Towards a modular architecture of formal modelling : system/sub-systems decomposition in event-B

Kenza Kraibi

CENTRALE LILLE

THÈSE

Présentée en vue d'obtenir le grade de

DOCTEUR

En

Spécialité : Automatique, Génie Informatique, Traitement du Signal et Images

Par

Kenza KRAIBI

DOCTORAT DELIVRÉ PAR CENTRALE LILLE

Titre de la Thèse

Vers une Architecture Modulaire de Modélisation Formelle : Décomposition Système/Sous-systèmes en B Événementiel

Soutenue le 21 janvier 2021 devant le jury d'examen :

Présidente	Laurence DUCHIEN	Professeure à l'Université de Lille
Rapporteurs	Michael LEUSCHEL	Professeur à Heinrich-Heine-Universität Düsseldorf
	Yamine AIT-AMEUR	Professeur à INPT-ENSEEIHT/IRIT, Toulouse
Examinatrice	Régine LALEAU	Professeure à l'Université Paris-Est Créteil
Invité	Dorian PETIT	Maitre de Conférence à l'Université Polytechnique Hauts-de-France, Valenciennes
Directeur de thèse	Simon COLLART-DUTILLEUL	Directeur de Recherche à l'Université Gustave Eiffel, Lille
Co-Encadrants	Rahma BEN AYED	Ingénieure de Recherche à l'IRT Railenium, Lille
	Philippe BON	Chargé de Recherche à l'Université Gustave Eiffel, Lille
	Hector RUIZ BARRADAS	Ingénieur Chercheur à Clearsy, Aix-en-Provence

Thèse préparée dans le Laboratoire d'Évaluation des Systèmes de Transports Automatisés et de leur Sécurité COSYS/ESTAS
Université Gustave Eiffel, Villeneuve d'Ascq

École Doctorale SPI 72

CENTRALE LILLE

THESIS

Presented in order to obtain the grade of

DOCTOR

In

Speciality: Automation, Computer Science, Signal and Image Processing

By

Kenza KRAIBI

DOCTORATE DELIVERED BY CENTRALE LILLE

Thesis Title

Towards a Modular Architecture of Formal Modelling: System/Sub-systems Decomposition in Event-B

Defended on January 21st, 2021 in front of the examination jury:

President	Laurence DUCHIEN	Professor at Université de Lille
Referees	Michael LEUSCHEL	Professor at Heinrich-Heine-Universität Düsseldorf
	Yamine AIT-AMEUR	Professor at INPT-ENSEEIHT/IRIT, Toulouse
Examiner	Régine LALEAU	Professor at Université Paris-Est Créteil
Guest	Dorian PETIT	Assistant Professor at Université Polytechnique Hauts-de-France, Valenciennes
Supervisor	Simon COLLART-DUTILLEUL	Research Director at Université Gustave Eiffel, Lille
Co-Supervisors	Rahma BEN AYED	Research engineer at IRT Railenium, Lille
	Philippe BON	Researcher at Université Gustave Eiffel, Lille
	Hector RUIZ BARRADAS	Engineer researcher at Clearsy, Aix-en-Provence

Thesis prepared in Laboratory of Évaluation des Systèmes de Transports Automatisés et de leur Sécurité COSYS/ESTAS
Université Gustave Eiffel, Villeneuve d'Ascq

École Doctorale SPI 72

ACKNOWLEDGEMENT

This thesis took place, in the context of PRESCOM, a project of the IRT Railenium (Institut de Recherche Technologique Railenium) in partnership with Clearsy, at the ESTAS Laboratory of the Université Gustave Eiffel. I would first like to thank Railenium for giving me this opportunity to have this experience.

It is with all my deep gratitude that I express my sincere thanks to my thesis supervisor Simon COLLART-DUTILLEUL for supervising this thesis work. Without the advice he gave me over these three years and his words of encouragement, this work could not have been completed.

I warmly thank my co-supervisor Rahma BEN AYED, who followed by close this work, for her availability and patience. Her judicious observations and her comments were too important for the conduct of this work. Thank you. I also express my deep gratitude and thanks to my co-supervisors Philippe BON and Dorian PETIT for supervising this thesis work, for their support, advice and participation in the completion of this thesis work.

I must express my gratitude to Joris RHEM and Hector RUIZ BARRADAS, Research Engineers at Clearsy, for their help and their technical and scientific contributions in this thesis. Their knowledge and industrial expertise were significant and relevant to achieve this work.

My sincere thanks also go to Mr Michael LEUSCHEL, Professor at Heinrich-Heine-Universität Düsseldorf and Mr Yamine AIT-AMEUR, Professor at INPT-ENSEEIHT/IRIT, Toulouse, for accepting to examine this work as referees.
I also want to express my gratitude to Mrs Laurence DUCHIEN, Professor at University of Lille, and Mrs Régine LALEAU, Professor at Université Paris-Est Créteil, for accepting to be part of the jury as examiners.

My thanks also go to all the staff of the IRT Railenium and the Gustave Eiffel University of Lille for the kindness and friendliness they showed and which made my very pleasant stay among them. I would like to especially thank Sébastien LEFEBVRE, the project manager, for his availability and patience as well as for the time and effort devoted to the smooth running of this thesis.

My deep and sincere gratitude to my family for their continuous and unparalleled love, help and support. Thank you.
All this work would never have been possible without the unconditional support of my parents

Naïma and El Kattani, who have always believed in me. You were always there for me. Thanks to my dear siblings Amine, Diae and Rayahine for their advice and encouragement in the hard times.

I must express my gratitude to Soufiane, my husband, for his continued support and encouragement. I was continually amazed by his willingness to understand and read countless pages of meaningless mathematics. This stage in my life would not have been the same without the presence of you dear husband. I thank you with all my heart for being patient and smiling, to support and encourage me to go further and overcome the difficulties. On this occasion, I thank my family-in-law for their support and encouragement.

I would like to thank my friends Noha, Kaoutar, Rahma and Grecia for their support, love and for the wonderful times we shared. You were always there with a word of encouragement and listening ear.

Lastly, I would like to thank all those who have helped me from near and far to achieve this thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

Atelier B Industrial tool allowing the operational use of classical B-Method for proving software developments. www.atelierb.eu. 24, 28, 37, 38

B Classical B-Method. 23–25, 34, 36–38

B2RODIN Plugin for the transformation from B to Event-B in Rodin tool. 29

CBTC Communication-Based Train Control is a railway signaling system for the infrastructure control and traffic management using the telecommunications between the train and track equipment. 38

Clearsy Specialist company in safety critical systems. Among its works Atelier B tool. www.clearsy.com. 24

Event-B Event-B Method. www.event-b.org. 24–26, 34, 37, 38

Rodin Open source platform for Event-B modeling and mathematical proof [Abrial et al., 2005]. 26, 28–30, 37, 74

SIL Security Integrity Level. It is a measure of operational reliability. The notion of SIL derives directly from the [IEC 61508, 2010] standard. There are 4 SIL levels: a SIL 4 system is the most safe, while SIL 1 is the least. 23

UML-B Plugin for the transformation from UML to B in Rodin tool [Snook and Butler, 2006]. 29

Z Specification language Z. 23, 24

ACRONYMS

ATP Automatic Train Protection. 90

CENELEC Comité Européen de Normalisation ÉLECtrotechnique
- European Committee for Electrotechnical Standardization. 30

ERTMS European Railway Traffic Management System. 31, 36, 38, 74, 90, 91

ISO International Organization for Standardization. 30

MÉTEOR MÉTro Est-Ouest Rapide
- Fast East-West Metro. 24, 37

NSA National Safety Agency. 31

PERF Proof Executed over a Retroengineered Formal model. 34

RATP Régie Autonome des Transports Parisiens
- Autonomous Parisian Transportation Administration. 34, 37

RSS Refinement Seen Split. 15–17, 59, 94, 114, 115, 117, 118, 175, 176

SysML System Modeling Language. 23

UML Unified Modeling Language. 22, 23

V&V Verification and Validation. 22, 24, 36, 38

VDM Vienna Development Method. 23, 24

INTRODUCTION

Critical systems are systems whose malfunction would have a significant impact on businesses, properties, safety or life of people. Systems qualified as critical can be found in military applications, energy production, health and transport systems for instance. The most critical systems are usually submitted to the certification authorities, who verify compliance with the requirements set out in the standard. Safety is a major issue for critical systems given the complexity and serious consequences that may arise such as design errors. In order to limit these errors, the architectures of these safety-critical systems are subjected to a development process using techniques for specification as well as verification and validation (V&V). A specification is made on a program or a system. As a way of increasing confidence in such systems, formal methods are becoming more acceptable in industrial circles [Bowen and Stavridou, 1993]. The use of formal methods is recommended by standards such as [CENELEC EN50128, 2011] in the railway field. Besides, the *European Space Agency* (ESA) has issued guidelines for software engineering standards [ESA, 1991]. This suggests that formal notations such as Z [Brien et al., 1992], VDM [Andrews, 1992], B [Abrial, 1996] or Event-B [Abrial, 2010] should be used for specifying software requirements in safety-critical systems.

Indeed, **formal methods** are techniques that allow rigorous reasoning, using mathematical logic, on computer programs or electronic equipment, in order to demonstrate their validity with respect to a certain specification. These methods make it possible to obtain a very strong insurance of the bugs absence in the system, i.e. acquire high insurance evaluation levels. Besides, these methods are based on the programs semantics. However, they are generally costly in resources (human, material and time) and currently reserved for the most critical systems. Their improvement and the widening of their practical fields of application are the motivation of many scientific research in computer science. Formal methods take their interest when the evidence itself should be formally guaranteed correct. We can distinguish two main categories of tools allowing the V&V on formal models: model checking and theorem proof. Model checking consists in checking properties by an exhaustive and clever enumeration (according to some defined algorithms) of the reachable states. Theorem proof consists in proving the properties of the system, given a specification, through a set of axioms and a set of mathematical rules. There are possible mixtures between these methods. For example: - a proof assistant could be sufficiently automated to automatically prove most of

the utility lemmas of a program proof; - a model-checker can be applied to a model built using an automatic theorem prover; - a preliminary abstract interpretation may limit the number of cases to be demonstrated in a proof of theorems, etc.

Actually, formal methods can be applied at different stages of the system development process (software, electronics, mixed), from specification to final realisation. We distinguish two categories of formal methods, those destined for programs and those made for systems. The first category defines the model-oriented specification methods such as VDM [Andrews, 1992], Z [Brien et al., 1992] and B [Abrial, 1996]. The second category concerns the analysis and model-oriented formal methods like Alloy [Jackson, 2006] and Event-B [Abrial, 2010]. Event-B is considered as one of the foremost analysis and model-oriented formal methods [Bjrnerϕ and Havelund, 2014].

Beside all the advantages in formal practices, there are communication difficulties between several engineers. Each engineer manages a separate component and then the validation of the whole system is done manually. Besides, there is a large number of proof theorems. Moreover, modelling several independent bricks, of the same need without overall system reasoning, is difficult. In industry, the growing complexity of systems, multidisciplinary or even interdisciplinary, leads to technological failures and to time and cost overruns. In addition, communication difficulties can be faced because of the lack of global vision in the engineering and management, defective technical interfaces, difficulties in bringing together professions, organisations and few multi-disciplinary specialists.

Nowadays, face to these issues, it becomes necessary to have a set of activities allowing the design and the development of a system. This necessity results in the apparition of a new domain called **System Engineering (SE)** [ISO 15288, 2002]. It is a field of engineering that focuses on how to manage, design and integrate complex systems over their life cycles. It is a way of thinking and of understanding business through a structured approach to move from the need to the solution. One of the disciplines of system engineering is requirement engineering. Requirement engineering is the expression of the conditions or the functionalities that a system or software must meet [IEEE 729, 1983].

Indeed, industrial practices as well as dependability standards define systems or software development processes that generally begin with the analysis of the overall system. This analysis is based on the analysis of system requirements and the important properties expressed at a global level such as safety properties. In general, independently of the formal and the industrial context, the system/sub-system reasoning is approached mainly by the system engineering paradigm. A transport system is made up of many sub-systems that interact together for a common and coherent goal of providing passenger or goods transport. System engineering is an interdisciplinary and comprehensive approach. It studies the system as a whole, in addition to the development of the various subsystems. By using a structured method, it manages the complexity of the whole and reduces the risks when integrating several subsystems. It also helps plan and monitor developments throughout the life cycle through milestones, reviews, appraisals and integration points. The overall approach to system engineering is summarised in the V-diagram of the life cycle, where activities at the “system” level and at the “subsystem” level can be distinguished. This thesis is interested in the analysis of safety-critical systems using the Event-B method for the verification of railway

systems specifications.

Scientific and industrial contexts

In this thesis, the work is based on the railway area and particularly on the railway signalling systems. **Railway signalling** is an information system intended to inform the driver of a railway traffic. It gives him the information that is necessary for him to regulate the progress of his convoy and to drive in complete safety. This information is given in the form of codes produced by signals of various shapes, combinations, or colours. The information given by this means may relate to a speed limit to be observed, a stop not provided for in the course to be performed, information on a geographical direction which the convoy is going to take, prescriptions concerning electric traction, etc. Signalling is one of the basic elements of railway safety. Since the beginnings of the railway, rail signalling has generally been specific to the network of each rail company. The harmonisation of the different signals is an important issue for the interoperability of rail networks in Europe, and in North America where the presence of hundreds of private companies very early on imposed a major standardisation effort. The sectioning is based on a division of the line into sections. These sections, also called blocks, on a line between two stations are an integral part of the system. Railway safety is a set of human and technical resources that make it possible to avoid rail accidents, or to reduce the consequences of such accidents.

In this work, we start from the stage of expressing the requirements to produce formal models. These models need to be split and need the management of the reuse of bricks imported or produced by other partners. There is a need for decomposition mechanisms. Conventionally, a traditional document of the functional specification type consists of a list of detailed, documented and justified requirements concerning the different functional entities of the system. If this list is formalised in a mathematical language, it will be possible to verify, in a systematic and instrumented manner (i.e. using software) the consistency of these requirements with each other. As a reminder, the methodology which makes it possible to design a formal model which makes sense, is called a formal method: B-Method is one of these methods and it makes it possible to generate B models (this will be more widely presented later in this document).

Mainly in the railway sector, B is arguably among the formal methods of greatest industrial impact. It seems like a Domain Specific Language (DSL) for the railways [Butler et al., 2020]. However, our work is centred around the system modelling and behaviour analysis, so we use the **Event-B method** as the extension of B-Method for system analysis. The use of Event-B becomes a necessity in order to analyse the behaviour of the railway systems.

Problematic and Motivation of the Thesis

In the context of this thesis, our work concerns the modular architecture in Event-B. In practice, each engineer works on a separated brick of the system. These bricks are also called sub-systems, sub-components or sub-machines. This method of work requires the study of the partitioning of global functions on these sub-systems and communications between them. However, the interactions between the subsystems lead to problems whose formulation is delicate. These interactions can be critical locally and/or in their entirety. Consequently, we plan to study methodologies in-

cluding a modelling process based on the mechanisms of decomposition from a system to multiple sub-systems.

In the literature, several approaches are proposed to address this issue. For instance, generic instantiation [Abrial and Hallerstede, 2007], modularisation [Hoang et al., 2011], fragmentation and distribution [Siala et al., 2016] can be found. In addition, there are the most used methods in the litterature and industry: the shared variable decomposition [Abrial and Hallerstede, 2007], A-style, an the shared event decomposition [Butler, 2009a], B-style. In this work, we focus on the study and analysis of those two latter methods.

Our goal is to analyse these methods and find why there still a need in the industry. So, our motivation is based on these challenges:

– *Modularity*: each system can be splitted into several sub-systems. This must be done taking into account the system complexity. In other words, it must be possible to manage complex and huge size models after several steps of refinement. In fact, after each step of refinement new variables can be defined, new events can be added and the invariants may be more complex.

  Indeed, A-style is based on decomposing a system by functionality, like decomposing parallel programs [Hoang and Abrial, 2010]. For B-style, is based on decomposing the behaviour of a system. However, the industrial goal is to reason by sub-systems. In this case, a system can be decomposed by functionality, by behaviour or both.

– *Semantic Coherence*: In both of the cited methods, there is no link between the initial machine and sub-machines. In addition, the sub-machines are not either linked to each other. So, each sub-machine is enriched and refined separately.

Contributions

The study of decomposition in Event-B is hardly discussed in the literature. Most research in this context has focused on the partition of the events or the variables of a model. Our study on the problematic of this thesis allowed us to choose on which of these approaches can be candidate for solution of this issue. As a consequence, we choose the decomposition by shared variables and the decomposition by shared events. Following the results of the performed analysis on these works, an approach of decomposition is proposed regarding the industrial need. This is performed following these steps:

1. Study and analysis of the existing approaches of decomposition in Event-B. In our work, we focus on A-style and B-style because the other approaches are combining other languages with the Event-B language. However, our goal is to enrich the Event-B language. So, A-style and B-style are the ones are using only the Event-B language. This analysis leads to the identification of some limitations regarding an industrial need.

2. Proposition of a new decomposition method for partitioning systems into sub-systems: the *Refinement Seen Split* (*RSS*). This approach allows the decomposition of a system into multiple sub-systems and does not depend on any Event-B tool. In addition, it defines a new

link REFSEES, that gives the possibility to connect between the sub-components and get a visibility of each other.

3. Demonstration of the correctness of the proposed solution. In other words, we prove that the set of all the resulting sub-machines constitutes a refinement of the splitted one.

4. Proposition of new additional needed proof obligation rules. The goal of these proof obligation rules is to complete the correctness of the approach. These proof obligation rules are to be integrated in the Atelier B and/or Rodin tool independently of any platform.

5. Illustration of the *Refinement Seen Split* (*RSS*) method by its application on a concrete signalling railway system. This case study is validated by the domain experts.

Consequently, to cater for the above challenges, the main contributions of this thesis are summarised through the following points:

– *Modularity*: proposition of a new approach of system/sub-systems reasoning. We propose a new method of decomposition, called *Refinement Seen Split* (*RSS*), that allows to obtain modular systems. Furthermore, we define a partitioning strategy to follow as well as the rules to be respected.

– *Semantic Coherence* : a new clause is defined, called REFSEES. It is a semantic link that allows to keep the global semantic coherence of the system.

– *Scalability*: the new proposed notions are independent of the tools. They are mainly based on the Event-B language. So, it can be implemented in different tools.

After presenting the context of this thesis, problematic, motivation and contributions, we detail now the organisation of this manuscript.

Organisation of the Manuscript

This manuscript is structured in two parts, each containing two chapters. The first part is devoted to the presentation of the scientific context as well as the industrial context, particularly the railway area. Furthermore, it contains the presentation of the state of the art related to Event-B and the decomposition mechanism of this method. The first part is illustrated by chapters 1 and 2.

Chapter 1 provides the scientific and the industrial contexts of our work. In a first step, an overview of the different types of specification methods is presented, and formal methods, one of the main used methods for critical systems, are introduced. Then, we focus on the adopted formal method in this thesis, namely Event-B, which is the basis of our approach of modelling and validating railway systems. In addition, we present the verification and validation phases as well as their different techniques. In a second step, we present how formal modelling had been used in the railway area through the last years. After that, we explain the interest of this type of modelling for the critical systems industry and in particular for the safety railway systems. Finally, the chapter ends with the presentation of the PRESCOM project, its problematic and the motivation behind it.

Chapter 2 is devoted to the Event-B notations. First of all, we focus on the presentation of the structure of an Event-B model, the syntactic definitions as well as the semantic definitions through the presentation of proof obligation rules. Once these basic notions are defined, we move on to one of the main mechanisms of Event-B method: refinement. An overview on the different types of refinement in Event-B is given. Then, an explanation is given on how a refinement can be correct regarding the refined machine. Additionally, we give a presentation of the decomposition mechanism, and a citation of the existing works in the literature which are related to the modular architecture in Event-B.

The second part is dedicated to the analysis of the existing works by their application on some examples. One of these examples is a railway case study that we modelled and proved in Event-B. Moreover, this part gives the presentation of the main contributions of this thesis. The second part is illustrated by chapters 3 and 4.

Chapter 3 contains the full analysis of the most known and used approaches in Event-B for the decomposition mechanism: the decomposition by shared variables and the decomposition by shared events. In a first step, the specification of the case study, the corresponding model and the different steps of refinement are defined. Then, we proceed with the application of the decomposition methods A-style and B-style on this case study. Finally, a discussion on the results and an analysis regarding the industrial needs are presented. This chapter ends with an overview of the proposed solution, we present our approach based on decomposition into sub-components used in the PRESCOM project.

Chapter 4 is devoted to the presentation of the decomposition approach *Refinement Seen Split* (*RSS*) for modelling and validation of modular systems. We start with the definition of this approach as well as its new proposed syntax. Thereafter, the strategy for this approach use and the rules that must be respected are detailed. Among the new aspects, we define the clause *REFSEES* which allows the visibility between the sub-components after the decomposition. As one of the main mechanisms of modelling in Event-B, the refinement link should be preserved between the decomposed machine and the resulting sub-machines. So, a demonstration is made: the merge of these resulting sub-machines constitutes a refinement of the initial machine even after several steps of refinement of each sub-machine. Moreover, new proof obligation rules are introduced. These rules are necessary for the process of decomposition. In addition, we illustrate this proposition by its application on the same railway case study presented previously. This is done in order to show how our contribution solves some of the industrial issues concerning the systems modularity.

This manuscript ends with a general conclusion by giving an overview of our contributions as well as a set of short-term and long-term perspectives.

Publications

International Conferences

- Kraibi, K., Ayed, R. B., Rehm, J., Dutilleul, S. C., Bon, P., and Petit, D. "Event-B Decomposition Analysis for Systems Behavior Modeling". *In International Conference on Software Technologies (ICSOFT), short paper*, pages 278–286. 2019, July. Prague, Czech Republic.
- Kraibi, K., Ayed, R. B., Rehm, J., Collart-Dutilleul, S., Bon, P., and Petit, D. "Towards a Method for the Decomposition by Refinement in Event-B". *In International Symposium on Formal Methods (Refine@FM)* (pp. 358-370). Springer, Cham. 2019, October. Porto, Portugal.
- Kraibi, K. "Event-B: From Systems to Sub-systems Modeling". *In International Conference on Rigorous State-Based Methods (ABZ), doctoral symposium.* (pp. 418-422). Springer, Cham. 2020, May. Ulm, Germany.

International Journals: Master's degree project

- Kraibi, K., Ayed, R. B., Collart-Dutilleul, S., Bon, P., and Petit, D. "Analysis and Formal Modeling of Systems Behavior Using UML/Event-B". *Journal of Communications (JCM)*, 14(10):pp980–986. 2019, December. Paris, France.

Part I

Related Literature and Review

CHAPTER 1

METHODS OF SPECIFICATION IN THE RAILWAY AREA

Contents

Introduction

In the industrial sectors, critical systems are considered as important such as robotics, automotive, railway, aeronautic. The design and development of these critical systems must take into account several aspects such as safety, constraints, reliability, quality of service, flexibility, maintainability and performance, for instance. The goal of this chapter is to characterise specific needs of safety critical specification in railway systems, taking into account the legislative context and the industrial practices.

In a first step, we present the existing semi-formal and formal specifications and discuss the potential contributions of these methods. Particularly, we focus on the formal modelling and its ability to provide proofs. As a formal method, Event-B is used in the railway sector. This is a sufficient reason to present in detail this method and its history. In Event-B, we principally prove the models using theorem proofs as a validation technique. But, additional techniques can be used, such as model checking, on finite system states and are able to provide proofs. In addition, the animation technique can also be used to run some scenarios of the system.

In a second step, we define the industrial context, in term of formal modelling, specifically the railway sector and we detail the need of this critical systems specification with regards to the European legislative safety standards. Then, we identify the needs and presents an analysis of the railway state of the art and practices. In addition, we present how formal methods are used in the safety of railway circulations.

Finally, we analyse a list of industrial and scientific projects on railways through the 20 last years, since the apparition of formal methods in the industry. The efficiency of the tooled framework is discussed regarding the evolution of the needs. This leads us to introduce the PRESCOM project defining the context of this thesis.

1.1 Specification Methods

1.1.1 Main Types of Specification Methods

In system engineering, the specification is the step describing what the system must do. In other words, it consists in defining the system requirements. Verifying a functional specification, which is often complex, is a difficult task. The major difficulty manifests itself especially when understanding the problem (software or system). This difficulty is reflected in particular at the stage of testing the final system if a choice that should have been settled in the analysis phase has not been well defined.

The specification generates multiple documents which express the properties of the system in a certain language of specification. These documents can be used for modelling, verification and/or validation. The goal of the specification also depends on the type of the language used to detail the specification. Three main types of specification languages exist [Ben Ayed, 2016]:

– **Informal Language,** also called human language, is expressed by a natural language. It is simple and direct for communication and exchange, being the language the most easily used and understood by humans to express the needs or the perception of a problem [Sadoun,

2014]. However, neither its syntax nor its semantics are perfectly defined. Several formulations are possible for the same idea and several meanings can result from the same idea. This can cause understanding and interpretation problems for the experts.

– **Semi-Formal Language** is expressed by using a restricted syntactic language provided with a defined semantics, such as the *Unified Modeling Language* (*UML*) [OMG, 2011]. It allows a modelling activity to be more expressive while reducing the ambiguity of natural language. The graphic notations of semi-formal languages represent a good vector of communication between the collaborators of the project. They allow a structured intuitive synthetic view of the system [Dupuy, 2000]. However, the lack of precise semantics limits the use of *V&V* techniques.

– **Formal Language** is expressed in a restricted syntactic language with a semantics which is defined on well-established mathematical concepts.

Definition 1.1.1. "*A formal specification is a collection of sort, or type definitions, function and behaviour definitions, together with axioms and proof obligations constraining the definitions.*" [Bjøner, 2019]

Formal languages have well-defined syntax and semantics, as opposed to natural languages, which can give rise to several interpretations, and to semi-formal languages, which have a precise syntax but whose semantics are not well defined. No difference in interpretation is envisaged using formal languages, as it allows problems to be highlighted from the beginning. Indeed, it allows to formally prove properties on the system from its specification. There is no need to wait for the last modelling phase (where optimisation issues may be the focus of the designers), or the implementation phase for comprehensive testing. In [Idani, 2006], the author compares semi-formal methods and formal methods as in table 1.1.

In this table, the semi-formal methods give facilities to specify a system. It is easy to use and efficient to produce results on wide types of systems. However, the syntax of the specification is less precise and generalist. In addition, it lacks the semantic reasoning. In the context of this thesis, we have to prove that railway circulations are protected against dangers to obtain a commissioning authorisation for National Safety Authority. As a consequence, the use of formal methods is detailed in this document.

1.1.2 Formal Methods

Formal methods are generally used to characterise faults, errors, inconsistencies, etc., that may be faced during the life cycle of a system. These errors can have dramatic consequences when it comes to a critical system (transportation, robotics, aviation, military, etc.). The sources of these defects can be different and appear from the first phases of the cycle such as errors and mistakes of specifications, until the last phases of realisation or production. Several faced difficulties can be the origin of these defects: choice of architectures and the used tools, inappropriate tests, transmission of information between several experts, the lack of a global view of the system operational safety, etc., [Cannon et al., 2003].

