-Time Maude,” *Science of Computer Programming*, vol. 77, no. 12, pp. 1235 – 1271, 2012.

[54] B. I. Silva, K. Richeson, B. Krogh, and Alongkrit, “Modeling and verifying hybrid dynamic systems using checkmate,” in *4th International Conference on Automation of Mixed Processes*, vol. 4, pp. 1 – 7, De Gruyter, 2001.

[55] J. Hoenicke and E.-R. Olderog, “CSP-OZ-DC: A combination of specification techniques for processes, data and time,” *Nordic J. of Computing*, vol. 9, pp. 301 – 334, Dec. 2002.

[56] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2013.

[57] C. B. Jones, *Systematic Software Development Using VDM (2nd Ed.)*. USA: Prentice-Hall, Inc., 1990.

[58] D. Cansell, D. Méry, and J. Rehm, “Time constraint patterns for Event-B development,” in *B 2007: Formal Specification and Development in B* (J. Julliand and O. Kouchnarenko, eds.), pp. 140 – 154, Springer Berlin Heidelberg, 2006.

[59] G. Dupont, Y. Aït-Ameur, M. Pantel, and N. K. Singh, “Proof-based approach to hybrid systems development: Dynamic Logic and Event-B,” in *Abstract State Machines, Alloy, B, TLA, VDM, and Z* (M. Butler, A. Raschke, T. S. Hoang, and K. Reichl, eds.), pp. 155 – 170, Springer International Publishing, 2018.

[60] R. Banach, M. Butler, S. Qin, N. Verma, and H. Zhu, “Core hybrid Event-B I: single hybrid Event-B machines,” *Science of Computer Programming*, vol. 105, pp. 92 – 123, 2015.

[61] G. Babin, Y. Aït-Ameur, S. Nakajima, and M. Pantel, "Refinement and proof based development of systems characterized by continuous functions," in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pp. 55–70, Springer, 2015.

[62] The ADVANCE project. Online Website: `www.advance-ict.eu/`.

[63] M. Verhoef, P. G. Larsen, and J. Hooman, "Modeling and validating distributed embedded real-time systems with VDM++," in *FM 2006: Formal Methods* (J. Misra, T. Nipkow, and E. Sekerinski, eds.), pp. 147–162, Springer Berlin Heidelberg, 2006.

[64] P. G. Larsen, J. Fitzgerald, J. Woodcock, P. Fritzson, J. Brauer, C. Kleijn, T. Lecomte, M. Pfeil, O. Green, S. Basagiannis, and A. Sadovykh, "Integrated tool chain for model-based design of cyber-physical systems: The INTO-CPS project," in *2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pp. 1 – 6, 2016.

[65] L. Lamport, "Hybrid systems in TLA+," in *Hybrid Systems* (R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds.), pp. 77–102, Springer Berlin Heidelberg, 1993.

[66] C. J. Fidge, "Specification and verification of real-time behaviour using Z and RTL," in *Formal Techniques in Real-Time and Fault-Tolerant Systems* (J. Vytopil, ed.), pp. 393–409, Springer Berlin Heidelberg, 1991.

[67] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL — a tool suite for automatic verification of real-time systems," in *Hybrid Systems III* (R. Alur, T. A. Henzinger, and E. D. Sontag, eds.), pp. 232–243, Springer Berlin Heidelberg, 1996.

[68] M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I. Lee, "Model-driven safety analysis of closed-loop medical systems," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 3–16, 2014.

[69] L. Shan, Y. Wang, N. Fu, X. Zhou, L. Zhao, L. Wan, L. Qiao, and J. Chen, "Formal verification of lunar rover control software using uppaal," in *FM 2014: Formal Methods* (C. Jones, P. Pihlajasaari, and J. Sun, eds.), pp. 718–732, Springer International Publishing, 2014.

[70] S. Tripakis and C. Courcoubetis, "Extending promela and spin for real time," in *Tools and Algorithms for the Construction and Analysis of Systems* (T. Margaria and B. Steffen, eds.), pp. 329–348, Springer Berlin Heidelberg, 1996.