Defects have a higher cost when they are detected late in the advanced phases of the development cycle. In fact, the higher the identification rates of errors and the detection of ambiguities

	Semi-Formal Methods	**Formal Methods**
Formalism	Textual or graphic (*UML*, *SysML*, etc.)	Mathematics (Z, *VDM*, B, etc.)
Language Syntax	Precise	Precise
Language Semantic	Quite weak	Precise
Validation	Syntactic + human expertise	Proof Theorems demonstration Model checking Animation and test
Tools	Software engineering atelier (SEA)	Provers + Animators
Application Area	Expects to be generalist	Critical and safe systems
Goal	Well-structured systems	Reliable and safe systems

Table 1.1: Semi-Formal Methods *vs.* Formal Methods [Idani, 2006]

and inconsistencies from the early stages of the development cycle, the more the complexity of the system is mastered. The use of formal methods, therefore, makes it possible to prove the absence of errors where the tests will only make it possible to highlight them. Using the power of mathematics, formal methods make it possible to rigorously specify the studied system. In this context, the users can have great confidence in the design of their systems. This trust is particularly essential in the rail sector where user safety is paramount. For this reason, according to the railway standard [CENELEC EN50128, 2011], the use of formal methods is highly recommended for Security Integrity Levels: *SIL* 3 and *SIL* 4. For the specification of railway systems, the formal method B is considered as one of the strongest approaches [Fantechi et al., 2013].

In general, the use of formal methods goes through three main phases [Bjøner, 1987]:

– **Establishment of needs**: the informal requirements respond to the needs expressed in the phase of the functional analysis.

– **Model construction**: the formal model precisely captures the informal requirements of the first phase. This leads us to answer the following question: "Have we well modelled the need?".

– **Model verification**: a model correctly maintains the invariants (the properties of the system) or refines another more abstract model. This leads us to two important questions: "Have we asked the right questions (properties of the system)?" and "have we fully understood the answers to these questions (results of formal verification techniques)?".

Based on the literature, a formal modelling system is based,in general, on two important concepts:

- **Abstraction** which can be seen as a process of understanding a system (its functions and properties). The abstraction is used to have a general/global vision of the system. It focuses on the most important properties and contains the foundations of the future system in order to master the complexity of the system.
- **Refinement** which is a process of enriching a model in order to increase the specification/description of the system in terms of functions or to explain how to achieve the objective. In other words, refinement consists in delaying the processing of certain functionalities of the system to latter levels of refinement while keeping the initial properties of the system defined at the level of abstraction.

In the context of the thesis, the formal method used is *Event-B*. We detail in the following some history of the apparition and use of this method, as well as the reason behind this choice. The choice is also justified in the section dedicated to the railway projects.

1.1.3 Adopted Formal Method: Event-B

B-Method

B-Method was designed by Jean Raymond Abrial [Abrial et al., 1991, Abrial, 1996], taking inspiration from the formal methods *VDM* [Jones, 1990] and *Z* [Spivey and Abrial, 1992]. Similarly, *B* is a formal model-oriented method. It has been industrially used in the *MÉTEOR* project [Behm et al., 1999]. **Classical *B*-Method**, also called *B*-Logiciel (***B* Software**) [Pouzancre and Servat, 2005, Patin, 2006], is a formal method for the specification and the *V&V* of critical systems. It is based on the use of set theory and first-order logic as the mathematical foundations of modelling. It is also based on the use of mathematical proof to check the correctness and consistency of the system regarding its specification, as well as to check the correctness and consistency between the different levels of refinement. This method ensures correct operation of the specified software and achieves a conform implementation of the latter with its specification. *Atelier B*[1] [Atelier B, 2018] is the recognised industrial tool which implements method B. It is developed by *Clearsy*[2] [ClearSy, 2020].

Event-B

The classical B-Method allows formal development from specifications. It is based on the refinement mechanism from specification to code. A few years after the appearance of the classical B-Method, an awareness raised in the importance of reasoning about the behaviour of the system, not just about the software. For example, the railway signalling system does not only represent the software part, but it also groups together a set of several software and hardware subsystems. An analysis of such systems is carried out in order to study a system or its components in order to identify its objectives. It is a problem-solving technique that improves the system and ensures that all its components work effectively to achieve their goal.

[1] Atelier B: www.atelierb.eu/
[2] Clearsy Systems Engineering: www.clearsy.com/

The choice between classical B-Method and *Event-B* depends on the purpose of modelling the system, i.e. an analysis of the system according to a refinement process which stops at a level of refinement considered as necessary to analyse the behaviour of the system regarding its objectives. Otherwise, an implementation of the studied system according to a refinement process leads to a model described in a language close to the code and which will be considered as the last refinement model or the implementation model.

Event-B is an extension of B-Method which allows the specification of reactive, sequential, concurrent and/or distributed algorithms. *Event-B* is based on mathematical approaches: the theory of sets and the logic of predicates. It allows an incremental construction of the system specification. The *Event-B* model is the first concept of *Event-B* because it describes a system by a set of states, a set of actions, an initial state and a transition relationship. The *Event-B* model, as described in detail in chapter 2, is made up of a set of machines and contexts.

B vs. Event-B

Initially, B-Method is limited to the development of software systems, but a need for the incorporation of the event approach has emerged, linked to the systematic derivation of reactive distributed systems. Models based on events have been found useful in the needs analysis, the modelling of distributed systems, and the design of distributed and sequential programming algorithms. The comparative study is based on the following points [Boulanger, 2012]:

- **The structure of models**: The structure of models in *Event-B* is different from that of B-Method. Indeed, in *Event-B*, as we have already mentioned in this document, the static part of the system is defined in a context, while the dynamic part of the system is defined in a machine [Benaissa, 2010]. In B-Method, the two parts coexist in the same machine.
- **Events vs. operations**: the operations of the classical B contain preconditions which must be true when the operation is invoked. The calling operation is then responsible for ensuring that the preconditions of the called operation are satisfied before calling it. The called operation can assume that its preconditions are satisfied and that it does not need to check its preconditions. On the other hand, an event in *Event-B* has a guard instead of a precondition. A guard is associated with each event. Several event guards can be true at the same time, however, only one event can be triggered. The choice of which event is triggered is not deterministic. Indeed, the call of events does not exist in *Event-B*; it is the model that controls its behaviour by choosing in a non-deterministic way the events to trigger.
- **The refinement mechanism**: The refinement in *Event-B* is considered more general than that of B-Method [Boulanger, 2012]. In *Event-B*, we can refine existing events by strengthening their guards, as in B-Method we can refine operations. In addition, in *Event-B* we can introduce new events in order to observe concrete behaviours that did not exist in abstraction. On the other hand, in B-Method you have to go back to the abstract machine, define the entire signature of the operation with its input and result parameters and then go to the refinement step. This requires the prior analysis of all the requirements and the definition of at least the signature of all the necessary operations for this machine and for all the successive refinements of this machine.
  Table 1.2 summarises the differences between classical *B* and *Event-B*.

	Classical B (B Software)	***Event-B* (B-System)**
Structure of the models	The static part and the dynamic part coexist in the same machine.	The static part is defined in a context and the dynamic part is defined in the machine.
Events vs. Operations	The operations can be called by other external operations of another calling machine.	Events are triggered by themselves if the guards of the event are verified.
Mechanism of refinement	Ability to add new variables, new invariants but not new operations. You have to go back to the abstract machine, define the entire signature of the operation with its input and result parameters and then go to the refinement step.	Possibility of introducing new variables, new invariants as well as new events which do not exist in the abstraction.

Table 1.2: Differences between Classical B and Event-B

Event-B vs. B-System

The Event-B method is supported by the integrated development environment *Rodin* [Abrial et al., 2005] which allows the editing, the validation of Event-B models, and the generation of proof obligations and their discharge (see chapter 2).

B-System designates a variant of *Event-B* offered within the integrated development software environment *Atelier B*. The B System and Event-B languages share the same semantics but differ in their syntax. In *B-System*, one can use all of the *classical B* method syntax and clauses such as *INCLUDES* and *DEFINITIONS* clauses in addition to some of the *Event-B* syntax. However, some of the *Event-B* syntax is not defined in *B-System*:

- For the clause of the system name (SYSTEM M), the term machine is used like in *classical B* (Machine M).

- There is no *CONTEXT* notion in *B-System*, both of the static part and the dynamic part can be present in the same machine. Otherwise, they can be separated into two machines. The SYSTEM M and the CONTEXT C in *Event-B* are defined by MACHINE M and MACHINE C respectively in *B-System*.

- Since the use of *Classical-B* syntax is allowed in *B-System*, the clause *SEES* can be used to see other machines defining the static part, dynamic part or both. Contrary to *Event-B* where the clause *SEES* only defines contexts.

- The clause *EXTENDS* is not defined in *B-System*. So, in order to extend a static part, the clause *SEES* can be used.

Verification and Validation (V&V)

Checking and validating critical systems requires reasoning on the scenarios and the related requirements for using the system. These scenarios must be modelled in a structured and intuitive way in order to offer a representation that facilitates understanding and communication. To give an initial and approximate illustration of the reasoning used by this method, either a proof or a model checking is necessary on the possible states of the system. For example, the initialisation must establishes the target properties and all subsequent developments retain these properties. This reasoning can be used at any level of detail and repeatedly as presented in figure 1.1:

- From high-level: which is used to demonstrate that the defined functional needs are coherent and complete and guarantee the desired fundamental properties.
- Until low-level: which is used to demonstrate that the final architecture is consistent and complete compared to the functional needs defined at the start.

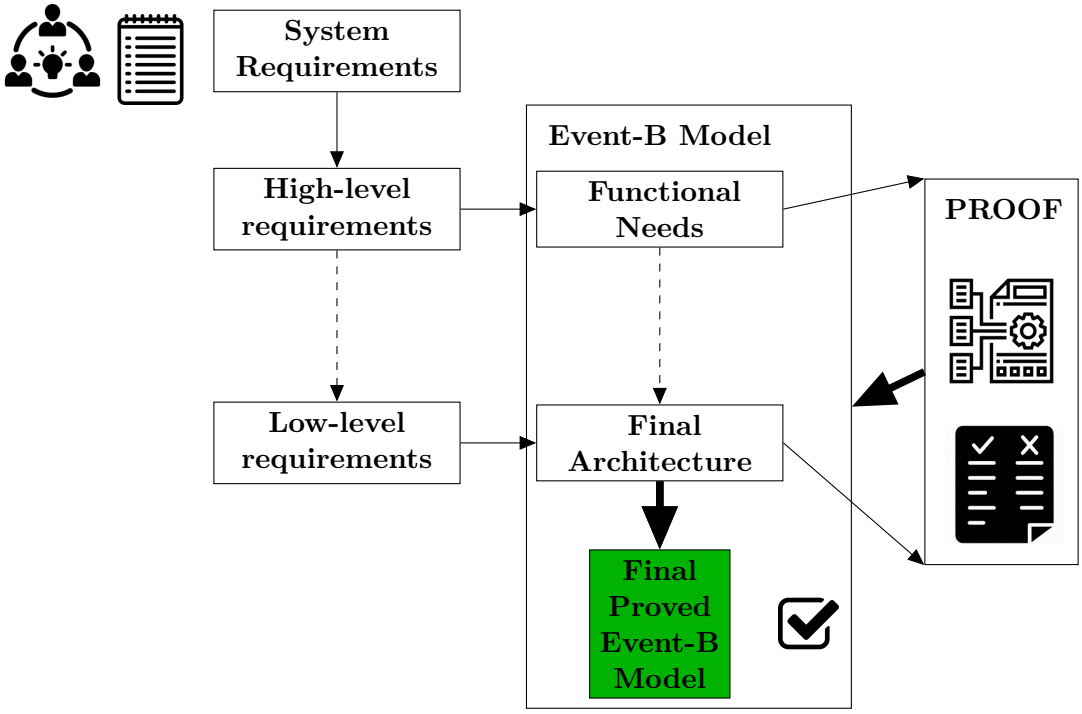


Figure 1.1: Formal Validation and Verification in Event-B

(i) Proof Technique

The Event-B method defines a mathematical language allowing the specification of a system, as well as the description of the target properties. The theory associated with this method makes it possible to know how to prove that this description of the system does indeed guarantee these target properties. In other words, it allows to demonstrate that an Event-B model is correct. This is done by defining lemmas which must be proved, called *Proof Obligations*

(see section 2.1.2).

The purpose of the formal proof technique is the system verification by performing lexical analysis, syntactic analysis and type verification. The proof obligations relating to each machine mainly concern the correction of events:

- The initialisation must establish the machine invariant: the machine invariant must be true after applying the initialisation substitution.
- Events must preserve the invariant: the machine invariant must be true after applying the event substitution, assuming that it was true before.

Theorem proof involves a demonstration and a formal proof usually using an automatic demonstrator and a semi-automatic demonstrator, also called an interactive demonstrator. If these demonstrations succeed, then the consistency of the specifications is guaranteed with respect to the defined properties.

Indeed, the goal of the theorem prover is to construct a mathematical proof for a mathematical statement in order to demonstrate that it is true. If this statement is proven by the evidence, then the statement is true, and it is considered as a theorem. Otherwise, if evidence is not found, it cannot be concluded that the statement is false. Actually, it could be false, as it could be true, but any interaction or the used tool of proof do not succeed in finding the suitable proof to demonstrate that the statement is true. Once the theorem is demonstrated, it is applied to the entire model.

Schematically, the description of the properties and the system specification in Event-B are both introduced in the tool which controls their syntax, generates the proof obligations and launches the integrated automatic demonstrator.

Atelier B and *Rodin* have a generator of proof obligations. These proof obligations are discharged by two types of provers:

- An automatic prover to demonstrate most of the verifiable proof obligations.
- An interactive prover with a number of interactive commands which allow to discharge the verifiable proof obligations that the automatic prover has failed to demonstrate. This prover also makes it possible to identify a modelling error after having interpreted one or more attempts to an interactive proof which does not lead to a demonstration.

If the model is correct, the interactive prover is able to finalise the proof by demonstrating all the proof obligations which are not discharged by the automatic prover. The tasks that can be automated during the development of a project are the syntax checks of the components, the automatic generation of proof obligations and the automatic translation of the B implementations into the C or Ada languages.

If all the proof obligations are demonstrated, the B description of the system specification is a valid model regarding the target properties. However, this specification must be well defined beforehand in an informal or in a semi-formal language in order to specify and formally verify them to guarantee the consistency and the completeness of this specification. We may have

to use a semi-automatic demonstrator if the automatic demonstrator fails to discharge all the proof obligations. However, this phase can be costly in time and resources.

The *Rodin* platform[3] [Abrial et al., 2010] is an IDE for modelling in Event-B based on Eclipse which provides effective support for abstract machines, refinement and mathematical proof. The platform is an open source and can be extended with plugins (*UML-B*, *B2RODIN*, etc.).

(ii) Model checking

Model checking is used for the formal verification of behavioural systems modelled in relation to the expected properties. This technique is based on the construction of a model (state machine) generally finite which describes all the possible states, the initial state and the state transitions. It is established by an exhaustive enumeration of the possible (or visited) states from the initial state. Given a property of the system to be checked on a machine, the model checking technique explores the set of reachable states by this machine in order to verify that this property is indeed satisfied. Two cases arise: either the property is checked and maintained by the model, or a sequence of state transitions leading to the violation of the property is generated as a counter example as presented in figure 1.2. This shows that the property is not maintained by the model.

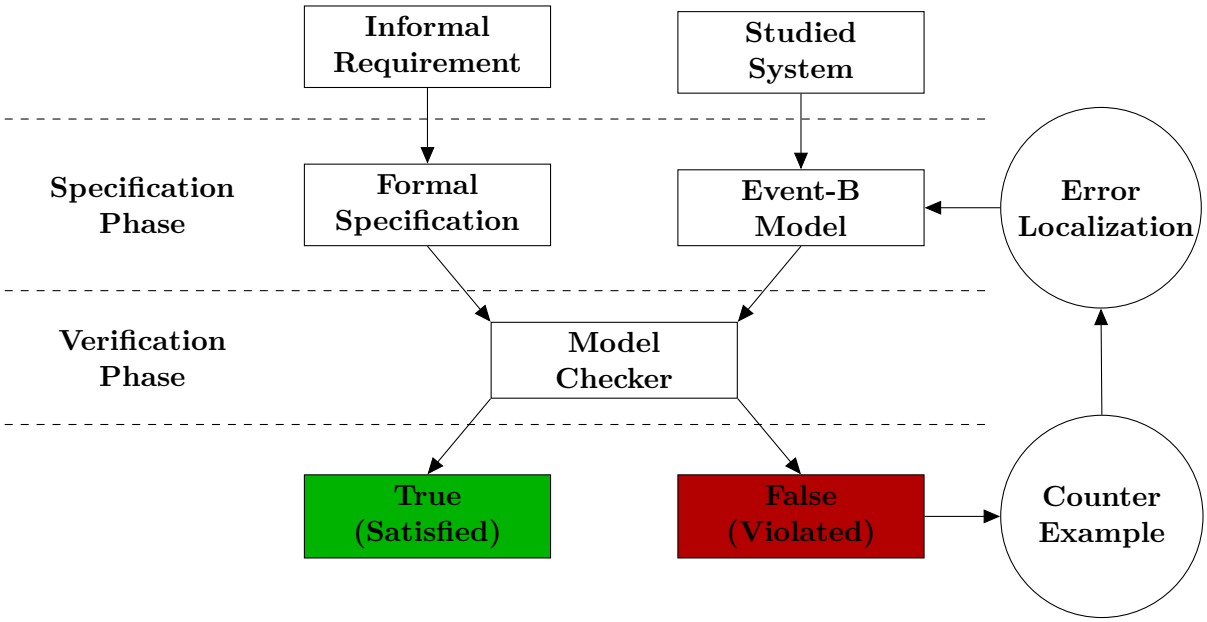


Figure 1.2: Process of Model Checking

Comparing model checking and proof, on the one hand, model checking is easier and faster than proof of theorems. On the other hand, the theorem demonstrator does not necessarily rely on finite and decidable systems, unlike the model checker. Indeed, the theorem demonstrator is applicable in certain cases: the model checking cannot be done because of the problem of the state space explosion or the studied system cannot be formalised as a finite state space model.

[3] *Rodin*: www.event-b.org/

(iii) Animation

Animation is a fast prototyping technique of validation. It allows to visualise certain formal scenarios of the system and thus to validate the dynamic behaviour of the system. Indeed, it makes it possible to demonstrate, at an early stage, the absence or an undesirable behaviour presence in the used scenarios. In absolute terms, it does not guarantee the correctness of the system, however it increases the user's confidence in their formal specifications. Animation is not an alternative to formal validation and cannot replace proof. These two techniques have an important complementary role and can be used in conjunction. Indeed, animation can be used as a validation method providing a quick view of the model execution, in addition to the proof for a deeper verification. It makes it possible to identify and locate possible problems in a model. Some animation tools such as ProB[4] support the analysis of liveliness properties and detect deadlock problems. ProB, proposed by [Leuschel and Butler, 2003], allows users a step-by-step animation of the machines in Event-B which makes it possible to see a description of the machine's current state, the history which led the user to access this current state, and a list of all triggerable events. There is also a ProB animation plug-in on *Rodin* [Butler and Hallerstede, 2007]. This Plugin can be included and performed in *Rodin*.

1.2 Formal Modelling and Verification in Railway Systems

1.2.1 Railway Systems

The railway system is a guided transport system used to transport people and/or goods. It is made up of specialised infrastructure, rolling stock and operating procedures, most often involving humans. Rail traffic management is ensured by control and command systems, whether train, tram or metro. These systems are used, for example, to control the speed, the distance to be respected between two trains or even signalling for drivers. Thus, they meet several European standards and those imposed as a worldwide standard used in the railway sector such as [ISO/TC269/SC1, 2017] for the infrastructure and [ISO/TC269/SC2, 2015] for the rolling stock. These standards are developed by the *International Organization for Standardization* (*ISO*). Their application is therefore required for all suppliers of railway control equipment.

The railway systems in France comply with several European standards from the European Committee for Electrotechnical Standardisation (*CENELEC*). Three specific standards were published in the early 2000s. The [CENELEC EN50126, 2001] standard concerns systems in their totality, the [CENELEC EN50129, 1998] standard is dedicated to electronics, and the [CENELEC EN50128, 2011] standard is dedicated to software. These European standards have established themselves as a standard used worldwide in the railway sector. Their application is therefore required for all suppliers of railway instrumentation and control-command equipment.

The *control-command* system directs the movements of rolling stock and makes it possible to manage the control and the command of trains of several lines and includes track and board automation on the trains. It is a modular, scalable and secure control, as well as a communication platform based on the open *CENELEC* standards that manages and controls the transmission of information between the various on-board subsystems (converters, doors, heating, ventilation and

[4]ProB: Animator and Model Checker https://www3.hhu.de/stups/prob/index.php/Main_Page

air conditioning, etc.), but also between the train and the track systems (rails, sensors, etc.) [Schön, 2013a].

It thus ensures efficient and reliable train operations, diagnostic-based maintenance, high-security rail operations, as well as practical and high-quality passenger services [Pawlik, 2015]. Thanks to the secure data transmission protocol and functions of the train control and management system, it is possible to set up secure train functions and protected communications between equipment and subsystems.

In Europe, railway principles and standards are used to be validated at the national level by the *National Safety Agency* (*NSA*). Historically, each country has its own requirements for managing trains on its network. As this national specific safety process was breaking the economic development, the European Union has introduced a new solution called the *European Railway Traffic Management System* (*ERTMS*)[5] creating a common and standardised management of rail traffic and signalling in Europe.

Actually, a proposition in [Blakstad, 2006] implies a human mastering all the connected knowledge and able to make synthesis and compromise. It is an evidence that a drawback of this approach is that it is difficult to apply with radically new technologies. Actually, it is not possible to find an expert of railway technology mastering all the connected knowledge, like, for instance, mastering the knowledge of railway safety, telecommunication and human factors. An alternative approach uses a set of dedicated experts. In this case, there is a problem with domain specific semantic. Furthermore, dedicated experts do not have a mental representation of the impact of their technical choices outside of their domain of knowledge. Let us consider the following definition:

Definition 1.2.1. "*A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system*" [Bézivin and Gerbé, 2001]

Building a model means representing the real world focusing on specific aspects. It is relevant because it provides an operational abstraction of a given knowledge, focusing on impacts on a given structure. Finally using a dedicated model for the formally projection of a specific knowledge regarding a given aspect is quite efficient from a conceptual point of view.

Formal methods have been widely used and implemented by manufacturers for different types of applications (automatic metros, signalling subsystems, train applications developed with Control-Build, for example) and at different levels (specification, design, code). The [CENELEC EN50128, 2011] standard dedicated to the realisation of software applications points to the interest of using formal methods.

Several driverless automatic metro projects by Siemens[6] have been developed without a formal method such as VAL[7] (Véhicule Automatique Léger-Light Automated Vehicle) of Lille, France in **1983**. This project was referred as the first fully automated driverless metro of any kind in the world [Bushell and Stonham, 1985]. Following these projects, formal methods were introduced to carry out several national and European projects.

[5]European Rail Traffic Management System (ERTMS): www.ertms.net
[6]Simens: siemens.com
[7]Ilevia: www.ilevia.fr

1.2.2 Safety of Railway Systems

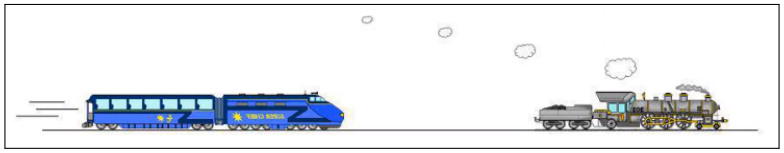
The work presented in this thesis takes place in the context of the railway industry whose safety requirements constitute a central concern in the development process. Many railway systems do not use software to implement critical safety functions, those whose failure can result in a catastrophic risk to passenger safety [Lecomte et al., 2007]. These systems must meet strong safety requirements [CENELEC EN50126, 2001] in order to avoid the five typical scenarios/accidents:

- The **Nose-to-nose Collision**: also called face-to-face, is a head-on collision between two trains running on the same track and in opposite directions (see figure 1.3a).
- The **Rear-end Collision**: unlike nose-to-nose, a train catches up with another train in front of it while going in the same direction (see figure 1.3b).
- The **Sideswipe Collision**: occurs when a train arrives on a switch already occupied by another train coming from another direction (see figure 1.3c).
- The **Train Derail**: occurs when a part of a train (car, wagon, etc.) runs off its rails. It includes leaving the track on a curve or on a switch crossed too quickly, or the circulation on a track with excess speed (see figure 1.3d).
- The **Collision with an Obstacle**: is the meeting of a train with an obstacle not strictly railway like a rock, an animal or a non-railway vehicle such as a car at a level crossing for example (see figure 1.3e).

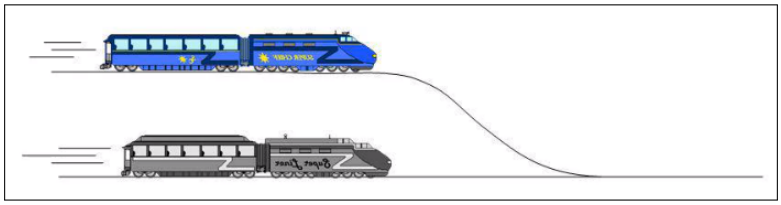
Over time and technological developments, signalling and railway automation have been proved to be major allies to improve the operation of rail networks. Beyond the security aspects, the implementation work of signalling and associated automation constitutes an essential basis in order to: fluidify and regulate circulation, improve the comfort of users and operating agents, and reduce operating and maintenance costs. Railway signalling is the management of safe train movements. It is based on dynamic behaviours to be respected by the systems. In order to achieve the spacing of traffic, the track is cut into sections called "blocks" [Kempen, 1993, Schön, 2013b]. Each block is then preceded by a signal indicating whether this block is *free* or *occupied* by another train. Consequently, it makes it possible to avoid catching up of trains on the same track, guarantee the protection of traffic in intersections, avoid derailments by speeding (in zones with limited speed or with curves, for instance), to protect a level crossing (rail-road crossings), etc.

In France, the railway sector is regulated by European and national rules. Consequently, the respect of the [CENELEC EN50126, 2001], [CENELEC EN50129, 1998] and [CENELEC EN50128, 2011] standards for the design of a railway system is mandatory. Railway signalling does not make it possible to identify system failures and to dynamically check the adequacy between the specifications and the source code. Analysis of this problem leads to consider that software failures fall into two categories: design errors, which cause the programming source code not to comply with the software specification; and compilation errors, which cause a program not to run according to its source code. Therefore, formal techniques appear to be an answer to the problem raised by the digitisation of critical security functions with regard to errors in the design of software applications. The [CENELEC EN50128, 2011] standard identifies formal methods as means to be implemented.