[71] Y. Sun, B. McMillin, X. Liu, and D. Cape, "Verifying noninterference in a cyber-physical system the advanced electric power grid," in *Seventh International Conference on Quality Software (QSIC 2007)*, pp. 363–369, 2007.

[72] B. De Schutter, W. Heemels, J. Lunze, and C. Prieur, "Survey of modeling, analysis and control of hybrid systems," in *Handbook of Hybrid Systems Control, Theory-Tools-Applications* (J. Lunze and F. Lamnabhi-Lagarrigue, eds.), pp. 31 – 55, Cambridge University Press, 2009.

[73] T. A. Henzinger, *The Theory of Hybrid Automata*, pp. 265–292. Springer Berlin Heidelberg, 2000.

[74] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183 – 235, 1994.

[75] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past hytech," in *Hybrid Systems: Computation and Control* (M. Morari and L. Thiele, eds.), pp. 258–273, Springer Berlin Heidelberg, 2005.

[76] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *Computer Aided Verification* (G. Gopalakrishnan and S. Qadeer, eds.), pp. 379–395, Springer Berlin Heidelberg, 2011.

[77] A. Chutinan and B. H. Krogh, "Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations," in *Hybrid Systems: Computation and Control* (F. W. Vaandrager and J. H. van Schuppen, eds.), pp. 76–90, Springer Berlin Heidelberg, 1999.

[78] E. Asarin, T. Dang, and A. Girard, "Hybridization methods for the analysis of nonlinear systems," *Acta Inf.*, vol. 43, pp. 451 – 476, Jan. 2007.

[79] S. Ratschan and Z. She, "Safety verification of hybrid systems by constraint propagation based abstraction refinement," in *Hybrid Systems: Computation and Control* (M. Morari and L. Thiele, eds.), pp. 573–589, Springer Berlin Heidelberg, 2005.

[80] L. P. Carloni, R. Passerone, A. Pinto, and A. L. Angiovanni-Vincentelli, "Languages and tools for hybrid systems design," *Found. Trends Electron. Des. Autom.*, vol. 1, pp. 1 – 193, Jan. 2006.

[81] S. Schupp, E. Ábrahám, X. Chen, I. Ben Makhlouf, G. Frehse, S. Sankaranarayanan, and S. Kowalewski, "Current challenges in the verification of hybrid systems," in *Cyber Physical Systems. Design, Modeling, and Evaluation* (M. R. Mousavi and C. Berger, eds.), pp. 8–24, Springer International Publishing, 2015.

[82] D. Harel, *Dynamic Logic*, pp. 497–604. Dordrecht: Springer Netherlands, 1984.

[83] A. Platzer, "Differential dynamic logic for hybrid systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.

[84] J. M. Davoren, "On hybrid systems and the modal $\mu$-calculus," in *Hybrid Systems V* (P. Antsaklis, M. Lemmon, W. Kohn, A. Nerode, and S. Sastry, eds.), pp. 38–69, Springer Berlin Heidelberg, 1999.

[85] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. er, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theor. Comput. Sci.*, vol. 138, pp. 3–34, Feb. 1995.

[86] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *Automated Deduction - CADE-25* (A. P. Felty and A. Middeldorp, eds.), pp. 527–538, Springer International Publishing, 2015.

[87] A. v. Lamsweerde, "Formal specification: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ICSE 00, pp. 147 – 159, Association for Computing Machinery, 2000.

[88] A. Pnueli, "The temporal logic of programs," in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS 77, (USA), p. 4657, IEEE Computer Society, 1977.

[89] D. Jackson, "Alloy: A lightweight object modelling notation," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, pp. 256 – 290, Apr. 2002.

[90] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231 – 274, 1987.

[91] R.-J. Back, "Refinement calculus, part II: Parallel and reactive programs," in *Proceedings on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, REX workshop, pp. 67–93, Springer-Verlag New York, Inc., 1990.

[92] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: An open toolset for modelling and reasoning in Event-B," *Int. J. Softw. Tools Technol. Transf.*, vol. 12, pp. 447 – 466, Nov. 2010.

[93] M. Butler and I. Maamria, "Theories of programming and formal methods," ch. Practical Theory Extension in event-B, pp. 67–81, Berlin, Heidelberg: Springer-Verlag, 2013.