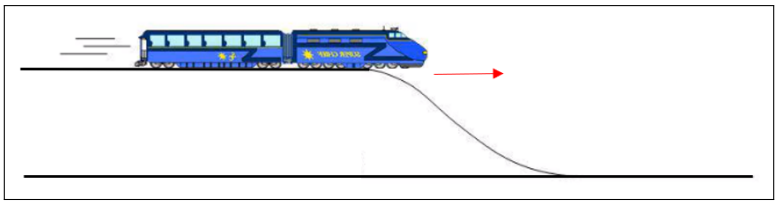
(a) Nose-to-nose Collision



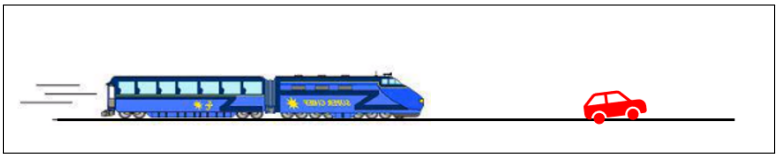
(b) Rear-end Collision



(c) Sideswipe Collision



(d) Train Derail



(e) Collision with an Obstacle

Figure 1.3: Representation of the Main Accidents of Trains Circulation

In order to extract the design errors and to model the behaviour of the railway systems so that the model obtained is safe and reliable, it is necessary to formally model these systems and reason over it. Formal modelling requires more expertise and advanced knowledge in mathematics [Kraibi et al., 2019a]. In what follows, we present in detail some famous projects that used formal methods in their design and validation process.

1.2.3 Formal Modelling in the Railway Sector

Based on a feedback of 25 years using formal methods, the Autonomous Parisian Transportation Administration *RATP* noticed that the use of the *Proof Executed over a Retroengineered Formal model* (*PERF*) approach [Bonvoisin and Benaissa, 2015] instead of validation tests, reduces the overall workload by a proportion of 25% [Benaissa et al., 2016]. *PERF* is an approach being developed by *RATP* based on two main phases The first one, uses *B* method or *Event-B* depending on the nature of the project. It is a Top-Down design process [Bonvoisin, 2016]. The second phase of *PERF* is rather a Bottom-Up approach based on model checking, induction and abstractions. This phase is applied at a software level, considering that the lower levelled product are software entities.

The SACEM project (Système d'Aide à la Conduite à l'Exploitation et à la Maintenance-Driving Assistance, Operation and Maintenance System) is a railway automation system for the RATP[8] (Régie Autonome des Transports Parisiens-Autonomous Parisian Transportation Administration) [Dollé et al., 2003]. SACEM has been put into operation in **1989**. This system allows almost optimal operations for the busiest part of the RER network (Réseau Express Régional-Regional Express Network) [Hennebert and Guiho, 1993].

In **1998**, a project was developed for the RATP by Matra Transport International (now Siemens). It concerns line 14 of the fully automatic driverless metro in Paris. Piloting this system required the use of safety software formally developed with the B method, including proof, allowed to suppress the unit tests and it gave a remarkable result [Behm et al., 1999].

The application and improvement of these formal techniques for system studies is a differentiating factor for the competitiveness of the experts.

In **2012**, the EPSF (Établissement Public de Sécurité Ferroviaire-Public Railway Safety Establishment) was very receptive to proposals of the formal methods use within the framework of the PERFECT[9] project (Performing Enhanced Rail Formal Engineering Constraints Traceability) which studies the LGV-Est (Ligne à Grande Vitesse Est européenne-East European High Speed Line) [Ben Ayed, 2016]. The main goal of this project is to develop the safety specification and verification of French railway interlocking systems in the context of national rules and the influence of implementing ERTMS[10].

laws on the original systems [Bon et al., 2013, Sun et al., 2014, Sun et al., 2015]. The study proposes a methodology for consistency assessing of the following two aspects:

– The operating rules of local signalling systems and interlocking;

[8]RATP: www.ratp.fr

[9]PERFECT: a project of the National Agency of Research (ANR)

[10]ERTMS: European Rail Traffic Management System, instructed by the European Union (EU), is the system of standards for the management of railways signaling.

– The additional safety requirements (like ERTMS).

This methodology allows addressing the safety assessment of new systems, the analysis of given scenarios and the evaluation of safety requirements of system updates. In the framework of the perfect project, modeling operating rules was presented in [Ben Ayed et al., 2014].

Following the PERFECT project results and perspectives, in **2016**, IRT Railenium[11] developed the NExTRegio project (initially ERTMS Regional) for SNCF[12] (Société Nationale des Chemins de fer Français-French National Railway Company). An original solution was proposed for the system security analysis including the operating rules. Also, some tools were proposed for the validation of certain configurations, in particular the change of a component or the change of distribution of the human/machine collaboration in order to perform a given task [Idani et al., 2019]. The NExTRegio project focuses on the French regional line. The scientific work proposed to provide tools for the system safety analysis including the operating rules. Different kinds of traffic are considered: freight, passenger and mixed traffic. The variation of the need in terms of capacity may vary from a line to another. The need integrates some regional phenomena, like pick hours and seasonal traffic. This wide diversity in terms of needs, has to be built on an existing infrastructure, ensuing from the history of the region: they may be oversized, overloaded, more or less automatized and using various technologies. One of the common motivations for changing the global technical environment for controlling the regional line is that human workers mainly remaining in the railway stations perform a lot of controls and operations. This kind of solution increases the cost of the global exploitation of the line and decreases possibilities of building a good business plan. Scientifically, the preceding proposals bring out a certain number of difficulties:

– A methodology for the design of abstract architecture has to be constructed;

– An exploitation of this abstract architecture, for example by changing or refining a component, will raise the question of compliance with the requirements materialized by system invariants.

Today, in **2020**, the Autonomous Train[13] is an ongoing project at the heart of the French railway industry research and innovation strategy. To develop this project, SNCF is partnering with major industrial players as well as the Railenium Technological Research Institute. It aims to optimize the speed of trains and therefore better harmonized traffic. This leads to an improved punctuality, a smoother traffic, a reduced energy consumption and a greater circulation capacity.

A dedicated task of this project focuses on the engineering needs by the means of formal methods.

In the scope of this project, the train stores and analyses a lot of information (including continuous position provided by a composed system mixing odometry, GNSS and accelerometer). Actually, an autonomous train has many states, which may be taken into account by the control railway center in order to manage the whole system. Moreover, in the safety analysis, the autonomous train may provide some safety critical information from its industrial vision system to the control railway center: detection of obstacle on the line, detection of obstacle on adjacent lines, detection of broken rail, detection of damaged catenary system, detection of people near the track area, etc.. All these information will trigger dedicated procedure in the control center. From a formal assessment architecture point of view, it looks non-tractable to validate a new kind of autonomous train including its

[11]Institut de Recherche Technologique Ralenium: a research institute specialized in the railway field
[12]SNCF: www.sncf.com
[13]Autonomous Train: a French project of SNCF in partnership with IRT Railenium

model into a huge model of the infrastructure. A modular process is needed, allowing to implement locally a specification, but the associated tools and methodologies have to be introduced.

1.2.4 PRESCOM Project

Modular System Design

Designing a system in a modular way by studying each of the components, aspects or points of view separately is a widespread industrial practice, especially in the design of guided transport control systems. The fact remains that a number of basic requirements are expressed across the entire system. The objective of this project is to demonstrate the system safety reasoning in a global and automated way. The target market is the railway sector in all its components which could require the development of safe rail systems and subsystems. The efforts made by the heavy rail sector, which is experiencing some delays in this process of using formal methods compared to the urban rail sector, must be continuous. This project aims to interest the French rail safety authority since the developed tools have the potential to make it possible to demonstrate the safety of the system in a comprehensive manner.

The aim of this thesis is to demonstrate that formal methods can contribute to the process of examining the various safety documents, which is a necessity for obtaining a safety certificate.

In terms of targeted applications, this PRESCOM project will:

- Improve functional pre-studies;
- Improve the quality of system studies (functional description and Top-Down design to subsystems);
- Carry out a *V&V* activity of interoperability standards for signalling (*ERTMS* type) and facilitate the activity of subsystem acceptance by the project manager;
- Improve the possibilities of reusing formal models for a new system;
- Improve the validation of the hypotheses that each subsystem must guarantee in order to ensure the safety and the functionality of the system.

Description of the Project

The rail sector relies on the safety of its systems, which must be proven before any commercial operation. The PRESCOM project has the overall objective of improving the development automation of safety systems. This is done using formal methods (*B* method here) by providing Proof of Global Safety for the Modular Design of railway systems and subsystems. This project is proposed due to the results and perspective of the NExtRegio project. Indeed, the complexity of the systems is such that the use of formal models and formal verification allows better control (precision, allocation, completeness, etc.) of the sub-systems expectations. The system study and modelling are therefore important steps for the control of any modifications or realisations.

Conventionally, a traditional document of the functional specification type is made up of a list of detailed, documented and justified requirements concerning the different functional entities of the system. If this list is formalised in a mathematical language, it will be possible to verify in

a systematic and instrumented manner (i.e. using software) the consistency of these requirements with each other. From this modelling of the system in natural language, it is possible to convert it into mathematical language in order to obtain a so-called formal model. As a reminder, the methodology that allows to design a formal model which makes sense, is called a formal method. *B*-Method is one of these methods and it makes it possible to generate B models (this will be more widely presented later in this document). In fact, these arguments are economically strong because the cost of anomaly detection during the installation phase is 25 times higher than that of the specification phase. As mentioned above, the first industrial use of formal methods was carried out on the SACEM software for the RER A of Paris [Guiho and Hennebert, 1990] followed by the Fast East-West Metro project *MÉTEOR* in 1998 for line 14 of the *RATP* [Behm et al., 1999].

Thanks to the use of formal methods, no software bug was discovered after the proof: neither during integration tests, functional or on site since the line was in operation, for 20 years. This successful deployment has opened up the use of formal methods for the development of critical software and systems.

Principal Objectives of the Project

The research work carried out as part of the PRESCOM project is the mechanisms of decomposition and partitioning of the global system and its global functions into communicating subsystems. The objective is seen according to three components: scientific, technical and industrial.

From a **scientific** point of view, the mechanisms of refinement and decomposition is studied in this project. In a refinement-oriented approach, the abstract global system is modelled according to a minimal architectural structure. Refinement and decomposition are then used to separate elements of the structure.

Several mechanisms of decomposition exist in the literature: the decomposition of models and the decomposition of atomic events.

The main idea of decomposing models is to be able to decompose a model into several sub-models which will be more easily refined separately than globally. This decomposition facilitates the proof phase on the modelled system, as well as the automation of this proof. The decomposition of atomic events, in turn, has the vocation of moving from an abstract atomic event towards sub-events of fine granularity.

In order to understand the needs and model the system requirements, the structure of a system can be seen as a set of interacting components. Two main aspects are then taken into account: synchronisation between the sub-components of the system and communication via interfaces. These aspects will consolidate the multi-component modelling architecture thanks to the decomposition mechanisms.

From a **technical** point of view, the implementation of the decomposition mechanisms in the *Atelier B* tool is carried out by taking inspiration from the plugins developed within the *Rodin* research project. *Atelier B* is at the centre of the research work for this project. It should allow the partition of models in *Event-B*. Currently, *Atelier B* supports classical *B* and *Event-B* modelling. The main features are:

– Automatic generation of proof obligations from components in *B* language;

– Evidence aid thanks to suitable proof tools: an automatic prover (allowing to automatically search for a proof for a given theorem) and an interactive prover (allowing a user to interactively build a correct proof);

– Design mechanisms: management of relationships and dependence between B components.

This latter functionality essentially includes the refinement mechanisms and the compositional mechanisms of classical B models. However, several *Event-B* decomposition mechanisms are not yet taken into account in the tool. The project proposes to implement these mechanisms and integrate them in *Atelier B*.

From the **Industrial** point of view, the industry practice creates an abstract formal model corresponding to the requirements of the global system and then decomposes this model into subsystems corresponding to an industrial architecture. Our application case is a rail signalling system: it is an essential complex system to master from a safety point of view. This system is central to the passenger safety, but it also contributes to the quality of the provided service (having an impact on the frequency of trains and the rate of the lines use). This system is made up of subsystems with specific characteristics, such as lateral signalling or other more modern signalling (*ERTMS*, Communication-Based Train Control (*CBTC*) for urban). Thus, we seek through this project for a better formalisation of railway systems and therefore a better definition of the subsystems contours and specific requirements, inducing proof obligations, associated with each of these subsystems.

1.3 Conclusion

The critical systems modelling, such as railway systems, causes difficult problems of validation, verification, safety and certification. The formal specification of these systems as their environment is essential. Creating a system description of high quality is still a challenging problem in the field of formal modelling.

This chapter aims at analysing specification needs of railway industrial systems. Characteristics of both semi-formal and formal methods were presented, but regarding the need of proofs ensuing from the legislative railway context, we focus on formal methods. Then the genesis of the Event-B is presented, and its existing ecosystem is detailed. The industrial efficiency of the considered tools is proven through the industrial history. Nevertheless, it seems that the more a train is becoming intelligent, the more a modular tooled approach is needed.

In our thesis, we are interested in the use of Event-B formal method and its application on the railway sector. Particularly, we focus on the use of modular architecture in order to partition the systems and to better control them in both of the phases: the specification phase and the *Verification and Validation* (*V&V*) phase. The next chapter details the existing potential provided by Event-B method, as well as a state of the art concerning the refinement and the modulisation concepts.

CHAPTER 2

STATE OF THE ART: MODULAR ARCHITECTURE IN EVENT-B

Contents

Introduction

The analysis and modelling activities of railway dynamic behaviours are major tasks requiring rigorous mechanisms. Based on mathematical foundations, formal methods can help to rigorously carry out these activities and reduce the ambiguity of the specification of critical systems such as railway signalling systems.

As part of the PRESCOM project, Clearsy needs an enrichment of the Event-B method [Abrial et al., 2010, Abrial, 2010] providing appropriate techniques for system modelling based on the B method [Abrial, 1996]. Event-B methods have been widely used in the railway field in research such as the *PERFECT*[1] and *NExTRegio* projects [Ben Ayed et al., 2016, Ben Ayed et al., 2014] and in industry sectors as in the *METEOR* project [Behm et al., 1999]. In the same context, CLEARSY[2] has also driven railway projects using formal proofs [Sabatier, 2016].

In fact, modelling of critical systems such as railway signalling systems can lead to complex and voluminous models. One of the Event-B techniques for this issue is refinement. Refinement consists in detailing the design to reach a concrete level by progressive steps. However, the final level of modelling is still difficult to manage. In order to reduce this complexity, refinement can be completed by another technique called decomposition of atomicity [Butler, 2009a]. Model decomposition is another technique that can reduce the complexity of large models and increase their modularity. This technique consists in dividing a model into sub-models that can be refined separately and more easily than the original one. Several model's decomposition approaches have been proposed. Some of them are supported by *Rodin*[3] [Butler and Hallerstede, 2007] plugins[4] [Silva et al., 2011].

In this chapter, we define in more detailed way the Event-B method, its structure and the mathematical rules that allow to prove a model, as well as the proof obligation rules. Since Event-B is based on the notion of refinement, we present the different types of refinement and how the refinement can be verified. Then, we present a state of the art of the various types of decomposition in Event-B.

2.1 Event-B

Event-B [Abrial, 2010] is a model-oriented formal specification designed for the analysis of critical systems. It is based on the use of set theory and first-order logic as mathematical modelling foundations. This method is an extension of *Classical B/B-Method* [Abrial, 1996], for the software modelling, with some additional characteristics, as presented in section 1.1.3.

An *Event-B* model contains the complete mathematical development of a discrete transition system, allowing the modelling of static and dynamic aspects of a system. The static aspects concern the data, their typing and their specific properties. In the formal specification, these data are characterised in the form of constants or sets. On the other hand, the dynamic aspects are expressed by a group of events describing the states evolution.

[1] *PERFECT*: http://www.agence-nationale-recherche.fr/Projet-ANR-12-VPTT-0010
[2] CLEARSY: https://www.clearsy.com/
[3] *Rodin*: http://www.event-b.org/
[4] Modularisation: http://wiki.event-b.org/index.php/Modularisation_Plug-in

2.1.1 Structure of an Event-B Model

An *Event-B* model is composed of two types of components: context and machine. The context contains the static part of a model, namely sets, constants, axioms and theorems; while the machine holds the dynamic part of a model, i.e., the variables, invariants, theorems, variant and events.

Event-B Context

The context contains the static part of a model. It defines the sets, constants and properties (*axioms*). These properties define the predicates to be respected by the constants and the sets. A model can contain many contexts. Contexts can also extend or be extended by other contexts. In a case where a context C_1 extends another context C_0, C_1 can use the sets and the constants of C_0. In addition to its own constants, sets and properties, C_1 defines new constants, new sets and new properties. In general, the B/Event-B Method specification is divided into clauses where each one defines a different information about the system. In a context, the following clauses can be found:

- CONTEXT: in this clause, the name of the context is defined, and it should be distinct from the other components of the model.
- EXTENDS: defines the extended context if there is one, if not it can stay empty.
- SETS: defines the sets of this context.
- CONSTANTS: defines the constants of this context.
- AXIOMS: includes the axioms and theorems. It defines the properties of the constants and sets, such as the typing properties.

For example, in table 2.1, a context C_0 defines the sets s, the constants c and the axioms $A_0(s, c)$. This context can be extended by another context C_1 with the sets d, the constants t and the axioms $A_1(d, t)$ as in table 2.2.

CONTEXT	C_0	The context name
SETS	s	Sets of the context C_0
CONSTANTS	c	Constants of the context C_0
AXIOMS	$A_0(s, c)$	Axioms of the context C_0

Table 2.1: Context Structure

CONTEXT	C_1	The context name
EXTENDS	C_0	Extended context
SETS	d	Sets of the context C_1
CONSTANTS	t	Constants of the context C_1
AXIOMS	$A_1(d, t)$	Axioms of the context C_1

Table 2.2: Extending Context Structure

Event-B Machine

A machine defines the dynamic part of the model. It contains the variables and their properties (invariants), as well as variants and events. A machine can refine another machine or not and it can see many contexts or none. The machine clauses are defined as follows:

- MACHINE: defines the name of the machine. It should be distinct from the other components name of the model.
- REFINES: in this clause, the refined machine can be added. If the machine is not a refinement of another one, this clause is not used.
- SEES: a machine can see contexts or not. In this clause, we put the name of the seen contexts. If the machine does not need to see other contexts, this clause is not used.
- VARIABLES: specifies the variables of the machine.
- INVARIANTS: establishes the different properties that the machine must preserve.
- VARIANT: determines the variant of the system, i.e., the system stops after a certain number of transitions.
- INITIALISATION: establishes the variables initial values.
- EVENTS: describes the different events of the system, where the behaviour is presented through the event substitutions called actions.

For example, in table 2.3, a machine M_0 sees a context C_0. It defines the variables v, the invariants $I(v, s, c)$, the variant $V(v, s, c)$, the initialisation $K(v', s, c)$ and the events $event_i$.

MACHINE	M_0	Machine name
SEES	C_0	SEEN context name
VARIABLES	v	Variables of the machine
INVARIANTS	$I(v, s, c)$	Invariant to preserve in the machine
VARIANT	$V(v, s, c)$	Variant
INITIALISATION	$K(v', s, c)$	Initialisation
EVENTS	$event_i$	events of the machine

Table 2.3: Machine Structure

Events

A machine M_0 contains events $event_i$ which can be specified in three different ways (Figure 2.1):

- Simple (**BEGIN Q END**): where the guard is always true so it can be observed at any time. The variable v has a new value v' such as the substitution $v :|Q(v, v', s, c)$ changes the state of v to the new state v' where $Q(v, v', s, c)$ is the before-after predicate.
- Guarded (**WHEN G THEN Q END**): which is triggered when the guard $G(v)$ is satisfied and where the action $v :|Q(v, v', s, c)$ depends only on the state variables of the model.

– In-deterministic (**ANY p WHERE G THEN Q END**): is such that v is a state variable and p is a local variable of the event. In this specification, the event is triggered only if there exists a value of the variable p that satisfies the guard $G(p, v)$ such as $v :| Q(p, v, v', s, c)$ is the action of the event said non-deterministic (see table 2.4).

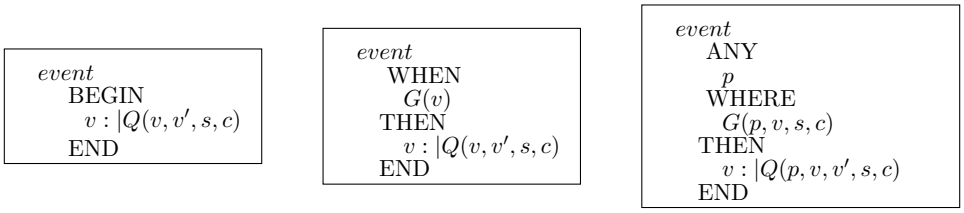


Figure 2.1: Different Types of Events

p	Event parameter
$G(p, v, s, c)$	Guard of the event
$Q(p, v, v', s, c)$	Before/After predicate

Table 2.4: In-deterministic Event Structure

Machine Types

There are two types of machines: an *abstract machine* and a *refinement machine*. In a first step of the system development process, an informal specification is modelled into an *abstract machine*. This machine defines the initial specification that reflects the behaviour of the system to be studied. Then, it can be refined by a *refinement machine*, which, in turn, can also be refined by another refinement machine and so on.

For example, a machine M_0 containing a set of variables v can be refined by another machine M_0 with another set of variables w. In this context, each variable in w is either refining the variables in v or it is a new variable of M_1. Besides, M_1 must contain an invariant $J(v, w, d, t)$, which is the gluing invariant describing the new variables properties stemming from the relations between the abstract variables and the refining variables.

Similarly, the events r_event_i, in M_1, are either new events of this machine or they are refining the abstract events a_event_i from M_0. In the refinement, r_event_i must define guards $H_i(q, w, d, t)$ and substitutions $w :| R_i(q, w, w', d, t)$. The refinement machine M_1 can also define new events n_event_k with the guards $N_k(o, w, d, t)$ and the substitutions $w :| T_k(o, w, w', d, t)$. M_1 $SEES$ a context C_1 which *extends* C_0, as in figure 2.2.

Machines and Contexts Relationships

Machines and contexts have different relationships. A machine can be refined by another machine, and a context can be extended by other contexts. In addition, a machine can see one or more contexts. When a machine M *sees* a context C, it means that the sets and the constants of C can be used in M. The different types of relationships between machines and contexts are illustrated in figure 2.3:

REFINEMENT	M_1	Refinement of the machine M0
REFINES	M_0	Refined machine
SEES	C_1	Context seen by M_1
VARIABLES	w	New variables and/or variables refining v
INVARIANTS	$J(v, w, d, t)$	Gluing invariant to be preserved in the refinement
INITIALISATION	$K_r(w', d, t)$	Initialisation in the refinement
EVENTS	r_event_i n_event_k	Events refining a_event_i New events
r_event_i	q $H_i(q, w, d, t)$ $R_i(q, w, w', d, t)$	Refining event parameter Guard of the refining event Before/After predicate
n_event_k	o $N_k(o, w, d, t)$ $T_k(o, w, w', d, t)$	New event parameter Guard of the new event Before/After predicate

Table 2.5: Refinement Machine Structure

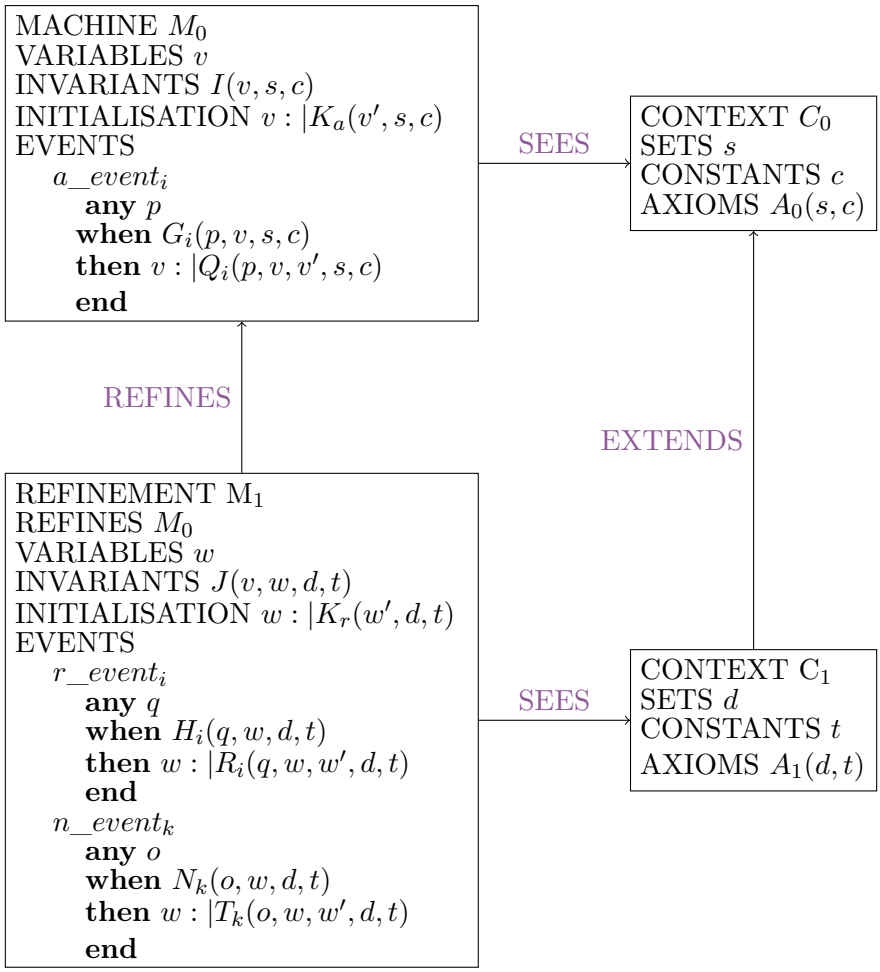


Figure 2.2: Structure of an Event-B Model with One Refinement and One Extending Context

– A machine can explicitly see multiple contexts (M_1 *sees* C_0 and C_1) or any context (M_0 does not see any context).

– A context can explicitly extend several contexts (C_3 *extends* C_1 and C_2) or any context (C_0 does not extend any context).

– When a context C_3 *extends* a context C_1, the sets and the constants of C_1 can be used in C_3.

– The notion of context extension is transitive: a context C_4 explicitly *extends* a context C_3. Then C_4 implicitly *extends* all extended contexts by C_3 (C_4 implicitly *extends* C_1 and C_2).

– A machine implicitly sees all contexts extended by a context explicitly seen (M_2 implicitly *sees* the contexts C_1 and C_2 explicitly extended by C_3).

– The relations *REFINES* and *EXTENDS* must not lead to a cycle.