[94] M. Butler, J.-R. Abrial, and R. Banach, *From Action Systems to Distributed Systems: The Refinement Approach*, ch. Modelling and Refining Hybrid Systems in Event-B and Rodin, pp. 29–42. Computer and Information Science Series, Chapman and Hall/CRC, Apr. 2016.

[95] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef, *Validated Designs For Object-Oriented Systems*. Santa Clara, CA, USA: Springer-Verlag TELOS, 2005.

[96] The DESTECTS project. Online Website: `http://www.destecs.org/`.

[97] The INTO-CPS project. Online Website: `https://projects.au.dk/into-cps/`.

[98] A. Platzer and J.-D. Quesel, "European Train Control System: A case study in formal verification," in *Formal Methods and Software Engineering* (K. Breitman and A. Cavalcanti, eds.), pp. 246–265, Springer Berlin Heidelberg, 2009.

[99] S. M. Loos, D. W. Renshaw, and A. Platzer, "Formal verification of distributed aircraft controllers," in *Hybrid Systems: Computation and Control (part of CPS Week 2013), HSCC'13, Philadelphia, PA, USA, April 8-13, 2013* (C. Belta and F. Ivancic, eds.), pp. 125–130, ACM, 2013.

[100] S. M. Loos, A. Platzer, and L. Nistor, "Adaptive cruise control: Hybrid, distributed, and now formally verified," in *FM 2011: Formal Methods* (M. Butler and W. Schulte, eds.), pp. 42–56, Springer Berlin Heidelberg, 2011.

[101] A. Platzer, "Stochastic differential dynamic logic for stochastic hybrid programs," in *CADE* (N. Bjørner and V. Sofronie-Stokkermans, eds.), vol. 6803 of *LNCS*, pp. 446–460, Springer, 2011.

[102] A. Platzer, "A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems," *Logical Methods in Computer Science*, vol. 8, no. 4, pp. 1–44, 2012.

[103] A. Mashkoor, F. Kossak, and A. Egyed, "Evaluating the suitability of state-based formal methods for industrial deployment," *Software: Practice and Experience*, vol. 48, no. 12, pp. 2350–2379, 2018.

[104] D. Cansell and D. Méry, "Formal and incremental construction of distributed algorithms: On the distributed reference counting algorithm," *Theor. Comput. Sci.*, vol. 364, pp. 318–337, Nov. 2006.

[105] T. S. Hoang, H. Kuruma, D. Basin, and J.-R. Abrial, "Developing topology discovery in event-b," *Science of Computer Programming*, vol. 74, no. 11, pp. 879 – 899, 2009.

[106] A. Iliasov, L. Laibinis, E. Troubitsyna, and A. Romanovsky, "Formal derivation of a distributed program in event b," in *Formal Methods and Software Engineering* (S. Qin and Z. Qiu, eds.), pp. 420 – 436, Springer Berlin Heidelberg, 2011.

[107] G. Dupont, Y. Aït-Ameur, M. Pantel, and N. K. Singh, "Hybrid systems and Event-B: A formal approach to signalised left-turn assist," in *New Trends in Model and Data Engineering*, pp. 153–158, Springer International Publishing, 2018.

[108] M. Butler, J.-R. Abrial, and R. Banach, "Modelling and refining hybrid systems in event-b and rodin," in *From Action System to Distributed Systems: The Refinement Approach* (L. Petre and E. Sekerinski, eds.), Taylor & Francis, April 2016.

[109] M. Butler and I. Maamria, *Practical Theory Extension in Event-B*, pp. 67–81. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[110] D. Déharbe, P. Fontaine, Y. Guyot, and L. Voisin, "Integrating SMT Solvers in Rodin," *Sci. Comput. Program.*, vol. 94, pp. 130–143, Oct. 2014.

[111] A. Iliasov, P. Stankaitis, D. Adjepon-Yamoah, and A. Romanovsky, "Rodin platform Why3 plug-in," in *Abstract State Machines, Alloy, B, TLA, VDM, and Z* (M. Butler, K.-D. Schewe, A. Mashkoor, and M. Biro, eds.), pp. 275–281, Springer International Publishing, 2016.