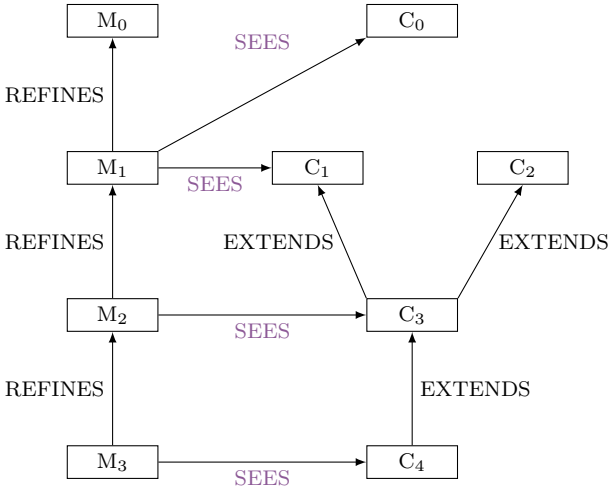


Figure 2.3: Different Possible Relations Between Contexts and Machines

2.1.2 Proof Obligation Rules in Event-B

In order to ensure that a model is correct, mathematical rules should be verified, called *proof obligations*. Different proof obligation rules exist. To facilitate the specification modelling, systems are modelled in several steps. The first step of modelling is the abstraction, it does not model all the details of the system. These details are specified and added gradually in a subsequent model which refines the abstract model. Then, at each stage of refinement, one should ensure that the operations of a machine preserve the invariant as the first main proof activity in B called *consistency checking*. The second main proof activity is *refinement checking*, which is used to show that one machine is a valid refinement of another [Leuschel and Butler, 2003]. In other words, a refinement relationship ensures consistency between two levels of modelling and is carried out in an incremental way up to a certain level. This level can be useful to analyse the behaviour of the system. This relationship involves a set of invariants that must be preserved to prove the refinement correctness.

Invariant POs

The invariant proof obligation allows to ensure the invariants preservation by each event. There exists two main invariant proof obligation types (see table 2.6):

- Initialisation invariant PO: named INV_{init}, it allows to verify if the initialisation establishes the invariant in the abstract machine by $INV_{init}1$ and in the refinement by $INV_{init}2$.
- Events invariant PO: allows to verify if each transition of the events preserves the invariants. For the abstract machine, the invariant proof obligation concerning the abstract events is denoted by INV_1. For the refinement machine, there are two types of invariant proof obligations: INV_2 for the refining events and INV_3 for the new events.

PO description	**PO name**	**PO Formula**
Preservation of the invariant $I(s, c, v)$ by the initialisation of the abstract machine M_0	INV$_{init}$1	$A(s, c)$ $K_a(v, s, c)$ $\vdash$ $I(v, s, c)$
Preservation of the invariant $J(v, w, d, t)$ by the initialisation of the refinement machine M_1	INV$_{init}$2	$A(d, t)$ $K_r(w, d, t)$ $\vdash$ $\exists v.(K_a(v, s, c) \wedge J(v, w, d, t)$
Preservation of the invariant by the events of the abstract machine M_0, where v' is the new state of v after the events observation	INV1	$A(s, c)$ $I(v, s, c)$ $G_i(p, v, s, c)$ $Q_i(p, v, v', s, c)$ $\vdash$ $I(v', s, c)$
Preservation of the refinement machine invariant by the refining events, where w' is the new state of w after the events observation	INV2	$A(d, t)$ $I(v, s, c)$ $J(v, w, d, t)$ $H_i(q, w, d, t)$ $R_i(q, w, w', d, t)$ $\vdash$ $J(v', w', d, t)$
Preservation of the refinement machine invariant by the new events	INV3	$A(d, t)$ $I(v, s, c)$ $J(v, w, d, t)$ $N_k(o, w, d, t)$ $T_k(o, w, w', d, t)$ $\vdash$ $J(v, w', d, t)$

Table 2.6: Invariant Proof Obligation Rules: INV

Feasibility POs

The feasibility proof obligation rule ensures that an event can be triggered at least one time. $FIS1$ verifies the feasibility of the initialization. $FIS2$ allows to verify the feasibility of the abstract events and $FIS3$ is for the feasibility of the refining events (see table 2.7):.

PO description	**PO name**	**PO Formula**
Feasibility of the initialisation	FIS1	$\vdash$ $\exists v.K(v,s,c)$
Feasibility of the events in the abstraction	FIS2	$A(s,c)$ $I(v,s,c)$ $G_i(p,v,s,c)$ $\vdash$ $\exists v'.Q_i(p,v,v',s,c)$
Feasibility of the events in the refinement	FIS3	$I(v,s,c)$ $J(v,w,d,t)$ $H_i(q,w,d,t)$ $\vdash$ $\exists w'.R_i(q,w,w',d,t)$

Table 2.7: Feasibility Proof Obligation Rules: FIS

Event POs

Table 2.8 presents the proof obligation rules corresponding to the events:

- GRD allows to make sure that the guards in a refining event are stronger than the abstract ones in the abstract event. This ensures that when a refining event is triggered, so is the corresponding abstract event.
- SIM ensures that when a refining event is triggered, it does not create any contradiction with the corresponding abstract event. Each action in an abstract event is correctly simulated in the corresponding refinement.

Deadlock-Freedom Proof Obligation Rules

If the system reaches a state where there are no outgoing transitions, the model is considered to be deadlocked. This proof obligation allows to verify if the system does reach a state where it is deadlocked. Table 2.9 shows the deadlock freedom proof obligations:

- $DLF1$ defines the proof obligation rule for the deadlock-freedom of the abstract machine. At least one event can be triggered.
- Two rules of deadlock-freedom proof obligations are defined for the refinement machine: the weaker one and the stronger one.
  → The weaker rule $DLF2_w$ means that at least one of the refining events is triggered.
  → The stronger rule $DLF2_s$ requires that each refining event is triggered at least one time.

PO description	**PO name**	**PO Formula**
Proof obligation of the refining events guards	GRD	$A(s,c)$ $I(v,s,c)$ $J(v,w,d,t)$ $H_i(q,w,d,t)$ $\vdash$ $G_i(p,v,s,c)$
Proof obligation of the refining events actions	SIM	$A(s,c)$ $I(v,s,c)$ $J(v,w,d,t)$ $H_i(q,w,d,t)$ $R_i(q,w,w',d,t)$ $\vdash$ $Q_i(p,v,v',s,c)$

Table 2.8: Event Proof Obligation Rules

– $DLF3_w$ is the weaker proof obligation to be verified in case of the existence of new events in the refinement machine. In the refinement machine, at least one of the existing events, new or refining ones, should be triggered.

– $DLF3_s$ is the stronger proof obligation to be verified in case of the existence of new events in the refinement machine. It requires that each refining event is triggered at least one time. Otherwise, at least one of the new events should be triggered.

Variant POs

In the case of introducing some new events in a refinement machine, we have to prove that they do not diverge. In other words, the new events must not be indefinitely enabled. This proof obligation allows to verify that a system stops after a certain number of transitions. The proof obligations of variant are defined as follows in table 2.10:

– NAT allows to prove that the variant $V(w,d,t)$ is a natural number assuming the axioms $A(d,t)$, the abstract invariant $I(v,s,c)$, the refinement machine invariant $J(v,w,d,t)$, and the guards of each new event $N_k(o,w,d,t)$.

– $VAR1$ verifies that the variant $V(w,d,t)$ is decreased. This has to be proved for each new event with guards $N_k(o,w,d,t)$ and before–after predicate $T_k(o,w,w',d,t)$. The same variant should be decreased by each new event.

– FIN allows to verify that the set $S(w)$ of the states of the variables w is finite.

– $VAR2$ ensures that the set $S(w')$ of the new states w' of w is included in $S(w)$.

PO description	**PO name**	**PO Formula**
Deadlock freedom of the abstract machine	DLF1	$I(v, s, c)$ $\vdash$ $G_1(p, v, s, c) \vee .. \vee G_n(p, v, s, c)$
Weak deadlock freedom of the refinement machine in case of the refining events existence	DLF2$_w$	$I(v, s, c)$ $J(v, w, d, t)$ $G_1(p, v, s, c) \vee .. \vee G_n(p, v, s, c)$ $\vdash$ $H_1(q, w, d, t) \vee .. \vee H_n(q, w, d, t)$
Strong deadlock freedom of the refinement machine in case of the refining events existence	DLF2$_s$	$I(v, s, c)$ $J(v, w, d, t)$ $G_i(p, v, s, c)$ $\vdash$ $H_i(q, w, d, t)$
Weak deadlock freedom of the refinement machine in case of the existence of refining and new events	DLF3$_w$	$I(v, s, c)$ $J(v, w, d, t)$ $G_1(p, v, s, c) \vee .. \vee G_n(p, v, s, c)$ $\vdash$ $H_1(q, w, d, t) \vee .. \vee H_n(q, w, d, t)$ $\vee N_1(o, w, d, t) \vee .. \vee N_m(o, w, d, t)$
Strong deadlock freedom of the refinement machine in case of the existence of refining and new events	DLF3$_s$	$I(v, s, c)$ $J(v, w, d, t)$ $G_i(p, v, s, c)$ $\vdash$ $H_i(q, w, d, t) \vee N_1(o, w, d, t) \vee .. \vee N_m(o, w, d, t)$

Table 2.9: Deadlock Freedom Proof Obligation Rules: DLF

PO description	**PO name**	**PO Formula**
Proof obligation of a decreasing natural	NAT	$A(s, c)$ $A(d, t)$ $I(v, s, c)$ $J(v, w, d, t)$ $N_k(o, w, d, t)$ $\vdash$ $V(w, d, t) \in N$
Variant proof obligation using a natural	VAR1	$I(v, s, c)$ $J(v, w, d, t)$ $N_k(o, w, d, t)$ $T_k(o, w, w', d, t)$ $\vdash$ $V(w', d, t) < V(w, d, t)$
Proof obligation of a set finiteness	FIN	$I(v, s, c)$ $J(v, w, d, t)$ $N_k(o, w, d, t)$ $\vdash$ $finite(S(w))$
Variant proof obligation using a finite set	VAR2	$I(v)$ $J(v, w, d, t)$ $N_k(o, w, d, t)$ $T_k(o, w, w', d, t)$ $\vdash$ $S(w') \subset S(w)$

Table 2.10: Variant Proof Obligation Rules: VAR

2.2 Refinement in Event-B

2.2.1 Event-B Refinement Types

In *Event-B*, refinement is a central concept used for modelling the system incrementally from an abstract machine on the basis of the system specification. At each stage of refinement, details of the system are gradually added in a concrete machine that must preserve the functionality and the properties of the refined machine. As a matter of fact, an abstract machine can be refined by only one refinement and a refinement machine refines only one abstract machine. For that reason, we consider the refinement in B/Event-B as "linear" in the sequel of this manuscript.

Two Event-B refinement techniques exist: horizontal refinement and vertical refinement [Abrial et al., 1991, Bolusset and Oquendo, 2002]. The latter contains the *data refinement* and the *events refinement*. The different refinement techniques are defined as follows:

Horizontal refinement: consists in adding the specification details in order to define progressively new functionalities of the system in the refinement such as introducing new variables and new events that make these new variables evolve. New events refine a particular event of an abstract machine which is the empty event with *skip* substitution.

Vertical refinement: has as a goal the concretisation of the abstract machine by adding variables through a *data refinement* [Back, 1989] and the behaviour by detailing abstract events or adding new events by *events refinement*, also called *algorithmic refinement* [Abrial et al., 1991]. These two types of refinement, *data refinement* and *algorithmic refinement*, are not exclusive: they can be operated in the same stage of refinement. It is obvious that any refinement of data leads to an algorithmic refinement. These two types of vertical refinement are detailed below:

- *Data refinement*: consists in defining concrete variables w in the refinement machine in order to replace abstract variables v. Since the substitutions no longer make the same abstract variable v space evolve, they must be rewritten (refined) with respect to the new variable w space. In this case, a predicate $J(v, w)$, called a *gluing invariant*, must be specified. This invariant makes it possible to establish the link between the variables v and w. The gluing invariant $J(v, w)$ is specified in the INVARIANT clause of the refining component. Proof obligations are generated at each refinement stage to ensure the refinement correctness (see section 2.1.2).

  In classical B-Method, refinement is based on this technique to bring the model closer to the implementation.

- *Events refinement*: aims to refine an abstract event by one or many events in the refinement machine in order to make the event more concrete. It is the rewriting of an abstract substitution evolving v into a less abstract substitution evolving w. A graphical approach of events refinement has been presented in [Butler, 2009a, Dghaym et al., 2017, Dghaym et al., 2016] called Event Refinement Structures (ERS). Its main goal is to represent explicitly the events refinement and the behaviour sequencing [Fathabadi et al., 2011].

Event Refinement Structures

A proposition in [Butler, 2009a] is called decomposition of event atomicity. This approach is a structuring mechanism for refinement in Event-B. This mechanism is based on decomposing an abstract atomic event to many sub-events, where one event refines this abstract event. Decomposing atomic events is inspired from Jackson System Development (JSD) approach [Butler, 2009b] and it is represented by the ERS approach (Event Refinement Structures) [Dghaym et al., 2016, Dghaym et al., 2017]. The idea of the ERS approach is to enrich the Event-B refinement with a graphical tree notation able to represent explicitly the events decomposition in the refinement and the behaviour sequencing [Fathabadi et al., 2011]. Figure 2.4 presents a sub-tree. The child nodes of each node are transformed into events in the refinement. The nodes order describes the order of events observation (from left to right).
This method is defined as follows:

- The root of the tree represents the abstract machine (here we represent a sub-tree).
- The child nodes of each node are transformed into events in the refinement.
- The order of leaves / nodes determines the order of observation of events (from left to right).
- The dotted line indicates the addition of new events.
- The solid line indicates that an event refines the parent event. At most one child event can refine a parent event.
- XOR indicates the triggering of one and only one event.
- In case of XOR, an event can be refined by several events.
- AND allows interleaved execution of events.

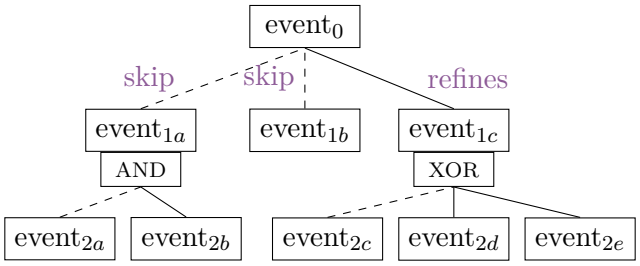


Figure 2.4: Example of Event Refinement Structures (ERS) Diagram

For example, let consider the machine of figure 2.5, the machine M1 on the left side refines a machine M0 which contains the abstract specification of "AbstractEvent". The M1 machine controls the sequence of events "Event1" and "Event2" with guards on these events. The control of this sequence is presented in the ERS diagram on the right side. The solid line indicates that "Event2" refines "AbstractEvent" while the dashed line indicates that "Event1" is a new event that refines "skip". In the Event-B model on the left side, 'Event1' has no explicit relation with 'AbstractEvent', but the diagram indicates that the atomicity of 'AbstractEvent' is broken into two

sub-events in the refinement. The parameter "par" of the diagram indicates that we are modelling several instances of AbstractEvent and sub-events. The effect of an event with the "par" parameter is to add the value of "par" to a control variable defined with the same name as the event, that is, "$par \in Event1$" means that Event1 occurred with the value "par". Using a set means that the same event can occur multiple times with different values for "par".

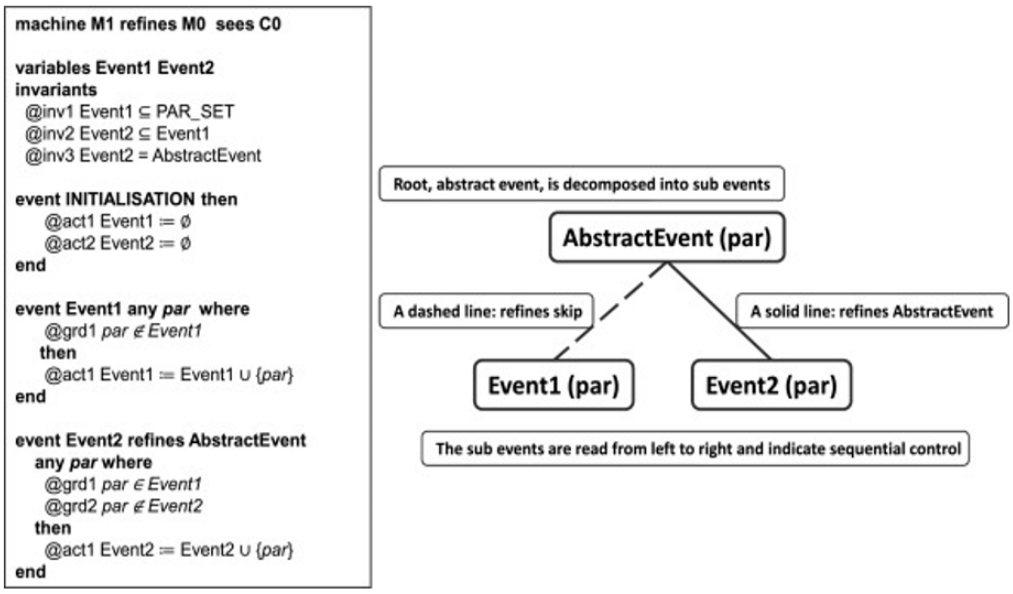


Figure 2.5: Example of an Event Structure Diagram [Alkhammash et al., 2015]

2.2.2 Correctness of the Event-B Refinement

This section relies on the Abrial definition of the refinement in the Event-B book [Abrial, 2010], but to well understand and justify this definition and the underlying relations and rules, this leads us to explain the correctness of the Event-B refinement in another manner, in our own way. This will be useful for our decomposition approach proposal.

In order to simplify the explanation, let consider as an example an abstract machine that gives the maximum of a set of positive integers: GET_MAX example. The abstract machine, in the left side of figure 2.6, contains the following elements:

- *Set*: a variable that is a non-empty set of positive integers (NAT). *Set* is initialised to the singleton {0}.
- *current_max*: a variable containing the current maximum of *Set*.
- *add*: an event that allows to add a positive integer to *Set*.
- *get_max*: an event that returns the maximum of *Set*.

At this level, we notice that only new values are added to the set. Hence, it is useless to keep in the memory the already inserted values in the set: only the maximum is interesting. So, we

```
SYSTEM
    GET_MAX
VARIABLES
    Set,
    current_max
INVARIANT
    Set : FIN (NATURAL) &
    current_max : NATURAL &
    Set /= {}
INITIALISATION
    Set := {0}
    || current_max := 0
EVENTS
    add =
        ANY
            nn
        WHERE
            nn : NATURAL
        THEN
            Set := Set \/ {nn}
        END;
    get_max =
        BEGIN
            current_max := max (Set)
        END
END
```

```
REFINEMENT GET_MAX_r
REFINES GET_MAX
VARIABLES
    Max,
    current_max
INVARIANT
    Max : NATURAL &
    Max = max(Set)
INITIALISATION
    Max := 0
    || current_max :=0
EVENTS
    add =
        ANY
            nn
        WHERE
            nn : NATURAL
            & nn > Max
        THEN
            Max := nn
        END;
    get_max =
        BEGIN
            current_max := Max
        END
END
```

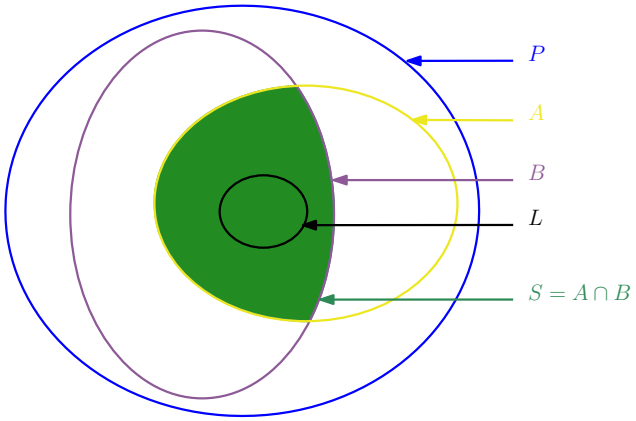
Figure 2.6: GET_MAX Example: Abstract Machine and its Refinement

refine this machine by deleting the variable *Set* and keeping only the current maximum of *Set*. We present, in the right side of figure 2.6, the Event-B refinement of GET_MAX model:

- *Max*: a variable keeping the maximum positive integer.
- *current_max*: a variable containing the current maximum positive integer.
- *add*: an event that allows to add a positive integer as maximum if it is bigger than the previous value of *Max*.
- *get_max*: an event that returns the current maximum.

Before dealing with this example, we introduce some sets definitions that seem to be useful for the sequel:

- P: defines the state space, i.e. a set of all the possible states.
- L: set of the initial states.
- A: set of reached states by the transition system.
- B: set of "safe" states, which preserve the invariant.
- $S = A \cap B$: set of the reached states by the transition system preserving the invariant.



We define in table 2.11 the following sets of the abstract machine and its refinement:

Set Name	Set Definition	Set Formula
S	Set of the reached states by the transition system preserving the invariant in the abstract machine	$S = \{v \mid I(v)\}$
T	Set of the reached states by the transition system preserving the invariant in the refinement machine	$T = \{w \mid \exists v.(I(v) \wedge J(v,w))\}$
L_a	Set of the initialisation states in the abstraction	$L_a = \{v \mid K(v)\}$
L_r	Set of the initialisation states in the refinement	$L_r = \{w \mid N(w)\}$

Table 2.11: Sets Definitions

Abstract machine Back to the example, in the abstract machine, the abstract state variables are defined as $v \triangleq (Set, current\_max)$, the Set is defined as $Set \subseteq \{0,1,2\}$ and $current\_max$ as $current\_max \in Set$. Here, we restrict the space state of the Set and of the $current\_max$. The invariant $I(v)$ is defined as $Set \neq \varnothing$. The sets introduced above are defined as follows:

- P = {(∅, 0), (∅, 1), (∅, 2), ({0}, 0), ({0}, 1), ({0}, 2), ({1}, 0), ({1}, 1), ({1}, 2), ({2}, 0), ({2}, 1), ({2}, 2), ({0, 1}, 0), ({0, 1}, 1), ({0, 1}, 2), ({0, 2}, 0), ({0, 2}, 1), ({0, 2}, 2), ({1, 2}, 0), ({1, 2}, 1), ({1, 2}, 2), ({0, 1, 2}, 0), ({0, 1, 2}, 1), ({0, 1, 2}, 2)}.
- $L_a = \{\{0\}, 0\}$.
- B = P - {(∅, 0), (∅, 1), (∅, 2)}. The set B contains all possible states minus all states breaking the invariant.

– S = $\{(\{0\},0),(\{0,1\},0),(\{0,1\},1),(\{0,2\},0),(\{0,1,2\},0),(\{0,1,2\},1),(\{0,2\},2),(\{0,1,2\},2)\}$.

Refinement machine In the refinement, we refine the variable $current\_max$ and define a new variable Max. So, the state variable of the refinement is defined as $w \triangleq (Max, current\_max)$ with the gluing invariant $J(v,w) : Max = \mathbf{max}(Set)$. T is a set of the reached states by the transition system preserving the gluing invariant in the refinement. Similarly to the abstract machine, we define in the refinement T the set of the reached states by the transition system preserving the invariant where T = $\{(0,0),(1,0),(1,1),(2,1),(2,0),(2,2)\}$, as shown in the right side of figure 2.7.

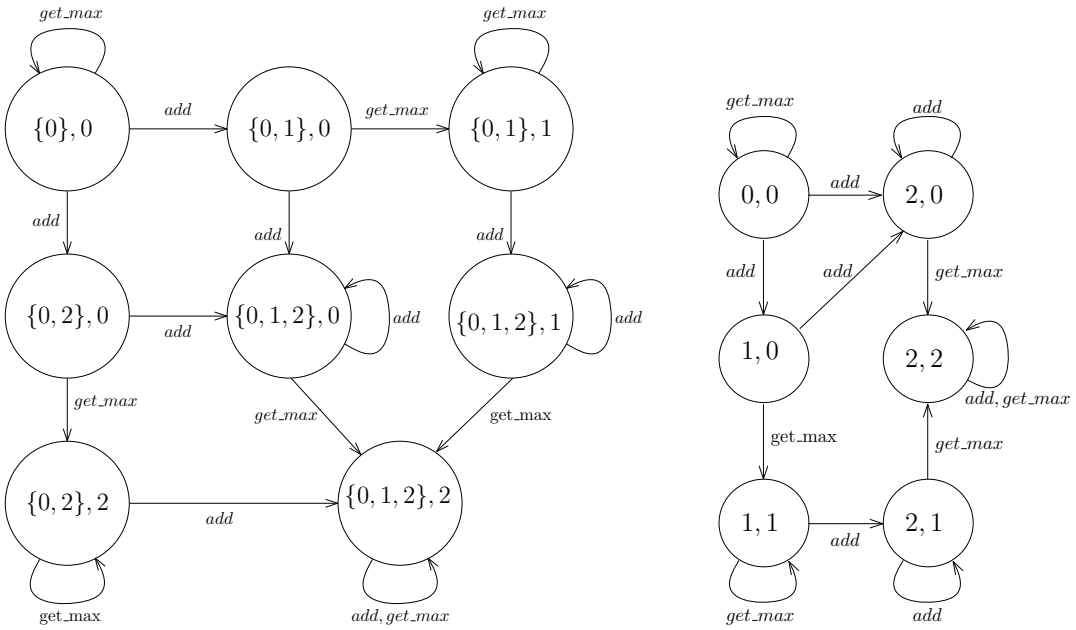


Figure 2.7: GET_MAX Example: Different Possible Transitions

Demonstration In [Abrial, 2010], the author defines three relations, as shown in figure 2.8:

– ae: presents all the possible abstract transitions between the states in S in the abstract machine. In other words, ae defines the couples (v, v') such that the invariants $I(v)$, the guards $G(v)$ and the before/after predicates $R(V, v')$ are true.

– re: defines all the possible refinement transitions between the states in T in the refinement machine. For each couple (w, w') in re, it exists refined variables v of the abstract machine such that the invariant $I(v)$, the gluing invariants $J(v, w)$, the guard $H(w)$ and the before/after predicates $Q(w, w')$ are true.

– r: is a refinement relation between the abstract machine states and the refinement states. For each couple (w, v) the invariants $I(v)$ and the gluing invariants $J(v, w)$ are true, where w are the state variables of the refinement machine and v are the state variables of the abstract machine.

Table 2.12 illustrates the definitions and the formulas of ae, re, the domain of ae and the domain of re.

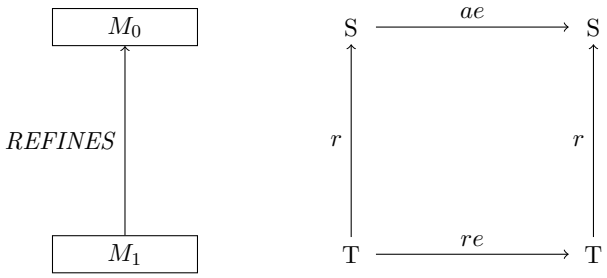


Figure 2.8: Relations Between an Abstract Machine and its Refinement

Abstract transitions	$ae \in S \leftrightarrow S$	$ae = \{v \mapsto v' \mid I(v) \wedge G(v) \wedge R(v, v')\}$
Refining transitions	$re \in T \leftrightarrow T$	$re = \{w \mapsto w' \mid (\exists v.I(v) \wedge J(v, w)) \wedge H(w) \wedge Q(w, w')\}$
Relation of refinement	$r \in T \nleftrightarrow S$	$r = \{w \mapsto v \mid I(v) \wedge J(v, w)\}$
Domain of the abstract transitions	$dom(ae)$	$dom(ae) = \{v \mid I(v) \wedge G(v)\}$
Domain of the refining transitions	$dom(re)$	$dom(re) = \{w \mid \exists v.(I(v) \wedge J(v, w)) \wedge H(w)\}$

Table 2.12: Definition of the Relations Between an Abstract Machine and its Refinement

Notion of external variables The state variables v are distributed into two categories: external variables e and internal variables i. External variables e, also called observable variables [Abrial, 2010, Abrial and Hallerstede, 2007] are the state variables of an abstract machine that are refined in a refinement machine. The external variables of a machine are formally linked to the external variables of its refinement [Metayer et al., 2005]. The variables which are not refined are considered as non-observable called internal variables i. As a consequence, the sets of external variables are defined as follows:

- E: a set of external variables states of the abstract machine.
- F: a set of external variables states of the refinement.

In GET_MAX example, *current_max* is an external variable of the abstract machine, so E is defined such as $E = \{0, 1, 2\}$. *Set* is an internal variable of the abstract machine. As a par-

ticular case, we consider *current_max* the external variable of the refinement so that $F = \{0, 1, 2\}$.

Correction of refinement Abrial defines some functions in the basis of the sets E and F for the purpose of the refinement correctness demonstration, as in figure 2.9:

- f is a function from S to E such as $f \in S \rightarrow E$. It is a projection of the safe reached states on the external variables states of the abstract machine.
- g is a function from T to F such as $g \in T \rightarrow F$. It is a projection of safe reached states on the external variables states of the refinement.
- h is a function from F to E such as

  $h \in F \rightarrow E$. It is a function that links the external variables of the abstract machine with those of its refinement. h is defined by the observer event (observe event) **BEGIN** $w := h(v)$ **END**, where w is the external variables of the refinement that refine the external variables v of the abstract machine.

In GET_MAX example, h is defined as the *identity* function such that $h(current\_max) = current\_max$. From that and since $w = h(current\_max)$, this implies $w = current\_max$.

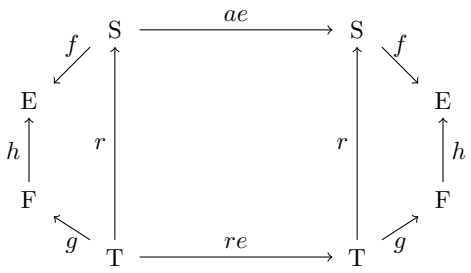


Figure 2.9: Relations Between State Variables and Observable Variables [Abrial, 2010]

After that, a property between r on the one hand and f, g and h on the other hand is defined:

$$\textbf{(P1)} \quad \forall v, w.(w \mapsto v \in r \Rightarrow f(v) = h(g(w)))$$

In Event-B, in order to ensure that a refinement is correct it is necessary to check the conditions in figure 2.10. **C1** and **C2** are the initialisation conditions, **C3** concerns the events and **C4** is relative to the deadlock freedom. These conditions are as follows:

- **C1**: the set of the initialisations in the refinement should be included in the abstract one.
- **C2**: the initialising set of the refinement machine shouldn't be empty which means that at least one state of the refinement machine must occur.
- **C3**: all the transitions of the refinement should be included in those of the abstraction. In other words, the behaviour of the refinement is at most equal to the behaviour of the abstraction. In the refinement, we shouldn't have a behaviour which does not exist in the abstraction.

– **C4**: the domain of abstract transitions ae is included in the domain of refinement transitions re, i.e. the set of states allowing to trigger a transition in the refinement is including the set of states allowing to trigger a transition in the abstraction.

In the right side of figure 2.10, we add an equivalence rewriting of these conditions in function of f, g and h functions as defined by Abrial in [Abrial, 2010].

(C1)	$L_r \subseteq L_a$		**(C1)**	$g[L_r] \subseteq h^{-1}[f[L_a]]$
(C2)	$L_r \neq \varnothing$	**P1**	**(C2)**	$L_r \neq \varnothing$
(C3)	$re \subseteq ae$	$\Longleftrightarrow$	**(C3)**	$(g^{-1}; re; g) \subseteq (h; f^{-1}; ae; f; h^{-1})$
(C4)	$dom(ae) \subseteq dom(re)$		**(C4)**	$h^{-1}[f[dom(ae)]] \subseteq g[dom(re)]$

Figure 2.10: Conditions to Verify for the Refinement Correctness

On the basis of these conditions and the supposed property **P1**, we have in [Abrial, 2010]:

$$
\begin{aligned}
&\textbf{(P1)}: && \forall v, w.(w \mapsto v \in r \Rightarrow f(v) = h(g(w))) \\
&\Longleftrightarrow \\
& && \forall v, w.(w \to v \in r \Rightarrow g(w) \to f(v) \in h) \\
&\Longleftrightarrow \\
& && \forall v, w, z.(z = g(w) \wedge w \to v \in r \Rightarrow z \to f(v) \in h) \\
&\Longleftrightarrow \\
& && \forall v, z.(\exists w.(z = g(w) \wedge w \to v \in r) \Rightarrow z \to f(v) \in h) \\
&\Longleftrightarrow \\
& && \forall v, z.(\exists w.(z = g(w) \wedge w \to v \in r) \Rightarrow \exists u.(u = f(v) \wedge z \to u \in h)) \\
&\Longleftrightarrow \\
& && \forall v, z.(\exists w.(v \to w \in r^{-1} \wedge w \to z \in g) \Rightarrow \exists u.(v \to u \in f \wedge u \to z \in h^{-1})) \\
&\Longleftrightarrow \\
& && \forall v, z.(v \to z \in (r^{-1}; g) \Rightarrow v \to z \in (f; h^{-1})) \\
&\Longleftrightarrow \\
&\textbf{(P1)}: && \boxed{r^{-1}; g \subseteq f; h^{-1}}
\end{aligned}
$$

Goal 1: we intend to apply this demonstration of a classical refinement on our decomposition by refinement approach in chapter 4: *Refinement Seen Split* (*RSS*).

In the next section, we introduce the decomposition in Event-B and we detail some existing approaches of Event-B decomposition in the literature.

2.3 Decomposition in Event-B

An Event-B machine can have so many events and state variables that an additional refinement can become difficult to manage. Model decomposition tackles this difficulty by providing a mechanism to divide a large model into several sub-models. In fact, a large model can be partitioned into smaller components after several steps of refinement. This step of partitioning can be a result of a model complexity or simply an architectural decision [Silva and Butler, 2010]. The top-down

Modelling style used in Event-B allows, during the refinement levels, the introduction of new events and variables. A consequence of this style of development is an increasing complexity of the refinement process when dealing with many events and variables. Decomposing models addresses this difficulty by providing a mechanism to divide a large model into several sub-models. Four descending steps are defined for the different decomposition techniques by [Hoang et al., 2011]:

1. Model the system abstractly by expressing all the main global properties of the system;
2. Refine the abstract model to adapt it to the expected structure by a given decomposition technique;
3. Apply decomposition;
4. Develop the resulting sub-systems independently.

By following this guideline, the overall properties are captured early in the model and guaranteed in the final models by combining refinement and decomposition. The development of each decomposed part is done independently of the others. Therefore, we can have different implementations for a decomposed model which is guaranteed to work with any implementation of other decomposed models.

Many techniques for decomposing Event-B models have been proposed. These decomposition techniques differ in that the different elements of the model are shared between the sub-components: variables or events. In the literature, the most known approaches of decomposition in Event-B are decomposition by shared variables and decomposition by shared events. For shared variables decomposition, part of the state information (variables) is shared between the sub-components. For the breakdown of shared events, a set of events is synchronised and shared by sub-components. There are also other methods that decompose such as modularisation, instantiation, fragmentation and distribution we present in detail these approaches in the following sub-sections.

2.3.1 Decomposition by Shared Variables

Methodology of the Decomposition by Shared Variables

Abrial proposes in [Abrial and Hallerstede, 2007] the shared variables decomposition which consists in distributing events of a machine between several sub-machines. This approach proposes to manage shared variables between several events. It is also used for decomposing parallel programs [Hoang and Abrial, 2010]. During the machine decomposition, events to be separated are selected in each sub-machine and considered as internal events. A variable that occurs only in the internal events is a private variable. If a variable is involved in internal events of different sub-machines, it is defined in each of them as a shared variable that cannot be refined. External events of a sub-machine are events that simulate the change of state of the shared variables in the abstract machine.

Figure 2.11 illustrates the decomposition by shared variables. The machine M_0 is defined by four events and it is decomposed into two sub-machines M_{1a} and M_{1b} by partitioning its events. Events *event1* and *event2* (resp. *event3* and *event4*) are internal events to the sub-machine M_{1a}

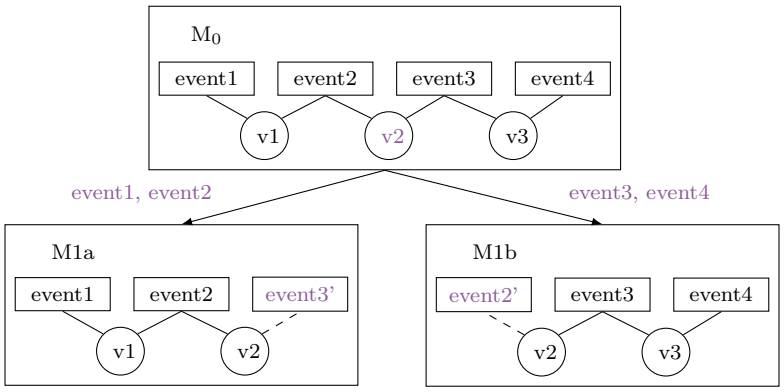


Figure 2.11: Decomposition by Shared Variable

(resp. M_{1b}). The variable *v1* (resp. *v3*) is private to M_{1a} (resp. M_{1b}). As for *v2*, it is a shared variable. Consequently, the machine M_{1a} (resp. M_{1b}) contains the external event *event3'* (resp. *event2'*) which simulates the state changes made by *event3* (resp. *event2*) on *v2* in M_0.

Correctness of the Decomposition by Shared Variables

In [Abrial, 2002, Abrial, 2009], after proceeding with the decomposition, the re-composition should be proved without explicitly composing. All the variables of sub-machines are put together and the external events are thrown away. We think that this is the reason behind.

Let M be the machine to decompose, P and N are the resulting sub-machines. NR and PR are respectively the resulting machines after several steps of refinement of N and P, as shown in the left side of figure 2.12.

S is the set of state variables in M. T and U are respectively the sets of states variables of N and P. X and Y are respectively the sets of states variables of NR and PR as shown in the right side of figure 2.12. MR is the theoretical re-composition of NR and PR, and Z is the corresponding set of its state variables.

Table 2.13 defines the different transitions in each machine and their refinement relations.

l, m and n are refinement relations as defined in section 2.2.2 with r. Let also consider these definitions as in table 2.14, predicates and lemmas as in table 2.15.

We detail the proof of lemmas $l1$ and $l2$ which are useful for the theoretical re-composition demonstration.

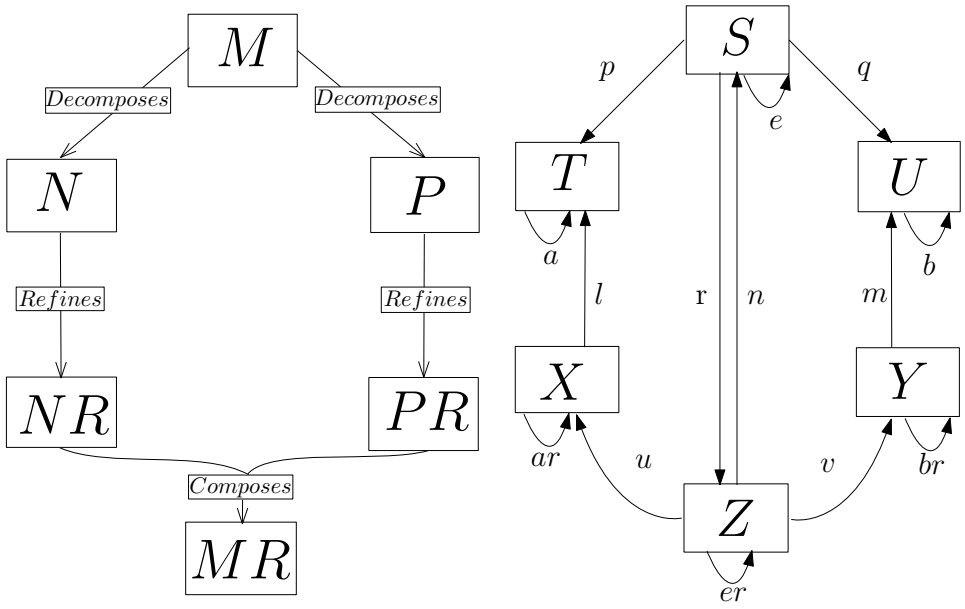


Figure 2.12: Sets of States Variables Corresponding to Each Machine

Transitions of M	e	$\in$	$S \leftrightarrow S$
Projection from S to T	p	$\in$	$S \twoheadrightarrow T$
Projection from S to U	q	$\in$	$S \twoheadrightarrow U$
Transitions of N	a	$\in$	$T \leftrightarrow T$
Transitions of P	b	$\in$	$U \leftrightarrow U$
Refinement relation from X to T	l	$\in$	$X \twoheadleftarrow T$
Refinement relation from Y to U	m	$\in$	$Y \twoheadleftarrow U$
Projection from Z to X	u	$\in$	$Z \twoheadrightarrow X$
Projection from Z to Y	v	$\in$	$Z \twoheadrightarrow Y$
Recomposed transitions on MR	er	$\in$	$Z \leftrightarrow Z$
Refinement relation from Z to S	n	$\in$	$Z \twoheadleftarrow S$
Relation from S to Z	r	$\in$	$S \rightarrow Z$
Transitions of NR	ar	$\in$	$X \leftrightarrow X$
Transitions of PR	br	$\in$	$Y \leftrightarrow Y$

Table 2.13: Formal Definitions of the Different Elements

d1	a	$\triangleq$	$p^{-1}; e; p$
d2	b	$\triangleq$	$q^{-1}; e; q$
d3	er	$\triangleq$	$(u; ar; u^{-1}) \cap (v; br; v^{-1})$
d4	n	$\triangleq$	$(u; l; p^{-1}) \cap (v; m; q^{-1})$
d5	r	$=$	$(p; p^{-1}; e; p; l^{-1}; u^{-1}) \cap (q; q^{-1}; e; q; m^{-1}; v^{-1})$

Table 2.14: Formal Definitions of the Relations

pr1	$l^{-1}; ar \subseteq a; l^{-1}$
pr2	$m^{-1}; br \subseteq b; m^{-1}$
l1	$n^{-1}; u \subseteq p; l^{-1}$
l2	$n^{-1}; v \subseteq q; m^{-1}$

Table 2.15: Predicates and Lemmas

Proof of lemma $l1$:

$$
\begin{array}{ll}
& n^{-1}; u \\
= & \qquad \text{definition of } n\text{: } n \triangleq (u; l; p^{-1}) \cap (v; m; q^{-1}) \\
& ((p; l^{-1}; u^{-1}) \cap (q; m^{-1}; v^{-1})); u \\
\subseteq & \qquad \text{set theory (distributivity)} \\
& (p; l^{-1}; \underline{u^{-1}; u}) \cap (q; m^{-1}; v^{-1}; u) \\
= & \qquad u \text{ is a total surjection: } u^{-1}; u \Leftrightarrow id \\
& (p; l^{-1}) \cap (q; m^{-1}; v^{-1}; u) \\
\subseteq & \qquad \text{set theory: } A \cap B \subseteq A \\
& p; l^{-1}
\end{array}
$$

Proof of lemma l2:

$$
\begin{array}{ll}
& n^{-1}; v \\
= & \qquad \text{definition of } n\text{: } n \triangleq (u; l; p^{-1}) \cap (v; m; q^{-1}) \\
& ((p; l^{-1}; u^{-1}) \cap (q; m^{-1}; v^{-1})); v \\
\subseteq & \qquad \text{set theory (distributivity)} \\
& (p; l^{-1}; u^{-1}; v) \cap (q; m^{-1}; \underline{v^{-1}; v}) \\
= & \qquad u \text{ is a total surjection: } v^{-1}; v \Leftrightarrow id \\
& (p; l^{-1}; u^{-1}; v) \cap (q; m^{-1}) \\
\subseteq & \qquad \text{set theory: } A \cap B \subseteq B \\
& q; m^{-1}
\end{array}
$$

The demonstration of the theoretical re-composition is tantamount to demonstrate that er is a refinement of e:

$$n^{-1}; er \subseteq e; n^{-1}$$

In other words, since r, $(n^{-1}; er)$ and $(e; n^{-1})$ are relations, it is sufficient to prove that

$$\textbf{(1)}\ n^{-1}; er \subseteq r$$

and

$$\textbf{(2)}\ r \subseteq e; n^{-1}$$

Proof of (1): $n^{-1}; er \subseteq r$

$n^{-1}; er$

$=$ — definition of er. $er \triangleq (u; ar; u^{-1}) \cap (v; br; v^{-1})$

$n^{-1}; ((u; ar; u^{-1}) \cap (v; br; v^{-1}))$

$\subseteq$ — set theory

$(n^{-1}; u\ ; ar; u^{-1}) \cap (n^{-1}; v\ ; br; v^{-1})$

$\subseteq$ — lemmas l1: $(n^{-1}; u \subseteq p; l^{-1})$, l2: $(n^{-1}; v \subseteq q; m^{-1})$

$(p; l^{-1}; ar\ ; u^{-1}) \cap (q;\ m^{-1}; br\ ; v^{-1})$

$\subseteq$ — refinements of a and b. pr1: $(l^{-1}; ar \subseteq a; l^{-1})$, pr1: $(m^{-1}; br \subseteq b; m^{-1})$

$(p; a;\ l^{-1}; u^{-1}) \cap (q; b; m^{-1}; v^{-1})$

$=$ — definitions of a and b. d1: $a \triangleq p^{-1}; e; p$, d2: $b \triangleq q^{-1}; e; q$

$(p; p^{-1}; e; p; l^{-1}; u^{-1}) \cap (q;\ q^{-1}; e; q; m^{-1}; v^{-1})$

$=$

r

Proof of (2): $r \subseteq e; n^{-1}$ this one is well explained in [Abrial, 2009].

Example of the Application of the Shared Variable Decomposition

To illustrate how this approach works, we will apply it on an example. Let consider the *Train System* case study presented in the Event-B Book, *Modeling in Event-B: System and Software Engineering*. The purpose of this case study is to help the train agent controlling trains and to have trains safely circulating in a certain network, as illustrated in figure 2.13. The case study specification is explained in details in chapter 17 of the Event-B Book [Abrial, 2010].

In order to apply this approach, let take the machine *train_0*, as in figure 2.14 . This machine defines the set of reserved routes, the set of reserved blocks, the association between reserved routes

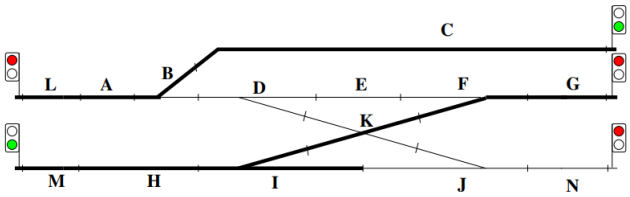


Figure 2.13: Train System Case Study

and blocks, as well as the set of the occupied block. Each occupied block is a reserved one. A set of blocks constitutes a route. Concerning the events, this machine describes the train movement on different blocks of the same route, the train entering to new reserved route, the route reservation and the route freeing.

```
MACHINE
  train_0
SEES
  train_ctx0
VARIABLES
  resrt      //   set of reserved routes
  resbl      //   set of reserved blocks
  rsrtbl      //   reserved route of reserved block
  OCC      //   occupied block
INVARIANTS
  inv1   :   resrt ⊆ R      //   a reserved route is a route
  inv2   :   resbl ⊆ B      //   a reserved block is a block
  inv3   :   rsrtbl ∈ resbl ⟶ resrt
  inv5   :   rsrtbl ⊆ rtbl      //   the reserved route of a reserved block is a route of that block
  inv4   :   OCC ⊆ resbl      //   occupied blocks are reserved
  inv6   :   ∀r· r∈R ⇒ nxt(r)[rtbl~[{r}]\rsrtbl~[{r}]] ∩ (rsrtbl~[{r}]\OCC) = ∅
  inv7   :   ∀r· r∈R ⇒ nxt(r)[rsrtbl~[{r}]] ⊆ rsrtbl~[{r}]
  inv8   :   ∀r· r∈R ⇒ nxt(r)[rsrtbl~[{r}]\OCC] ⊆ rsrtbl~[{r}]\OCC
EVENTS
  INITIALISATION   ≙
  STATUS
   ordinary
  BEGIN
   act1   :   resrt ≔ ∅
   act2   :   resbl ≔ ∅
   act3   :   rsrtbl ≔ ∅
   act4   :   OCC ≔ ∅
  END

  route_reservation   ≙
  STATUS
   ordinary
  ANY
   r
  WHERE
   grd1   :   r ∉ resrt
   grd2   :   rtbl~[{r}]∩resbl = ∅
  THEN
   act1   :   resrt ≔ resrt∪{r}
   act2   :   rsrtbl ≔ rsrtbl ∪ (rtbl⩥{r})
   act3   :   resbl ≔ resbl ∪ rtbl~[{r}]
  END

  route_freeing   ≙
  STATUS
   ordinary
  ANY
   r
  WHERE
   grd1   :   r ∈ resrt\ran(rsrtbl)
  THEN
   act1   :   resrt ≔ resrt\{r}
  END

  FRONT_MOVE_1   ≙        //   the train enters its route
  STATUS
   ordinary
  ANY
   r
  WHERE
   grd1   :   r ∈ resrt
   grd2   :   fst(r) ∈ resbl\OCC
   grd3   :   rsrtbl(fst(r)) = r
  THEN
   act1   :   OCC ≔ OCC ∪ {fst(r)}
  END

  FRONT_MOVE_2   ≙
  STATUS
   ordinary
  ANY
   b
   c
  WHERE
   grd1   :   b∈OCC
   grd2   :   c∉OCC
   grd3   :   b↦c ∈ nxt(rsrtbl(b))
  THEN
   act1   :   OCC ≔ OCC∪{c}
  END

  BACK_MOVE   ≙
  STATUS
   ordinary
  ANY
   b
   n
  WHERE
   grd1   :   b∈OCC
   grd2   :   n = nxt(rsrtbl(b))
   grd3   :   b∈dom(n) ⇒ n(b)∈OCC
   grd4   :   b∈ran(n) ∧
              n~(b)∈dom(rsrtbl)
              ⇒
              rsrtbl(n~(b))≠rsrtbl(b)
  THEN
   act1   :   OCC ≔ OCC\{b}
   act2   :   rsrtbl ≔ {b}⩤rsrtbl
   act3   :   resbl ≔ resbl\{b}
  END

END
```

Figure 2.14: Train System Case Study: Abstract Machine

Actually, this case is studying the track network structure on one hand, and the train object on the other hand. This constitutes a good example to apply the A-style in order to split the behaviour of the train and the behaviour of the track.

In order to apply the A-style, the events to split are chosen. The train sub-machine contains the events describing the train movement : $FRONT\_MOVE_1, FRONT\_MOVE_1$ and

BACK_MOVE. The other sub-machine, describing the track behaviour, contains the events : *route_reservation* and *route_freeing*. After that the decomposition is done.

In the train sub-machine, as in figure 2.15, there are three shared variables and one private variable. We notice that, in addition to the three chosen events, the other events are also appearing in this sub-component as external events. This is due to the use of shared variables.

```
MACHINE
  train_0
SEES
  Context_train_0
VARIABLES
  resrt      //   Shared variable, DO NOT REFINE
  resbl      //   Shared variable, DO NOT REFINE
  OCC       //   Private variable
  rsrtbl      //   Shared variable, DO NOT REFINE
INVARIANTS
  typing_resrt   :   resrt ∈ ℙ(R)
  typing_resbl   :   resbl ∈ ℙ(B)
  typing_OCC   :   OCC ∈ ℙ(B)
  typing_rsrtbl   :   rsrtbl ∈ ℙ(B × R)
EVENTS
  INITIALISATION   ≙
  STATUS
   ordinary
  BEGIN
   act1   :   resrt ≔ ∅
   act2   :   resbl ≔ ∅
   act3   :   rsrtbl ≔ ∅
   act4   :   OCC ≔ ∅
  END

  route_reservation   ≙        //   External event, DO NOT REFINE
  STATUS
   ordinary
  ANY
   r
  WHERE
   grd1   :   r ∉ resrt
   grd2   :   rtbl~[{r}]∩resbl = ∅
  THEN
   act1   :   resrt ≔ resrt∪{r}
   act2   :   rsrtbl ≔ rsrtbl ∪ (rtbl⩥{r})
   act3   :   resbl ≔ resbl ∪ rtbl~[{r}]
  END

  route_freeing   ≙        //   External event, DO NOT REFINE
  STATUS
   ordinary
  ANY
   r
  WHERE
   grd1   :   r ∈ resrt\ran(rsrtbl)
  THEN
   act1   :   resrt ≔ resrt\{r}
  END

  FRONT_MOVE_1   ≙
  STATUS
   ordinary
  ANY
   r
  WHERE
   grd1   :   r ∈ resrt
   grd2   :   fst(r) ∈ resbl\OCC
   grd3   :   rsrtbl(fst(r)) = r
  THEN
   act1   :   OCC ≔ OCC ∪ {fst(r)}
  END

  FRONT_MOVE_2   ≙
  STATUS
   ordinary
  ANY
   b
   c
  WHERE
   grd1   :   b∈OCC
   grd2   :   c∉OCC
   grd3   :   b↦c ∈ nxt(rsrtbl(b))
  THEN
   act1   :   OCC ≔ OCC∪{c}
  END

  BACK_MOVE   ≙
  STATUS
   ordinary
  ANY
   b
   n
  WHERE
   grd1   :   b∈OCC
   grd2   :   n = nxt(rsrtbl(b))
   grd3   :   b∈dom(n) ⇒ n(b)∉OCC
   grd4   :   b∈ran(n) ∧
              n~(b)∈dom(rsrtbl)
              ⇒
              rsrtbl(n~(b))≠rsrtbl(b)
  THEN
   act1   :   OCC ≔ OCC\{b}
   act2   :   rsrtbl ≔ {b}⩤rsrtbl
   act3   :   resbl ≔ resbl\{b}
  END

END
```

Figure 2.15: Train System Case Study: Train Sub-machine

For the other sub-machine Track, as in figure 2.16 it defines all the variables as shared, because there is any private variable. In addition, it contains the $BACK_MOVE$ event as an external event.

```
MACHINE
  train_0
SEES
  Context_train_0
VARIABLES
  resrt      //   Shared variable, DO NOT REFINE
  resbl      //   Shared variable, DO NOT REFINE
  rsrtbl      //   Shared variable, DO NOT REFINE
INVARIANTS
  typing_resrt   :   resrt ∈ ℙ(R)
  typing_resbl   :   resbl ∈ ℙ(B)
  typing_rsrtbl   :   rsrtbl ∈ ℙ(B × R)
EVENTS
  INITIALISATION   ≙
  STATUS
   ordinary
  BEGIN
   act1   :   resrt ≔ ∅
   act2   :   resbl ≔ ∅
   act3   :   rsrtbl ≔ ∅
  END

  route_reservation   ≙
  STATUS
   ordinary
  ANY
   r
  WHERE
   grd1   :   r ∉ resrt
   grd2   :   rtbl~[{r}]∩resbl = ∅
  THEN
   act1   :   resrt ≔ resrt∪{r}
   act2   :   rsrtbl ≔ rsrtbl ∪ (rtbl⩥{r})
   act3   :   resbl ≔ resbl ∪ rtbl~[{r}]
  END

  route_freeing   ≙
  STATUS
   ordinary
  ANY
   r
  WHERE
   grd1   :   r ∈ resrt\ran(rsrtbl)
  THEN
   act1   :   resrt ≔ resrt\{r}
  END

  BACK_MOVE   ≙         //   External event, DO NOT REFINE
  STATUS
   ordinary
  ANY
   b
   n
   OCC
  WHERE
   typing_OCC   :   OCC ∈ ℙ(B)
   grd1   :   b∈OCC
   grd2   :   n = nxt(rsrtbl(b))
   grd3   :   b∈dom(n) ⇒ n(b)∈OCC
   grd4   :   b∈ran(n) ∧
              n~(b)∈dom(rsrtbl)
              ⇒
              rsrtbl(n~(b))≠rsrtbl(b)
  THEN
   act2   :   rsrtbl ≔ {b}⩤rsrtbl
   act3   :   resbl ≔ resbl\{b}
  END

END
```

Figure 2.16: Train System Case Study: Track Sub-machine

2.3.2 Decomposition by Shared Events

The shared event decomposition is an evolution of event atomicity decomposition. The author in [Butler, 2009a] proposes this method to separate the variables of a system into two different sub-machines by decomposing a shared event. In fact, this approach allows to get sub-components that interact in parallel through synchronised events. This approach is suitable for distributed system development [Butler, 1997].