[112] M. Leuschel and M. Butler, "Prob: A model checker for b," in *FME 2003: Formal Methods* (K. Araki, S. Gnesi, and D. Mandrioli, eds.), pp. 855–874, Springer Berlin Heidelberg, 2003.

[113] V. Savicks, *Integrating Formal Verification and Simulation of Hybrid Systems*. PhD thesis, University of Southampton, May 2016.

[114] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "Prism: A tool for automatic verification of probabilistic systems," in *Tools and Algorithms for the Construction and Analysis of Systems* (H. Hermanns and J. Palsberg, eds.), pp. 441–444, Springer Berlin Heidelberg, 2006.

[115] M. Jastram, "ProR, an open source platform for requirements engineering based on RIF," in *Systems Eng. Infrastructure Conference*, 2010.

[116] F. R. Golra, F. Dagnat, J. Souquières, I. Sayar, and S. Guerin, "Bridging the gap between informal requirements and formal specifications using model federation," in *Software Engineering and Formal Methods* (E. B. Johnsen and I. Schaefer, eds.), pp. 54–69, Springer International Publishing, 2018.

[117] M. Jastram, S. Hallerstede, M. Leuschel, and A. G. Russo, "An approach of requirements tracing in formal refinement," in *Verified Software: Theories, Tools, Experiments* (G. T. Leavens, P. O'Hearn, and S. K. Rajamani, eds.), pp. 97–111, Springer Berlin Heidelberg, 2010.

[118] B. P. Rochard and F. Schmid, "A review of methods to measure and calculate train resistances," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 214, no. 4, pp. 185–199, 2000.

[119] C. Bogdiukiewicz, M. Butler, T. S. Hoang, M. Paxton, J. Snook, X. Waldron, and T. Wilkinson, “Formal development of policing functions for intelligent systems,” in *IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 194 – 204, Oct 2017.

[120] A. E. Haxthausen and J. Peleska, “Formal development and verification of a distributed railway control system,” *IEEE Transactions on Software Engineering*, vol. 26, pp. 687–701, Aug 2000.

[121] F. Whitwam and A. Kanner, “Control of automatic guided vehicles without wayside interlocking. Patent US 20120323411 A1,” 2012.

[122] K. P. Eswaran, J. Gray, R. A. Lorie, and I. L. Traiger, “The notions of consistency and predicate locks in a database system,” *Commun. ACM*, vol. 19, no. 11, pp. 624–633, 1976.

[123] P. A. Bernstein, D. W. Shipman, and J. B. Rothnie, Jr., “Concurrency control in a system for distributed databases (SDD-1),” *ACM Trans. Database Syst.*, vol. 5, pp. 18–51, Mar. 1980.

[124] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 1992.

[125] A. Fantechi, A. E. Haxthausen, and M. B. R. Nielsen, “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.

[126] P. Stankaitis, A. Iliasov, Y. Ait-Ameur, T. Kobayashi, F. Ishikawa, and A. Romanovsky, “A refinement based method for developing distributed protocols,” in *IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, pp. 90–97, 2019.

[127] G. Dupont, Y. Aït-Ameur, M. Pantel, and N. K. Singh, “Formally verified architecture patterns ofhybrid systems using proof and refinement with event-b,” in *Rigorous State-Based Methods* (A. Raschke, D. Méry, and F. Houdek, eds.), pp. 169–185, Springer International Publishing, 2020.

[128] A. Iliasov, D. Taylor, L. Laibinis, and A. Romanovsky, “Formal verification of signalling programs with SafeCap,” in *Computer Safety, Reliability, and Security*, pp. 91–106, Springer Berlin Heidelberg, 2018.

[129] A. Iliasov, I. Lopatkin, and A. Romanovsky, “The SafeCap platform for modelling railway safety and capacity,” in *Computer Safety, Reliability, and Security* (F. Bitsch, J. Guiochet, and M. Kaâniche, eds.), pp. 130–137, Springer Berlin Heidelberg, 2013.

[130] J.-C. Filliâtre and A. Paskevich, “Why3 – where programs meet provers,” in *Programming Languages and Systems* (M. Felleisen and P. Gardner, eds.), pp. 125 – 128, Springer Berlin Heidelberg, 2013.