To process a decomposition by shared event, the authors present in [Silva et al., 2011] the required steps to follow. First, the sub-components to generate have to be defined. After that, the variables should be split over the sub-components. As a consequence, the rest of the model components, such as events, invariants, contexts, etc., are partitioned on the basis of the defined variables allocation. For the invariants, they are decomposed regarding the variables scope. These invariants must include at least the definition of the variables typing. These are the required invariants in order to get a valid refinement. Additional invariants depend on the need of the user. Also, they may be defined for further refinements or to help the sub-components reuse. Concerning the invariants that are using the other sub-components variables, an additional refinement may be necessary to explicitly split the variables in question. For the event decomposition, it depends on the variables partition. Actually, the resulting events after decomposition are maintained and some new events appear which are interfaces of the original events. These interfaces are preserving the parts corresponding to the variables that belong to each sub-component.

As illustration, let M_0 be the machine as in figure 2.17. Variables *v1* and *v2* of this machine are partitioned respectively in two sub-machines M_{1a} and M_{1b}. *event2* is decomposed in the two sub-components as two events *event2'* and *event2"*, each event describes the change of state applied to *v1* and *v2* respectively.

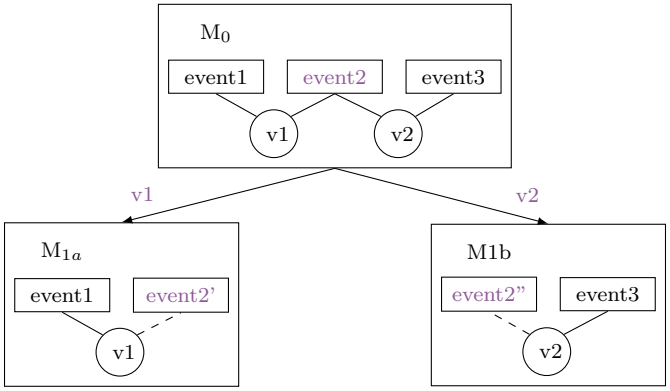


Figure 2.17: Decomposition by Shared Event

To decompose a machine using this method, variables to partition in each sub-component are chosen and then the decomposition is applied. Generated machines contain the selected variables, and shared events are defined in two different signatures for each sub-machine. These events describe the variables changes. In this approach, events are shared between sub-components and variable sharing is not allowed which is considered as a restriction of this method. This approach

is explained in detail, in section 3.3, with its application on the case study example, as well as the presentation of its limitations.

2.3.3 Other Methods of Decomposition in Event-B

Instantiation

In addition to refinement and decomposition by shared variables, generic instantiation is another proposition of Abrial in [Abrial and Hallerstede, 2007]. It is based on the reuse of the abstract model with slight modifications by instantiating sets and constants of this model. In [Hoang et al., 2011], the modularisation is another proposition based on defining interfaces in B-Method. This approach promotes the use of *USES* clause in order to call operations. *Fragmentation and distribution* approach in [Siala et al., 2016] defines a specification using DSL (Domain Specific Language) [Van Deursen et al., 2000] to decompose a model. In the same context, [Hoang et al., 2017] propose also a technique based on the use of a *classical-B* clause. This approach proposes the use of a composition mechanism based on the use of *INCLUDES* clause. So, the including machine can use variables and invariants of the included machine.

Modularisation

Modularisation is a conservative extension of the Event-B formalism proposed by [Hoang et al., 2011]. This is a special case of shared variables decomposition [Abrial and Hallerstede, 2007]. Modularisation allows: the decomposition of the models of the system into sub-components that can be developed easily, managing the complexity of models and the reuse of formally developed components using the clause USES. This method is based on the definition of sub-components called interfaces that contain variables and operations. These operations are specified by a pair of pre/post conditions. The interface is integrated in the refinement of the abstract machine M by the clause USES.

Fragmentation and Distribution

Another technique proposed by [Siala et al., 2016] that takes as input any Event-B model and generates a refinement of the machine in question from a specification based on two stages: fragmentation and distribution. Fragmentation aims to reduce the non-determinism of calculating the local parameters of an event. The order of the parameter calculation is described by a specifier using a Domain Specific Language (DSL), a language whose specifications are designed to meet the constraints of a specific application domain. Based on this specification and the abstract Event-B model, the fragmentation stage generates an Event-B model. This automatic refinement relies on simple rules ensuring a refinement that defines new variables, new events and refining events by reinforcing guards and invariants.

After the fragmentation step, the distribution step takes as input an abstract model and a distribution specification. This specification introduces the selected configuration: the names of the sub-components. Similarly, it distributes the variables and possibly the guards on the sub-components. The referenced variables by a guard must be located on the same sub-component. Otherwise, poorly consistent copies of its variables will automatically be added. They are updated

by convergent scheduled events before the event accessing the copies. Similarly, an action is performed by the (supposedly unique) component on which modified variables are located. The visible variables by an action can be remote. The values of these variables will be transmitted during synchronisation.

2.4 Synthesis

As a formal method, Event-B allows the observation of a system events and the validation of its properties using mathematics rules and different types of lemmas called proof obligations. For each step of modelling, some proof obligations should be discharged. When all the proof obligations are discharged the model is considered as correct. However, the formal method Event-B lacks the modularity aspect in its syntax and semantic. The user does not have the flexibility to manage its models according to each sub-component use.

In fact, several methods of decomposition and modularity exists. Some are based on the shared variables or events and others are based on a kind of interfaces or even the use of other language in order to enrich the approach. So, in the next chapter, we present a study and an analysis of these existing approaches.

Part II

Contributions: from Systems to Sub-systems Modelling in Event-B

CHAPTER 3

ANALYSIS AND DISCUSSION: A-STYLE AND B-STYLE

Contents

Introduction

Formal modelling and verification of safety-critical systems using formal methods is very relevant for the many reasons exposed in the first part of this manuscript, but these can be tedious without modular design and mechanisms. Indeed, starting from the modelling and verification of the whole system requirements specification can be time and resources consuming. The study of the different modularisation mechanisms in the literature leads us to analyse the decomposition by shared variables and the decomposition by shared events. The associated plugins in the Rodin platform are used to apply these approaches on some simple examples. This is presented in the first section of this chapter. By analysing and studying the examples, we have identified some limitations of these approaches which we cope to them in our proposed approach. In the second part of this chapter, we detail our concrete railway case study to which the existing and the proposed approaches are applied.

Notice that this chapter deals with "**Why we propose a new approach of decomposition?**" after analysing the existing decomposition approaches, whereas the next chapter deals with "**How we decompose?**".

The analysis of the existing approaches is motivated by the industrial practice who stress in the complexity of the formal modelling verification of the whole system. They are interested in breaking down systems to subsystems enabling the "distribution" of the requirements specification into subsystems requirements specifications. In this work, we do not take into account preliminary requirements engineering method before the formal modelling and verification of the subsystems specification [Tueno Fotso, 2019]. Actually, our work is based on the formal reasoning of the system and subsystems specifications in a straightforward way. So, we focus on ensuring that a requirement assigned to a subsystem is well preserved by the subsystem model. Therefore, a particular accentuation putting forward consists on how preserving invariants in subsystems specifications so as to preserve the whole distributed behaviours.

The main aim of the model decomposition techniques is to reduce the complexity of large models and increase their modularity. These techniques consist in dividing a model into sub-models that can be refined separately and more easily than the original one. But they decompose the initial model by shared variables or by shared events. The shared variables approach is suitable for designing parallel computing programs, whereas the shared events decomposition is suitable for developing message-passing for distributed systems [Tueno Fotso, 2019]. The question that arises here is "What about a model decomposition of a system that requires these two purposes: parallel computing programs and message-passing distributed systems?". Indeed, railway systems involves each of them: separating different railway signalling functionalities in different subsystems such as release and integrity for the first purpose, and separating the distributed subsystems such as on-board and track-side systems for the second one. Our goal is to propose a double-edged approach which focuses on distributing system into subsystems by sharing variables and sharing events at the same time if necessary. Also, this approach relies on the preservation of the syntactic/semantic coherence from the beginning to the lower level of modelling.

In the following, the analysis of these approaches is performed on some simple examples in a first step to identify some trivial limitations and on a railway case study in a second step. The case

study specification and model are presented in the first section. In the second section, an analysis of the decomposition by shared variables is made by the application of the associated plugin on a simple example then on the case study. The same process is followed in the next section for the decomposition by shared events: application of the B-style plugin on a simple example then on the case study. Finally, we discuss the results and present the limitations of these approaches according to our decomposition goal.

3.1 Modelling of a Railway Case Study

In order to analyse the existing approaches and illustrate our contribution, we choose a case study that illustrates one of the typical dangers of trains circulations cited above in section 1.2.2: the Rear-End Collision. It is a real case of train circulation danger but not the only one. So, we have modelled and formally proved the case study of railway signalling systems on *Atelier B*[1] (version 4.5 beta 12) and *Rodin* (version 2.7) tools. This case study is a simple example that focuses on a particular railway network requirement and it contains relevant elements to the decomposition analysis. In addition, this example is representative of what is done in the industrial railway field and in sub-systems traffic management such as the *European Railway Traffic Management System* (*ERTMS*). The specification of the case study was validated by Clearsy following the industrial need.

This case study is not a full-fledged industrial railway model. Indeed, it doesn't take into consideration some components of the track-side such as points, level-crossing, entire lateral signalisation, etc. and other components of the train-side as the train integrity, position, delays, etc. The goal is to use this simplified case study for a first analysis and application of the various decomposition approaches.

3.1.1 Informal Specification

The aim is to model a system which allows the trains control, in other words a system that ensures a safe train circulation in a certain railway network. The main goal of this case study is to avoid trains rear-end collisions as in figure 3.1.

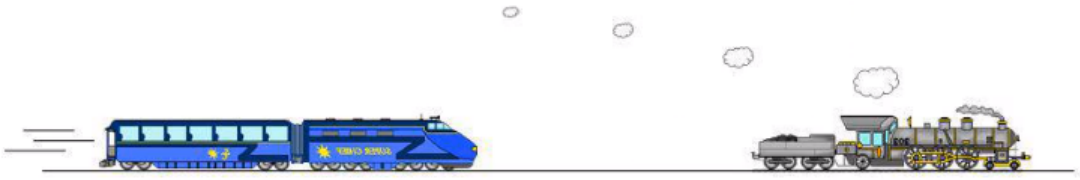


Figure 3.1: Example of a Train Rear-End Collision

In a one-way traffic split into blocks, let consider two trains *Train A* and *Train B* moving by a certain distance (number of steps). *Train A* is following *Train B*. The trains movements are based on the position of the front/head and of the end/tail of each train. Each block should be occupied at most by one train. When the front of the train enters a block, this block is turned to occupied. When the end of the train leaves the block, this later is turned to free. So, a train enters only a free block.

[1] *Atelier B* tool: https://www.atelierb.eu/

3.1.2 Abstraction

In a first step of modelling, we define the trains movements based in their positions on the traffic as shown in figure 3.2. At this level, we don't consider the existence of the blocks. To avoid a rear-end collision, the position of the front of *Train A* should not be at the same position of the end of *Train B* or after it.

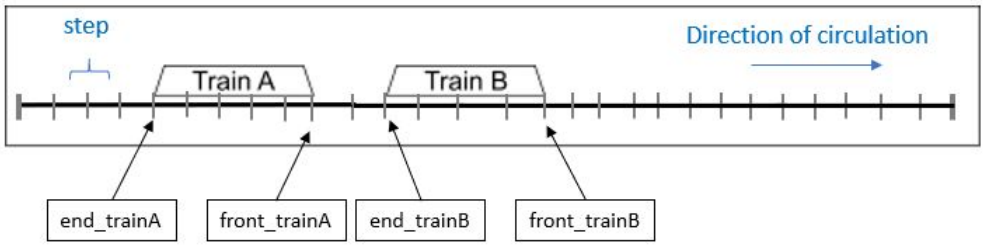


Figure 3.2: Case Study: Abstract Machine Description

The **abstract machine** M_0 defines, in figure 3.3, the variables describing the trains front and the trains end positions are respectively: *front_trainA, front_trainB, end_trainA* and *end_trainB*.

M_0 defines the invariant that must be preserved: the position of the front of *Train A* must always be following the position of the end of *Train B*. This is specified by this expression: $front\_trainA < end\_trainB$.

M_0 defines also the events that describe the trains movements by a certain number of steps, as shown in figure 3.3:

- move_front_trainA: changes the position of the front of *Train A* without catching up the next train. This is done through the action $act1$ such as
  $front\_trainA := front\_trainA + step$
- move_end_trainA: changes the *Train A* end position taking into consideration the position of its front.
- move_front_trainB: changes the *Train B* front position.
- move_end_trainB: changes the position of the *Train B* end taking into consideration its front position.

MACHINE M0
VARIABLES
front_trainA // position of the front of tainA
end_trainA // position of the end of tainA
front_trainB // position of the front of tainB
end_trainB // position of the end of tainB
INVARIANTS
inv5: $end\_trainA < front\_trainA$ // for each train, the position of the train end must be lower than the position of the train front
inv6: $end\_trainB < front\_trainB$
inv7: $front\_trainA < end\_trainB$ // since trainA is following trainB, the position of trainA front must be lower than the position of trainB end
EVENTS
Event move_front_trainA ⟨ordinary⟩ $\widehat{=}$ // movement of the front of trainA
any
step
where
grd1: $step \in \mathbb{N}_1$
grd2: $front\_trainA + step < end\_trainB$
then
act1: $front\_trainA := front\_trainA + step$ // change the position of the front of *Train A* to the new position.
end
END

Figure 3.3: Case Study: Excerpt of the Abstract Machine M_0

3.1.3 Refinement

In a second phase, we define a more concrete machine introducing the blocks notation and the trains movements on the blocks. The traffic is now partitioned into several blocks as in figure 3.4. Each block is defined by the coordinates of their front and end. The front of each block has the same coordinate as that of the end of the next block. A block can be occupied or free. Each train can occupy one block or more.

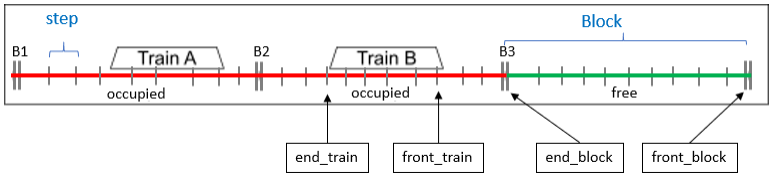


Figure 3.4: Case Study: Refinement Description

We define a **Context** C_1, as in figure 3.5, to describe the static part of the refinement.

CONTEXT C1
SETS
BlockState // a bloc state can be free or occupied
SUBSYS // track or train
CONSTANTS
Block
front_block // the front limit of the block
end_block // the rear limit of the block
next_block //the function defininf the next for each bloc
...
AXIOMS
axm1: $BlockState = \{Free, Occupied\}$
...
axm7:
$\forall bk \cdot ($
$bk \in Block$
$\Rightarrow end\_block(bk) < front\_block(bk)$
) // the rear limit of a block is lower than its front limit
axm8: $\forall b1, b2 \cdot (b1 \in Block \wedge b2 \in Block \wedge next\_block(b1) = b2 \Rightarrow front\_block(b1) = end\_block(b2))$
// the front limit of a block is equal to the rear limit of the next block
axm9: $\langle theorem \rangle\ ran(next\_block) = Block \setminus \{0\}$
END

Figure 3.5: Case Study: Excerpt of the Blocks Context C_1

C_1 specifies the blocks and some track properties (axioms). It defines $front\_block$, end_block and $next\_block$ as constants since they don't change. Two block states are possible: $Free$ or

Occupied, this is defined by *BlockState* (figure 3.5).

In the **refinement machine** M_1, as in figure 3.6, the variables that describe the change of the blocks states by *block_state* are defined. In addition, it defines new variables that allow the identification of the occupied blocks by each train. For example, in lines 4 in figure 3.6, the variables *fst_tAblock* and *lst_tAblock* respectively describe the first block occupied by the front of *Train A* and the last block occupied by the end of *Train A*. In the same way are defined *fst_tBblock* an *lst_tBblock* for *Train B*.

MACHINE M1
REFINES M0
SEES C1
VARIABLES
...
fst_tAblock // the block occupied by the head of trainA
lst_tAblock // the block occupied by the end of trainA
block_state // a function that gives the state of each block
next_turn // this variable allows to identify which events should be triggered: the trains events or the track events
INVARIANTS
inv11: $\forall bk \cdot (bk \in Block \land next\_turn = TRAIN \land block\_state(bk) = Free \Rightarrow bk \neq lst\_tBblock)$ // when it is the turn of the trains movements, trainB shouldn't be on the released blocks behind it
EVENTS
Event TRACKevent ⟨ordinary⟩ $\widehat{=}$ // block state changes
when
grd1: $next\_turn = TRACK$
then
act1: $next\_turn := TRAIN$
act2: $block\_state := (Block \times \{Free\}) \lhd ((lst\_tAblock \mathinner{..} fst\_tAblock \cup lst\_tBblock \mathinner{..} fst\_tBblock) \times \{Occupied\})$ // the block state is surcharged with the new states
end
END

Figure 3.6: Case Study: Excerpt of the First Refinement Machine M_1

Another new variable is defined, the variable *next_turn*. It allows the transition from the *Track* behaviour to the *Train* behaviour and vice versa.

A block can be occupied at most by one train, in other words the occupied block by the end of *Train B* named *lst_tBblock* should always be in front of the occupied block by the front of *Train A* named *fst_tAblock* as defined in the invariant: $fst\_tAblock < lst\_tBblock$.
Another useful invariant is defined in order to ensure the distance between *Train A* and *Train B*:
∀ bk.(bk:Block & next_turn = TRAIN & block_state(bk) = Free ⇒ bk ≠ lst_tBblock)

M_1 defines also the events describing the trains movement in a traffic portioned into blocks. It describes the events refining the abstract ones in M_0 and other new events:

- enter_tAblock: describes the occupation of a block by the front of *Train A*. This event is refining move_front_trainA of M_0.
- enter_tBblock: describes the occupation of a block by the front of *Train B*. This event is refining move_front_trainB of M_0.
- free_tAblock: describes the behaviour of the end of *Train A* when it leaves the block. This event is refining move_end_trainA of M_0.
- free_tBblock: describes the behaviour of the end of *Train B* when it leaves the block. This event is refining move_end_trainB of M_0.
- TRACKevent: is a new event that changes the block state after the trains movement, as shown in the excerpt of figure 3.6.

After this first step of refinement, we define a more concrete machine introducing the signals notation and the trains movements regarding the signals states. Now, the traffic contains also signals. Each signal is associated to a block. A signal can be red or green following the state of the associated block. If the block is occupied, its signal is red, and if it is free its associated signal is green.

So, we define a **Context** C_2 to describe the static part of the second refinement. As in figure 3.7, the context C_2 specifies the signals and some of their properties (axioms).

CONTEXT C2
EXTENDS C1
SETS
SignalState // a signal can be red or green
CONSTANTS
Signal
signal_block // each signal is associated to one block
red
green
AXIOMS
axm1: $SignalState = \{red, green\}$
axm2: $Signal = \mathbb{N}$
axm3: $signal\_block = (\lambda bk \cdot bk \in Block | bk)$ // each signal is associated to one block: signal $i \Leftrightarrow$ block i
END

Figure 3.7: Case Study: Excerpt of the Signals Context C_2

It defines the set of *SignalState* which is an enumeration of *red* and *green*. It defines also the constant *signal_block* that associates each block with its corresponding signal.

In the **refinement machine** M_2, as in figure 3.8, the variable that describes the change of the signals states by *signal_state* is defined. A safety property of signals is expressed through a new invariant. This property ensures that a signal becomes green when its associated block is free such as: $\forall\, bk(bk \in Block \wedge next\_turn = TRAIN \wedge signal\_state(signal\_block(bk)) = green \implies block\_state(bk) = Free)$

MACHINE M2
REFINES M1
SEES Signal
VARIABLES
signal_state // variable of the signal state
...
INVARIANTS
inv1: $signal\_state \in Signal \rightarrow SignalState$
inv2:
$\forall bk \cdot ($
$bk \in Block$
$\wedge\, next\_turn = TRAIN$
$\wedge\, signal\_state(signal\_block(bk)) = green$
$\Rightarrow block\_state(bk) = Free)$ // each signal can be red or green // a signal becomes green if its corresponding block is free
EVENTS
...
Event TRACKevent ⟨ordinary⟩ $\widehat{=}$ // block and signals states changes
refines TRACKevent
when
grd1: $next\_turn = TRACK$
then
act1: $next\_turn := TRAIN$
act2: $block\_state := (Block \times \{Free\}) \lhd ((lst\_tAblock..fst\_tAblock \cup lst\_tBblock..fst\_tBblock) \times \{Occupied\})$
act3: $signal\_state := (Signal \times \{green\}) \lhd ((signal\_block(lst\_tAblock) .. signal\_block(fst\_tAblock) \cup signal\_block(lst\_tBblock) .. signal\_block(fst\_tBblock)) \times \{red\})$ // signals states changes
end
END

Figure 3.8: Case Study: Excerpt of the Second Refinement Machine M_2

So, to describe the model structure of this case study, the machine M_1 is refining M_0 and seeing the context C_1, and the machine M_2 is refining M_1 and seeing the context C_2, as in figure 3.9. The context C_2 is an extension of the context C_1.

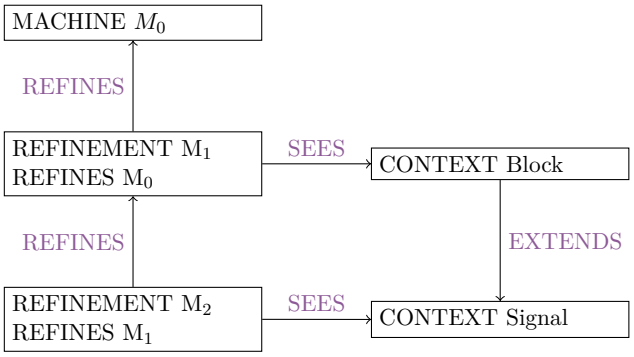


Figure 3.9: Structure of the Case Study Model

3.1.4 Proof and Animation of the Model

Using *ProB*[2] [Leuschel and Butler, 2003], an animation is elaborated on the model. This animation is done using VisB (see Annexe A.2.3). Figure 3.10 shows an example of a possible scenario of trains movements. In a first step, each of *Train A* and *Train B* occupy distinct blocks. Then, in step 2 *Train B* moves to the next block and occupies it. The blue train shadow presents the previous train position. As a third step, while *Train A* is moving inside the associated block, *Train B* releases the previous block. Hence, in step four *Train A* enters the next free block.

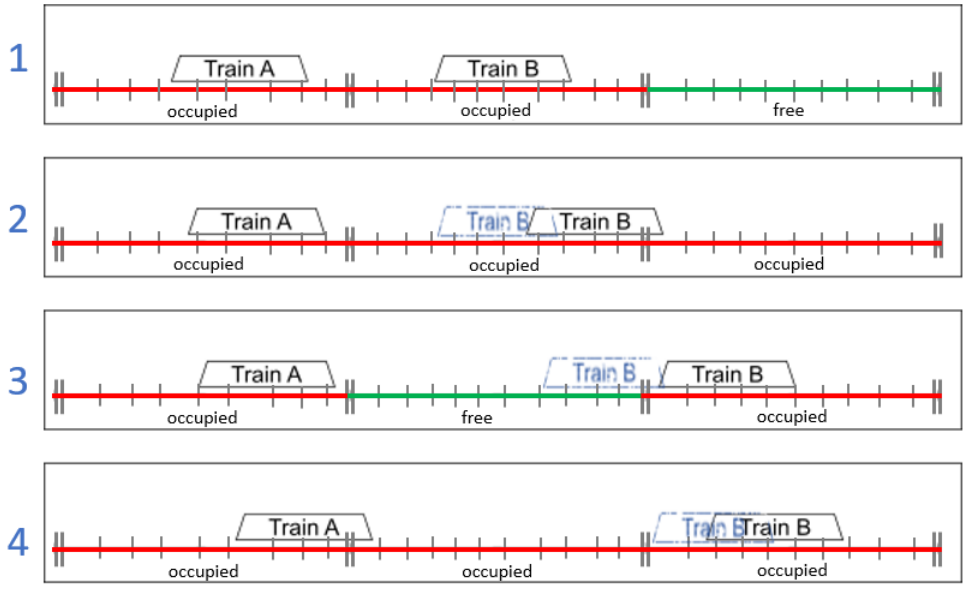


Figure 3.10: Example of Trains Movement Scenario

[2] *ProB: www3.hhu.de/stups/prob/index.php/Main_Page*

3.2 Decomposition by Shared Variables

3.2.1 Application of the A-style Plugin

The shared variables decomposition consists on distributing events of a machine between several sub-machines. This approach proposes to manage shared variables between several events. During the machine decomposition, events to be separated are selected in each sub-machine and considered as internal events. A variable that occurs only in the internal events is a private variable. If a variable is involved in internal events of different sub-machines, it is defined in each of them as a shared variable that cannot be refined. External events of a sub-machine are events that simulate the change of state of the shared variables in the abstract machine. The description of this approach is detailed in section 2.3.1.

According to [Abrial, 2009], the shared variables decomposition is motivated by the interest of the refinements which can be done independently after decomposition. This kind of decomposition performs events partition that leads to a certain variables distribution. When the events partition induces disjoint variables distribution, the decomposition is considered as trivial. However, in the case of sharing variables between events, some difficulties in the decomposition can appear since the splitting of the events will be conflicted with the presence of a common variable in events partitioned in different sub-machines. It is a problem of synchronisation of the communication channel which is specified through this shared variable.

As the shared variable exists in the different resulting sub-machines, it can be read by one or multiple sub-machines and also written by one or multiple sub-machines. If only one sub-machine writes on (change the state of) this shared variable, while this variable is read by the others, it still possible and normal. Nevertheless, when two or more sub-machines change the state of the shared variable, the difficulty arises because of the refinements that can be performed after decomposition. The shared variable can be data-refined using different refinement strategy and reasoning. For this reason, Abrial considers shared variables as not data-refined variables and proposes a partial overcome of this limitation by data-refining them in the same way in each independent sub-machine. This can be conflicting with the initial purpose of the decomposition which is modelling of sub-components independently. Hence the notion of external events is defined by Abrial, as we presented in the first chapter. These external events have a partial simulation of the shared variable which corresponds to the state changes of the non-decomposed model which modifies the shared variable in question. These external variables cannot be refined in turn.

Besides the partition of the events and the variables distribution, one must pay attention on how to manage invariants in this approach. Three types of invariants handling are possible:

1. Invariant involving only private variables is copied in its sub-machine;
2. Invariant involving only the shared variable is copied in the sub-machines where this variable is shared;
3. Invariant involving a shared variable and other variables together is not copied.

The last variant of invariant handling goes against the syntactic/semantic coherence that we want to guarantee in order to preserve of the whole behaviour of the system from the beginning

model to the lower-level models.

In practice, this decomposition may be less relevant when the model to be decomposed contains a large number of shared variables, especially in case of decomposing complex refinements rich with shared variables [Silva et al., 2011]. Furthermore, there exists a restriction of this method: shared variables and external events must be present in the resulting sub-components and cannot be refined when refining these sub-components [Abrial, 2009]. For these reasons, it may be necessary to proceed with an intermediate preparation step which consists in a manual refinement. This step can reduce the complexity of predicates such as invariants, guards and axioms, as well as substitutions (actions) by separating the variables assigned to different sub-components [Abrial, 2009]. The user must explicitly separate the variables in this refinement by introducing an auxiliary parameter p. For example, the predicate $v1 = v2$ becomes $p = v2 \& v1 = p$. If this manual refinement step is not performed, the complex predicates and substitutions are automatically marked by the tool via a message frame and then the user intervention is required to explicitly perform the separation.

Application on the case study

Our goal is to decompose the system into two sub-systems: Train and Track. The first sub-system, Train, allows the observation of the train movement. In other words, this sub-system will describe the evolution of the train components states such as the position of the front and the end of each train. The second sub-system, Track, gives the observation of the track changes, such as the block and the signals states. In other words, it gives the block state changes (free or occupied) and the signal state changes (red or green).

These two sub-systems are going to be synchronised through the variable *next_turn*. In fact, for each movement of each train, the track states are updated. So, when $next\_turn = train$, it is the turn of the $Train$ sub-system. This means that a train can leave a specific block for example. Then $next\_turn :=$ *(becomes equal to)* $track$, which means that it comes the $Track$ sub-system turn to change the block state to free. After that, the variable *next_turn* changes the state to *train* and so on.

To be more concrete and realistic, lets apply this approach on the modelled case study by following this approach steps:

1. Choose the number of the sub-machines: our goal is to obtain a train behaviour in one side and the track behaviour in the other side. So we need to obtain two sub-machines: Train and Track.

2. After the identification of the needed behaviour in each sub-machine, the events can be partitioned into each sub-machine. The Train sub-machine must define all the train movements: the events that allow the movement of the head and the tail of each train. The Track machine defines the event that allow to modify the state of the infrastructure: blocks and signals.

3. Identify the shared variables. the shared variables can be categorised into two types: variables that are shared for reading and those shared for writing. The first type means that a machine X can need to have a visibility on a local variable a, of another machine Y, in its guards to

update the state of its local variables. The second type means that the state of the variable can be updated by both sub-machines. In this decomposition, the concerned shared variables are those shared for writing. So, in the model of the case study, *next_turn* is a shared variable between the sub-machines Train and Track.

4. Proceed with an additional step of refinement to manage the shared variable and make the decomposition possible. Following [Abrial, 2009], we define two new variables *ww* and *tt* in a refinement machine (see figure 3.11).

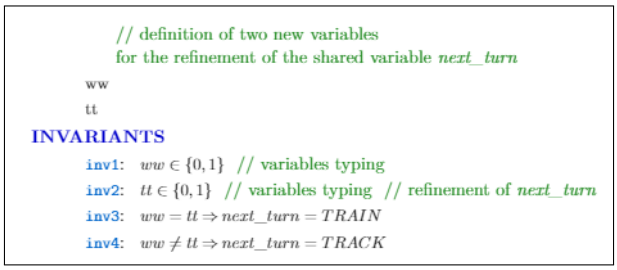


Figure 3.11: New Additional Variables to Decompose using A-style

These variables are refining *next_turn* by the invariants:

$$ww = tt \Rightarrow next\_turn = TRAIN$$

$$ww \neq tt \Rightarrow next\_turn = TRACK$$

In addition, all the events involving *next_turn* are refined taking into account the new defined variables (see figure 3.12).

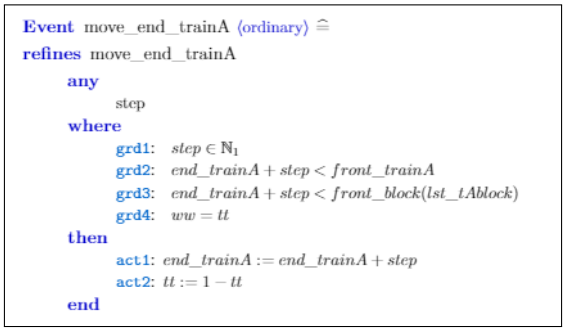


Figure 3.12: Example of an Event Refinement before the Decomposition

5. Apply the plugin by giving the number and the names of the sub-machines, as well as the chosen events in each machine.

After the application of the plugin, we obtain two distinct machines Train and Track. Each of them defines the chosen events. For the variables, we notice that some are mentioned by *do not refine*. So the refinement after this step of decomposition becomes more complex. In addition, we notice the apparition of some local variables in the external events. In figure 3.13, the variable $front\_trainA$ is local in the external event $move\_front\_trainA$.

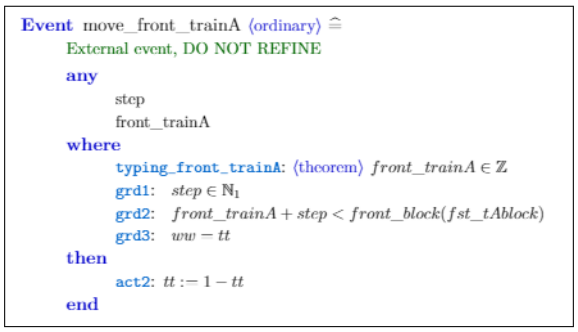


Figure 3.13: Example of an External Event

3.2.2 Synthesis

For shared variables decomposition, event partitioning is always possible in order to generate sub-components. However, this decomposition may be less important despite its potential: a large number of shared variables may not be of much interest, particularly for refinements that become more complex [Silva et al., 2011]. Due to the restriction of shared variables, it may be necessary to proceed with preparatory steps of refinement to resolve complex predicates (invariants, guards, axioms) or substitutions (actions) by separating the variables assigned to different sub-components. If this step is not performed, these complex predicates/assignments are automatically marked by the tool and the user intervention is required.

3.3 Decomposition by Shared Events

3.3.1 Application of the B-style Plugin

Let apply the decomposition by the shared event on an abstract machine M1 with two variables v1 and v2. This machine is decomposed into two sub-machines M2a and M2b where v1 belongs to M2a and v2 to M2b. We note here that not all actions are accepted to be decomposed and variables partitioning is not always possible. Table 3.1 shows different types of actions that make variables states evolve. The variables v1 and v2 are two case study variables.

Actions types of M_1		$M_{2a}(v1)$	$M_{2b}(v2)$
act1	v1 :| (v1=1) ‖ v2 :| (v2=2)	v1 :| (v1=1)	v2 :| (v2=2)
act2	v1,v2 :| (v1=2 ∧ v2=v1+2)	–	–
act3	v1,v2 := 1,2	v1 := 1	v2 := 2
act4	v1 := v2+1	–	–
act5	v1 :: {v2,1,2}	–	–

Table 3.1: Application of the Shared Event Decomposition.

When applying the decomposition, an error message is displayed asking to simplify some actions: the assignment is too complex because it refers to elements belonging to different sub-components.

The obtained errors are shown in table 3.1 by this symbol '–'. Predicates (invariants and guards) and actions should not refer to variables that must be partitioned into different sub-components. As an example, the substitution becomes such that in act2 cannot be decomposed: $v1, v2 :| (v1 = 2 \wedge v2 = v1 + 2)$.

This problem can be solved by adding an additional refinement step before proceeding with the decomposition. The user must explicitly separate the variables by introducing an auxiliary parameter p, such as $v1 = v2 <=> p = v2 = v1 = p$).

The partition of variables is still possible for all models. We apply the event split decomposition plugin on a machine M with two variables v1 and v2 and an event containing actions on these variables. During the decomposition, we find that the actions are not all accepted to be decomposed. An error message is displayed asking to simplify these actions.

Now, we consider a more realistic example. It seems to be a toy example, but it is a simple way to understand the faced difficulties for the decomposition. In this example, we represent train movements. A train can only move if the signal is green and the next block is free. So, we have a context that describes the static part: train or track turn (see figure 3.14).

CONTEXT C
SETS
SUBSYS
CONSTANTS
train
track
AXIOMS
axm1: $SUBSYS = \{train, track\}$
END

Figure 3.14: Context of Example 2

The dynamic part is modelled in a machine where are described the variables concerning the signal state, the block state, the movement authorisation and train movement authorisation. Besides that, a safety property is expressed by an invariant: the train must move after verifying that the signal is green ant the block to enter in free (see figure 3.15).

Therefore, we apply the decomposition by shared events by partitioning the variables between the Track sub-machine and the Train sub-machine.

3.3.2 Synthesis

For shared events decomposition, predicates (invariants, guards) and assignments (actions) should not refer to elements that must be partitioned into different sub-components. If we create the sub-component M1 with the element v1 and the sub-component M2 with the element v2 of the machine M, then a predicate P (v1, v2) in M, will produce this error by decomposing: "*The assignment is too complex because it refers to elements belonging to different sub-components*".

MACHINE Rail
REFSEES C
VARIABLES
green
occupied
move
next_turn
INVARIANTS
inv1: $green \in BOOL$
inv2: $occupied \in BOOL$
inv3: $move \in BOOL$
inv4: $move = TRUE \Rightarrow (green = TRUE \land occupied = FALSE)$
inv5: $next\_turn \in \{train, track\}$
EVENTS
Initialisation
begin
act1: $green := FALSE$
act2: $occupied := FALSE$
act3: $move := FALSE$
act4: $next\_turn := train$
end
Event train_move ⟨ordinary⟩ $\widehat{=}$
when
grd1: $green = TRUE$
grd2: $occupied = FALSE$
grd3: $move = FALSE$
grd4: $next\_turn = train$
then
act3: $move := TRUE$
end
END

Figure 3.15: Machine of Example 2

MACHINE Track
REFSEES C
VARIABLES
green
occupied
INVARIANTS
typing_green: ⟨theorem⟩ $green \in BOOL$
typing_occupied: ⟨theorem⟩ $occupied \in BOOL$
trsb_inv1: $green \in BOOL$
trsb_inv2: $occupied \in BOOL$
EVENTS
Initialisation
begin
act1: $green := FALSE$
act2: $occupied := FALSE$
end
Event train_move ⟨ordinary⟩ $\widehat{=}$
when
grd1: $green = TRUE$
grd2: $occupied = FALSE$
then
skip
end
END

Figure 3.16: First Sub-machine of Example 2

MACHINE Train
REFSEES C
VARIABLES
move
next_turn
INVARIANTS
typing_move: ⟨theorem⟩ $move \in BOOL$
typing_next_turn: ⟨theorem⟩ $next\_turn \in SUBSYS$
trsb_inv3: $move \in BOOL$
trsb_inv5: $next\_turn \in \{train, track\}$
EVENTS
Initialisation
begin
act3: $move := FALSE$
act4: $next\_turn := train$
end
Event train_move ⟨ordinary⟩ $\widehat{=}$
when
grd3: $move = FALSE$
grd4: $next\_turn = train$
then
act3: $move := TRUE$
end
END

Figure 3.17: Second Sub-machine of Example 2

This problem can be solved by passing an additional refinement step before decomposition. The user must explicitly separate the elements by this refinement by introducing an auxiliary parameter p representing the value of a variable. (Example: $v1 = v2 <=> p = v2^v1 = p$).

As far as the errors of the action act2 are concerned, no solution is proposed. The shared events decomposition Plugin is applied by partitioning the variables on two machines M1 and M2. This decomposition makes it possible to identify some limitations and inconsistencies in the behaviour of the resulting machines compared with that of the initial machine:

- The states changes of several variables in the same action is not decomposed (using the substitution "becomes such that");
- The loss of information when guards are broken down;
- The disappearance of shared invariants;
- The generation of empty events in the sub-component;
- The need for an intermediate step of manual refinement before applying the decomposition.

3.4 Discussion and Synthesis

In fact, the choice of a decomposition method depends on the work finality:

- Shared variables decomposition can decompose models by functionality, for instance in the railway field, an initial railway signalling model can be decomposed into three sub-components: train integrity, block release and train communication as in figure 3.18.

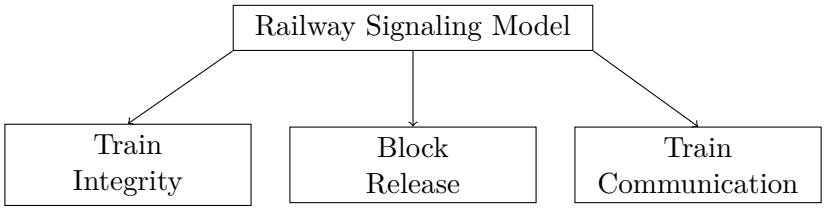


Figure 3.18: Example of Decomposition by Functionality

- Shared events decomposition is based on partitioning the behaviour of a system, e.g. partitioning according to different types of trains movements such as movement under the national *Automatic Train Protection* (*ATP*) system or under *European Railway Traffic Management System* (*ERTMS*) levels as in figure 3.19.

Nonetheless, the industrial need is to reason on sub-systems, in other words, to take into account both the behaviour and the functionality. The use of shared variables decomposition or the shared events decomposition does not address this need. Hence, after this analysis of the existing approaches and according to our industrial needs, some limitations to these techniques are identified, among others, the loss of shared invariants preserving a major safety property. Also, after the generation of the sub-machines by the plugin, the link between the original machine and the

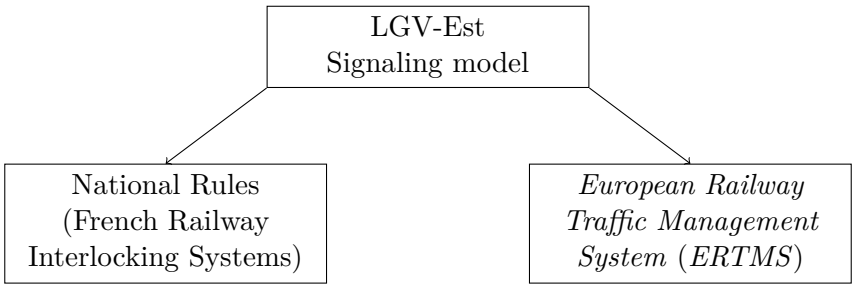


Figure 3.19: Example of Behavior Partition

sub-machine is not explicit.

As we place the preservation of the syntactic/semantic coherence from the abstract machine to the lower-level machines as a central key of the model decomposition, some conclusions are drawn from the application of these approaches which can be considered as limitations:

- Sharing variables appearing in events substitutions as full-duplex channels [Abrial, 2009] cannot be refined so that the decomposition requires an additional refinement step before decomposition in order to simplify the specification and replacing a full-duplex channel by two simple channels.
- States changes of several variables in the same action, such as becomes such that substitution, cannot be decomposed and should be dealt with the user intervention to disjoint variables following the shared events decomposition.
- Some invariants involving shared variables together with other variables in the case of shared variables decomposition are not copied in any resulting sub-machine so as to lose them when decomposing. It is also possible to lose invariants in the shared events decomposition.

These limitations, and specially the last one, give rise to a new semantic link and a new clause which we define in the Event-B language. This link is can be considered as "alike refinement" link which can exhibit the feasibility of the decomposition by preserving the behaviour like the refinement semantic link, on the one hand. On the other hand, the new clause definition makes the sub-machines interfacing possible (see figure 3.20).

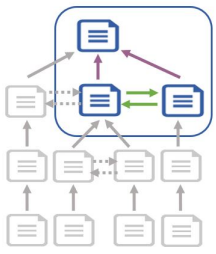


Figure 3.20: Proposed Solution for Decomposition

The next chapter is dedicated to detailing our approach, the decomposition strategy and the steps to be followed. Furthermore, it presents a demonstration of the correctness of the proposed

approach and the new underlying proof obligations.

CHAPTER 4

REFINEMENT SEEN SPLIT APPROACH

Contents

Introduction

Many approaches have been proposed to deal with the Event-B decomposition issue, among others one finds: the shared variable decomposition and the shared event decomposition. The shared variable decomposition [Abrial and Hallerstede, 2007], A-style, consists in distributing events of a system in several sub-systems. This approach proposes to manage shared variables between several events in different sub-systems. It is also used for decomposing parallel programs [Hoang and Abrial, 2010]. The shared event decomposition [Butler, 2009a], B-style, is based on the variables partition in each sub-system. Each sub-system contains the chosen variables, and the shared events between the resulting sub-systems are defined in two different signatures for each sub-system. In addition to these two approaches, one finds others such as generic instantiation [Abrial and Hallerstede, 2007], modularisation [Hoang et al., 2011], fragmentation and distribution [Siala et al., 2016].

The aim of this work is to illustrate on railway signalling systems the management of the complexity of the resulting models. For this reason, we choose to proceed with the study and analysis of A-style and B-style, because the other cited approaches imply some classical-B method [Abrial, 1996] semantics or use other languages. The analysis of A-style and B-style leads to these results: both approaches require several steps of refinement in order to simplify the model decomposition. For A-style, the shared variables shouldn't be refined but copied in the sub-systems in further refinements. The invariants involving the shared variables are not considered in the sub-systems. As for the shared events decomposition, the distribution of the variables is not always possible because of complex actions involving partitioned variables in different sub-systems or complex predicates (invariants and guards). This requires the separation of these variables by several steps of refinements with mathematical proofs. The detailed description of the state of the art, the application on a railway case study, the analysis and the identified limitations have been presented in [Kraibi et al., 2019b].

The analysis above leads us to build a new decomposition approach that corresponds to an industrial need. This technique is based on the partition of one system into many sub-systems. In this chapter, we define one of the main contributions of this thesis. It is a new approach of decomposition called *Refinement Seen Split* (*RSS*). This technique defines a new link between sub-systems: *REFSEES*. Each sub-system can make reference to other sub-systems through this new clause. In addition, we present a demonstration of the correctness of this approach by proving that the set of the resulting sub-systems is a refinement of the initial system. Then, we introduce new proof obligations associated to the new proposed approach. Finally, this approach is applied on the railway case study of the Rear-end collision (see section 3.1). The resulting sub-machines of the case study decomposition, after the application of *RSS* approach, are analysed by comparing them with the A-style and B-style results, as well as regarding the industrial need.

4.1 The Proposal: Refinement Seen Split (RSS)

In this section an informal overview of the proposed approach is presented through two sections. The first section *REFSEES Clause* 4.1.1 presents the *REFSEES* clause allowing access to a shared state space among machines, and the second section *Decomposition Strategy and Steps* 4.1.2 gives the steps to apply our proposed approach. The approach is formalised in section 4.1.3.

4.1.1 REFSEES Clause

Regarding the need of the proposed approach, we studied the different clauses of B language INCLUDES, IMPORTS, USES and SEES. The *REFSSES* clause is a similar notion to the *SEES* of the *classical-B* with particular characteristics:

– It allows the refinement of the shared variables (in writing) and

– It can be used at any level between sub-systems.

The name of the *REFSEES* clause is a combination of *REFERENCE* and *SEES* which means it allows a sub-machine to see and make reference to another sub-machine. In order to illustrate the use of this new link among machines, figure 4.1 shows an example of the decomposition of a machine M_0 into two sub-machines M_{1a} and M_{1b} in a graphical way. So we add a clause *REFSEES* to the machine M_{1a} (resp. M_{1b}), which would make reference to the variables of the machine seen M_{1b} (resp. M_{1a}). So there, we have a circular dependency with this notion of *REFSEES*. In *classical-B* there is normally no circular dependency and machines cannot *see* a refinement machine. Contrary to *SEES* clause, *REFSEES* can have a refinement machine as identifier and can be used in a cyclic way. Moreover, in the example depicted in figure 4.1 we can observe the following:

– M_0 defines the variables x, y and z, the invariants I(x,y,z) and some abstract events $event_1$,.., $event_n$. An $event_i$ contains guards $G_i(x, y, z)$ and before/after predicates $R_i(x, y, z, x', y', z')$.

– The resulting sub-machine M_{1a} (resp. M_{1b}) defines the variables x (resp. y) and z. The variable z is considered as a shared variable. the invariants $INV_{1a}(x, z)$ and $INV_{1b}(y, z)$ are defined in M_{1a} and M_{1b} respectively. The events of M_0 are partitioned in M_{1a} and M_{1b}. The machine M_{1a} defines the events $event_1$,.., $event_j$ and the machine M_{1b} defines the events $event_{j+1}$,.., $event_n$. The variable x of the machine M_{1a} is visible by each event $event_k$ in the machine M_{1b}. The variable y of the machine M_{1b} is visible by each event $event_l$ in the machine M_{1a}.

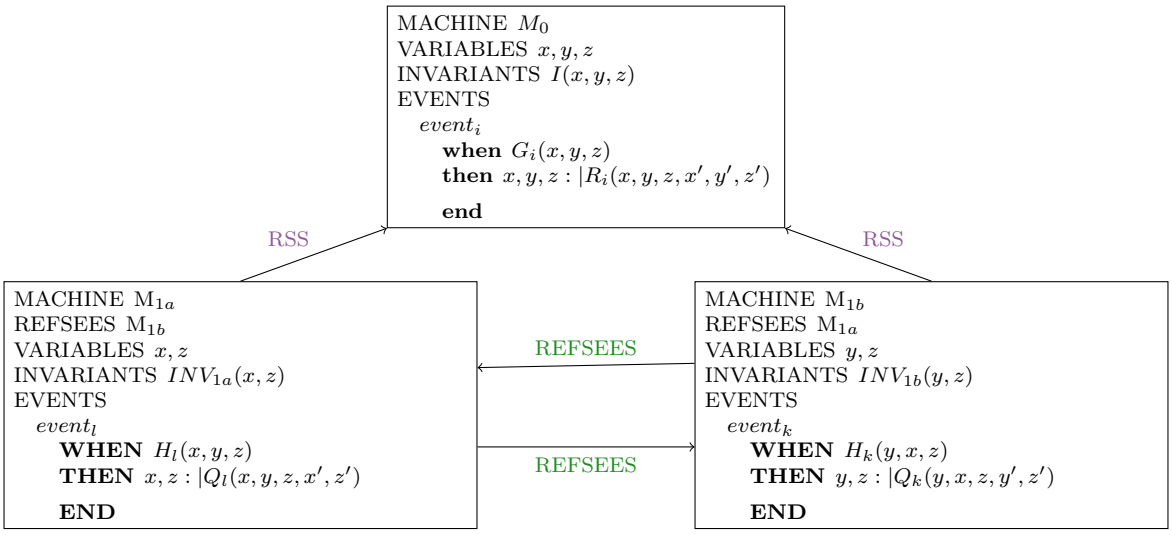


Figure 4.1: Example of the Application of the Proposed Approach

Table 4.1 is a visibility table of *REFSEES*. Sets and constants of M_{1a} are visible by axioms, invariants and events of M_{1b}. Private variables of M_{1a} are only visible by M_{1b} events. As for shared variables, they are visible and able to be modified by both the sub-machines.

M_{1b} REFSEES M_{1a}	AXIOMS	INV	INITIALIZATION / EVENTS
Sets	visible	visible	visible
Constants	visible	visible	visible
Variables		visible	visible
Events			

Table 4.1: REFSEES Visibility of M_{1a} by M_{1b}.

Figure 4.2 shows the general structure of the proposed approach. The decomposition can be applied on a certain level of refinement and done by multiple horizontal refinements. As shown in the figure a machine M_{n-1} can be refined by m machines. These resulting sub-machines keep the refinement link with the root.

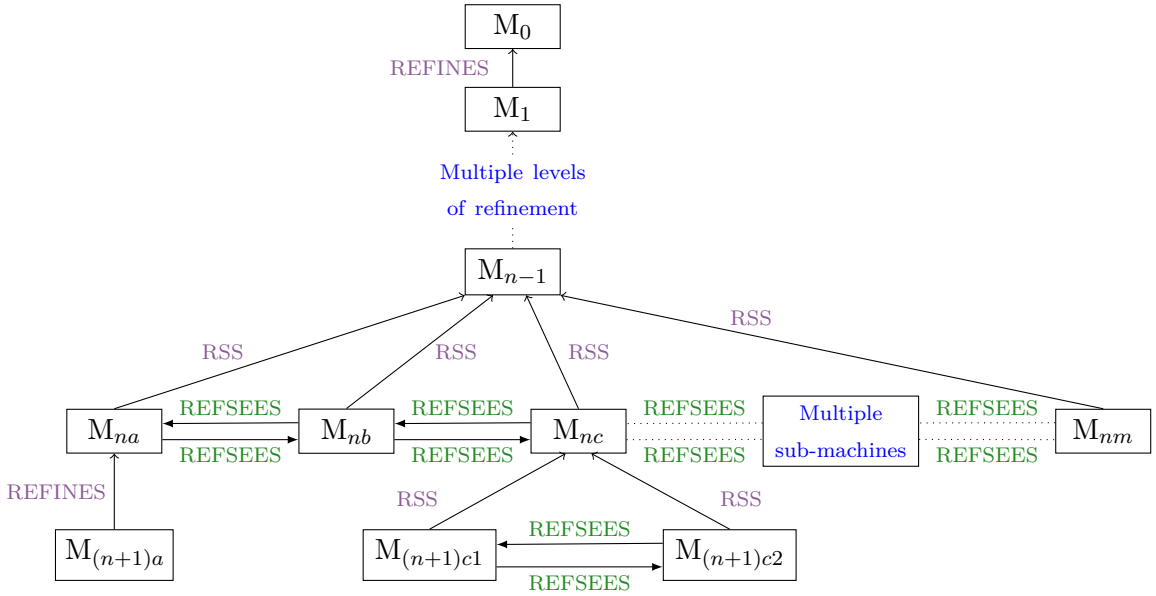


Figure 4.2: Structure of the Proposed Approach

4.1.2 Decomposition Strategy and Steps

Let consider M, the machine to be decomposed, and M_1 and M_2 the resulting sub-machines using the proposed approach. In the following, we present the strategy of decomposing the machine, their variables and their events.

Variables

(i) Variables Distribution

When decomposing, all of the existing variables in the decomposed machine M should appear at least in one of the sub-machines M_1 or M_2. In other words, the set of the variables of M constitutes the union of the existing variables in M_1 and in M_2 such as: $VAR(M) = VAR(M_1) \cup VAR(M_2)$ where $VAR(M)$ is the set of the variables of the decomposed machine M and $VAR(M_1)$ (resp. $VAR(M_2)$) is the set of the variables of the resulting sub-machine M_1 (resp. M_2). In each sub-machine M_i, the clause VARIABLES defines the variables that evolve in this sub-machine, i.e. the variables that change their states through the events: the local variables of M_i and the shared variables with the other sub-machines.

Figure 4.3 presents the case of the use of only the local variables in each sub-machine. We can observe that the variables of the machine M are partitioned as follows: the variables a and b are defined in the sub-machine M_1, and the variables c and d are defined in the sub-machine M_2. Then each local variable can be refined by another one like the variable a of the machine M_1 (resp. c of M_2) is refined by $a1$ in M_{11} (resp. $c1$ in M_{22}).

The sub-machines can also contain common variables called shared variables as shown in figure 4.4. For example, the variable y is shared in both sub-machines M_{11} and M_{22} resulting from the decomposition of the machine M_1. x is a private variable of M_{11} and z is a private variable of M_{22}.

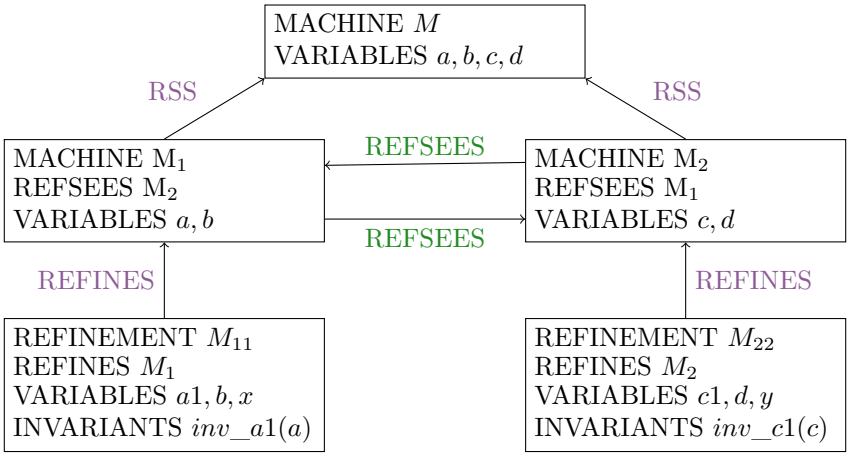


Figure 4.3: Decomposition Strategy of Variables: Case of private variables decomposition and/or add of new private variables

(ii) Definition of New Variables

When we need to define new private variables to each of the sub-machines, we can add them after the step of decomposition while refining the sub-machines. These variables can refine other private variables or not. For instance, in figure 4.3, the private variables x and $a1$ (resp. y and $c1$) are defined after the decomposition in the machine M_{11} (resp. M_{22}) where the variable $a1$ (resp. $c1$) is refining the private variable a (resp. c) of the sub-machine M_1 (resp. M_2). The invariants $inv\_a1(a)$ and $inv\_c1(c)$ are respectively the gluing invariants of the new variables $a1$ and $c1$ in function of the refined variables a and c. The variable x (resp. y) is a new variable in M_{11} (resp. M_{22}).

Otherwise, if the new variables are shared between the sub-machines, we can add them before the decomposition. This means, the machine M must be refined by another machine M_1 which defines the new needed variables. These variables will be shared in both of the resulting sub-machines after decomposing M_1. The new variables can refine some variables of M or not. For example, in figure 4.4, the variables p and q are new defined variables where p is refining w. After the decomposition, these variables are shared between M_{11} and M_{22}. After the decomposition, all the variables of the resulting sub-machines can be refined even the shared ones.

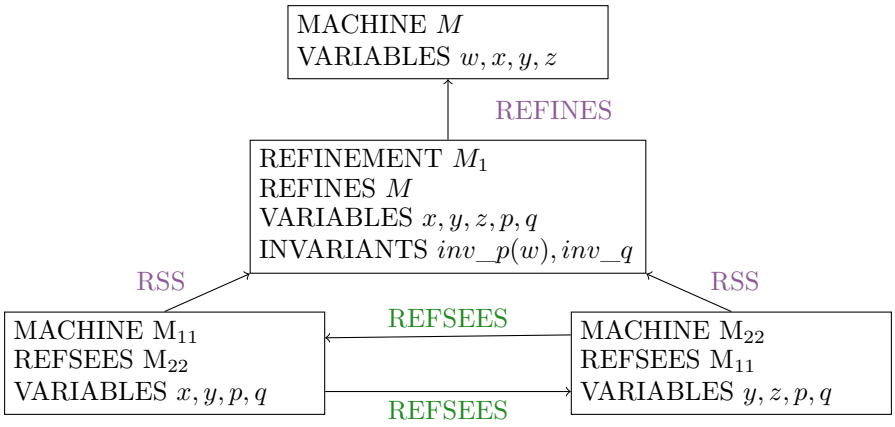


Figure 4.4: Decomposition Strategy of Variables: Case of shared variables decomposition and/or add of new shared variables

Events

(i) Events Distribution

Each event of the machine M should appear at least in one of the sub-machines M_1 or M_2, which means $EVENT(M) = EVENT(M_1) \cup EVENT(M_2)$ where $EVENT(M)$ is the set of the events of the machine M and $EVENT(M_1)$ (resp. $EVENT(M_2)$) is the set of the events of the machine M_1 (resp. M_2).

The sub-machine M_1 (resp. M_2) must contain distinct events from those of the other sub-machine M_2 (resp. M_1), in other words $EVENT(M_1) \cap EVENT(M_2) = \varnothing$. For example, in

figure 4.5, after the decomposition of the machine M that defines 4 events, the machine M_1 (resp. M_2) contains the events $event_1$ and $event_2$ (resp. $event_3$ and $event_4$).

If we need the same event in both of the machines, this event must be refined by two events where each one will be in a sub-machine. As example, in figure 4.6, $event_1$ of the machine M is refined by $event_{1a}$ and $event_{1b}$ in the machine M_1, then they are partitioned. $event_{1a}$ is defined in M_{11} and $event_{1b}$ is defined in M_{22}.

While decomposing, the events are copied according to the needed events in each sub-machine. In order to preserve the conditions that allow the events triggering and to keep the same behaviour as defined in the decomposed machine M, the **guards** should be copied as they are defined in the decomposed machine. The **actions** are also copied as they are defined in the abstract event of the decomposed machine because of the atomicity of the events. If guards and/or actions are decomposed, the system may block, and some states cannot be reached as in M.

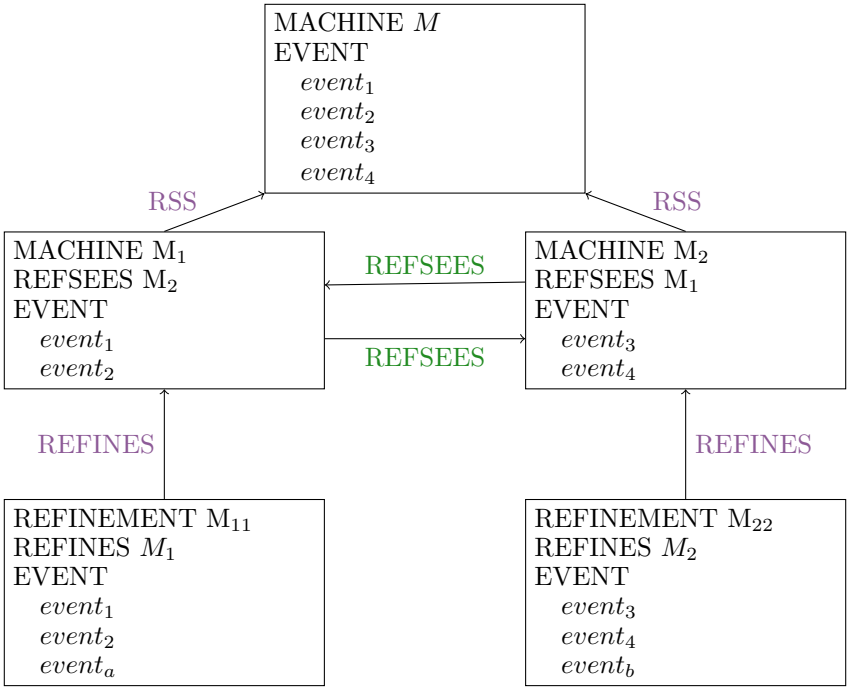


Figure 4.5: Decomposition Strategy of Events: Case of private events decomposition and/or add of new private events

(ii) Definition of New Events

The definition of new events can be under two forms. The first one is the definition of a private event in one of the sub-machines M_1 and/or M_2. This step can be done later in the refinement of the sub-machine M_1 and/or M_2. Figure 4.5 shows the definition of new private events. $event_a$ and $event_b$ are new private events in the refinement of M_1 and M_2 respectively. The event $event_a$ (resp. $event_b$) can be a refinement of *skip* or of an abstract event in M_1

(resp. M_2).

The second form is the definition of a shared event between both of the sub-machines M_{11} and M_{22}. In this case, the event is defined before the decomposition in a refinement machine M_1. Then this event is refined by two events in another refinement machine M_1' that is decomposed. For instance, , $event_w$ is a new defined event in M_1. This event is refined by $event_{wa}$ and $event_{wb}$ in a refinement machine M_1'. Then, the event $event_{wa}$ (resp. $event_{wb}$) is partitionned in M_{11} (resp. M_{22}). After the decomposition, all the events of the sub-machines can be refined.

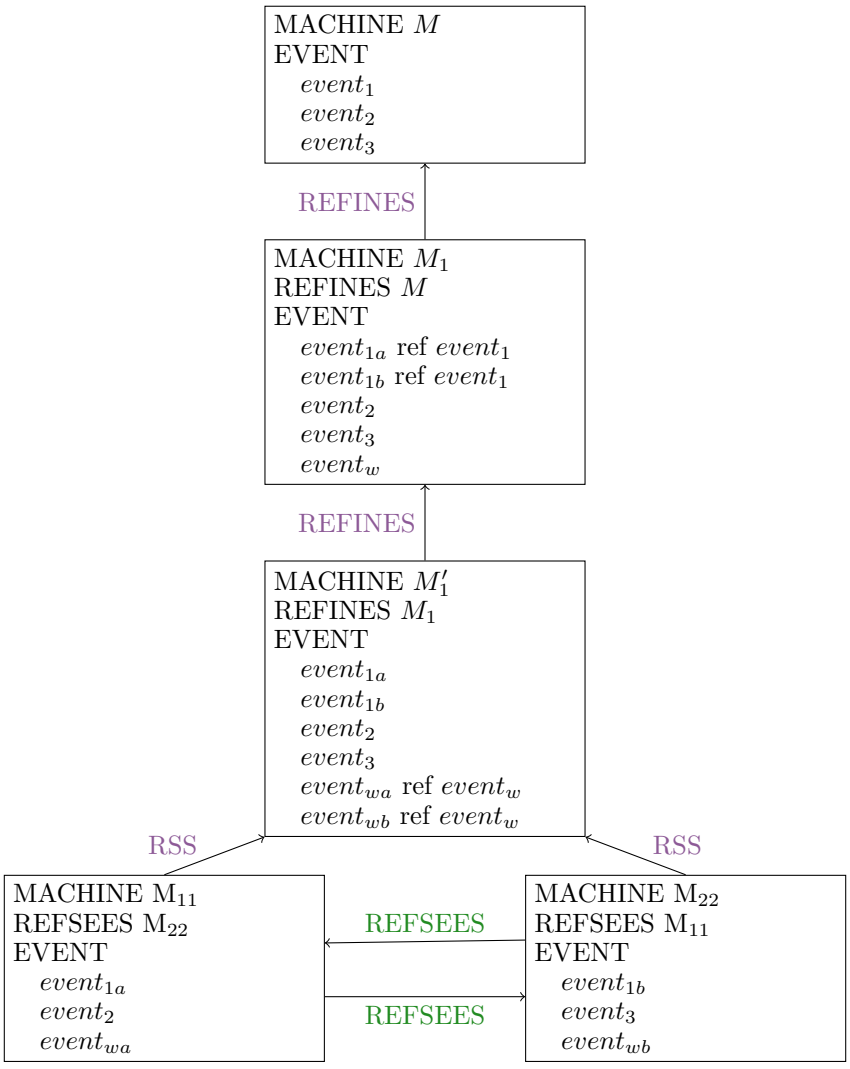


Figure 4.6: Decomposition Strategy of Events: Case of shared events decomposition and/or add of new shared events

Invariants

While decomposing, the invariants of M should appear in each sub-machine M_i, i.e. the union of the invariants of the sub-machines M_i constitutes the invariants of M such as $INV(M) = \bigcup INV(M_i)$, where $INV(M)$ represents the invariants of M. The definition of new invariants involving shared variables can be done in a refinement machine M_1 of the machine M and then decomposing M_1. Otherwise, if these new added invariants are involving the private variables of one sub-machine, this can be performed in the refinement of the corresponding sub-machine.

To conclude the proposal, some remarks should be noted:

- **Remark 1**: the state variables v of the decomposed machine M should all be present at least in M_a or M_b. Some state variables can only be in one of the sub-machines. In the semantic of Event-B [Abrial, 2002], there is a notion of external-set which allows to ignore some "internal" variables in the refinement. We believe we can use this concept to justify this new usage of the refinement. The proof obligation rules imply the semantic definition of refinement:

  $$r_a^{-1}; re_a \subseteq ae; r_a^{-1}$$

  where $r_a = \{w_a \mapsto v | I(v) \wedge J_a(v, w_a)\}$, I the invariant of M, J_a the invariant of M_a

  $$\& \qquad r_b^{-1}; re_b \subseteq ae; r_b^{-1}$$

  where $r_b = \{w_b \mapsto v | I(v) \wedge J_b(v, w_b)\}$, I the invariant of M, J_b the invariant of M_b.

- **Remark 2**: the main difference with normal refinement, is the fact that the sub-machine M_a (resp. M_b) can refer (in their events guards) to variables that can be only present in M_b (resp. M_a).
  This might be possible because we did already prove the M is correct. But we must prove that this refinement is correct regarding the theoretical definition of the B-Method [Abrial, 1996].

- **Remark 3**: the resulting sub-machines M_a and M_b should correspond to a one transition system that corresponds to the behaviour of M.

- **Remark 4**:

  M_a and M_b transitions are interlaced and then they are not synchronised contrary to the decomposition by shared events.
  This is done by the definition of the (theoretical) re-composition of the sub-machines. Consequently, we can demonstrate that this way of re-composition is a refinement of the machine M, following what has been presented in [Abrial, 2009].

- **Remark 5**: in addition to the partial correctness, which consists on proving that the system is safe i.e. preserving the safety invariants, we should ensure the complete correctness. This later consists on ensuring the vivacity properties such as those of the variant and the deadlock freedom. A transition should not be triggered indefinitely. So, a variant proof obligation rule should be defined. Concerning the deadlock freedom, it allows to prove that the system does not block [Abrial, 2010].

  In case of adding new events in the sub-machines, the deadlock freedom rules, and the variant rules should also be defined.

4.1.3 Formalising the Approach

In *Event-B*, during refinement, we have the right to merge one or more events, to refine an event and to add new events, but not delete events. The delete will cause a loss of some traces of the initial machine. In this case, the system may block when it was not the case in the initial machine. So, the idea is to use some characteristics of the classical refinement in order to keep the machines coherence. So, we define a new link between the decomposed machine and the resulting sub-machines. This link allows to partition a machine into multiple sub-machines.

Let consider a machine M. This machine is modelled by the choice of its events (see Definition 4.1.1) through a binary relation $ae(M)$ such as
$ae(M) : S \leftrightarrow S$, where $S = State\_Space(M)$.

Definition 4.1.1. "*Choice of events*: Let M be an Event-B machine with a set of events $event_1, event_2, ..event_n$. The "choice of events" of M, $event\_choice(M)$, is defined by:
$event\_choice(M) = event_1 [] event_2 [] .. [] event_n$. From an operational point of view only one event of $event\_choice(M)$ can be trigered."

The machine M is partitioned into several sub-machines M_i, as in figure 4.7. Events of M are partitioned, according to the *"Decomposition Strategy"* in section 4.1.2. Each sub-machine resulting from the partition can make reference to other sub-machines through the *REFSEES* clause. S is the state space of M. Each system M_i, resulting from the partition of a system M, shares the state space S of M. The set of systems resulting from the partition strategy is called "**Refinement Seen Split**" or **RSSplit** (see Definition 4.1.2).

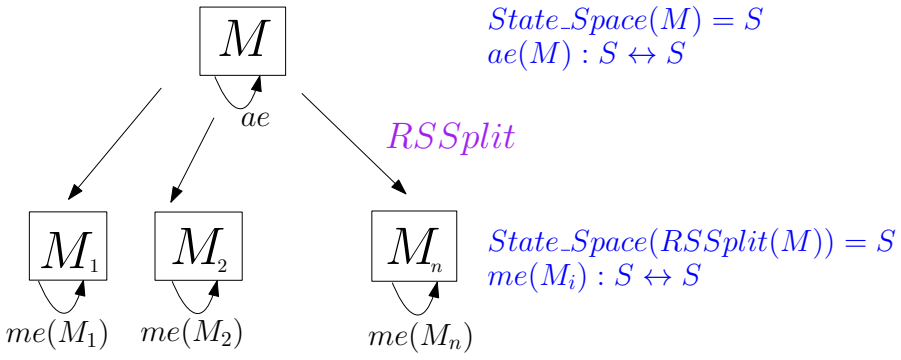


Figure 4.7: Refinement Seen Split

Definition 4.1.2. "*RSSplit*: Let M be an Event-B machine. $RSSplit(M)$ denotes the set of Event-B machines resulting from the "Decomposition Strategy" applied to the machine M.
RSSplit allows to partition a machine M into a set of multiple sub-machines such as $RSSplit(M) = \{M_1, M_2, ..., M_n\}$."

The state space associated with a *Refinement Seen Split* set, $RSSplit(M) = RSS$, is denoted by $State\_Space(RSS)$. Each system M_i is modelled by the choice of its events (see Definition

4.1.1) through a binary relation $me(M_i)$ such as $me(M_i) : S \leftrightarrow S$, where $M_i : RSSplit(M)$ and $S = State\_Space(RSSplit(M))$.

For a machine M, $RSSplit(M)$ models the result of the split operation defined through the "Decomposition Strategy". In order to be able to prove the correctness of the refinement of the machine M by a set of sub machines, we define in Definition 4.1.3 the *Merge* of sub-machines over a *Refinement Seen Split* set RSS:

Definition 4.1.3.

"*Merge of Sub-machines*: Let $RSS = \{M_1, ...M_n\}$ be the *Refinement Seen Split* of a system. The merge of sub-machines **Merge(RSS)** is the system made up of the union of the sub-systems in *RSS*.

- The VARIABLES clause is made up of the union of variables in each M_i.
- The INVARIANT clause is made up of the conjunction of invariants in each M_i.
- The EVENT clause is made up of the union of events in each M_i."

In the basis of this definition and the definition of the *Refinement Seen Split* of a system M we have:

$$Merge(RSSplit(M)) = M$$

As a consequence, we can say that *Merge(RSSplit(M))* is a refinement, in the Event-B sense, of System M.

Figure 4.8 shows the *Merge* of the sub-machines. The model of $Merge(RSS)$, for any *Refinement Seen Split* set RSS having $State\_Space(RSS) = S$, is given by a relation $mm(Merge(RSS)) : S \leftrightarrow S$ defined by Definition 4.1.4, where
$mm(Merge(RSS))$ is the choice of events of $Merge(RSS)$ (see Definition 4.1.1).

Definition 4.1.4. $mm(Merge(RSS)) = UNION(m).(m : RSS|me(m))$

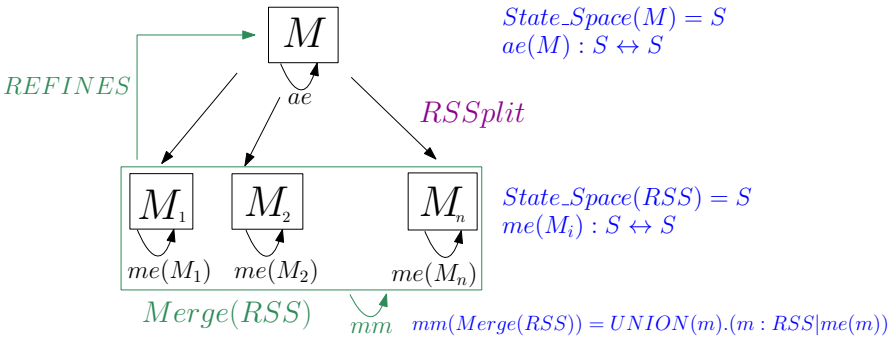


Figure 4.8: Merge of M_i is a Refinement of M

Each sub-machine M_i in the set RSS can be refined into another one M_i' leading to a set of refined sub-machines RSS', as illustrated in figure 4.9.

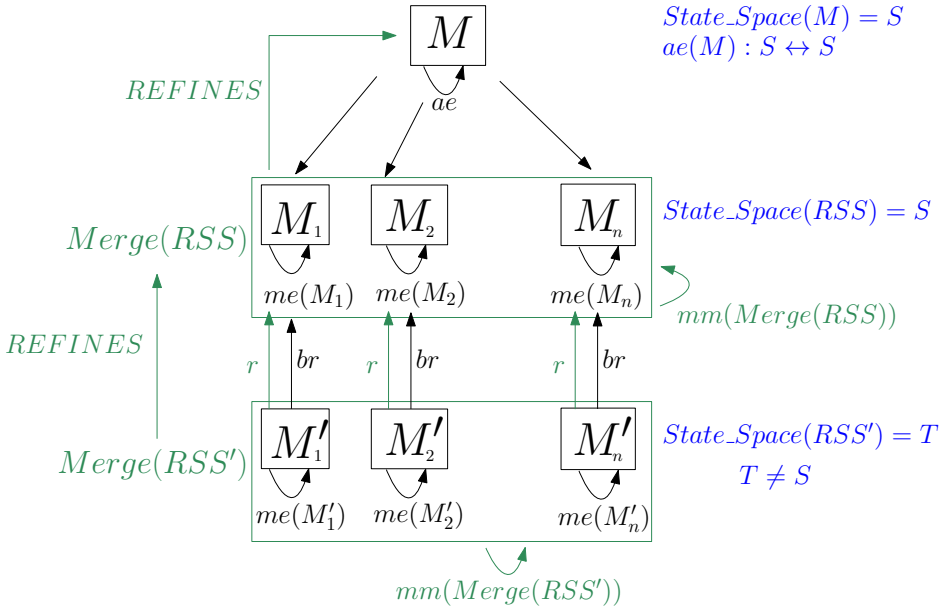


Figure 4.9: Refinement after Decomposition

In order to denote this link of refinement, a total bijection br among the machines of RSS' and RSS is defined:

$$br \in RSS' \twoheadrightarrow RSS$$

Therefore, as M'_i is the refinement of M_i, we have $M'_i = br^{-1}(M_i)$. We suppose that the state space of RSS' is T ($State\_Space(RSS') = T$) and that the refinement relation among the abstract and concrete sub-machines is r, therefore the relation refinement is a total relation between T and S:

$$r \in T \leftrightarrow S$$

Actually, in this part we present the core of the thesis contribution: a new decomposition approach (called RSS: Refinement Seen Split). This approach does not require external events in the sub-models to simulate the evolution of the shared variable. Among its properties, the state space of the combined model is simply the union of the state spaces of the sub-models. This is justified with argument in the next section of the approach correctness proof. This property is valid only at the highest level of RSS refinement where there is no data refinement or new defined variables. As for the lower levels of RSS refinement, the state space can change such that $S/ = T$ with the data refinement as shown in Figure 4.9. In the following section it is proved that the merge of refinement machines in RSS' is a refinement of the merge of machines in RSS.

4.2 Correctness of the RSS Approach

4.2.1 Demonstration of the Proposed Approach

In this section, our goal is to demonstrate that the combination of transition system of all the sub-machines corresponds to the transition system of the decomposed machine. So, the behaviour due to the combination of the sub-systems preserves the behaviour of the initial system. In other

words, our main goal is to prove that the global system $Merge(RSS')$ combining the resulting sub-machines M_i' is a refinement of M. So, since the global system $Merge(RSS)$ combining the resulting sub-machines M_i is refining M, it is sufficient to prove that $Merge(RSS')$ is a refinement of $Merge(RSS)$.

So, we have to prove this theorem:

$$\forall m.(m \in RSS' \implies (r^{-1}; me(m)) \subseteq (me(br(m)); r^{-1})) \vdash (r^{-1}; mm(RSS')) \subseteq (mm(RSS); r^{-1})$$

In figure 4.10, it is shown in a graphical view that each machine M_i', such as $M_i' \in RSS'$, is a refinement of one machine M_i, where $M_i \in RSS$ and $M_i = br(M_i')$, as presented in the assumption 4.1.

$$\boxed{\forall m.(m \in RSS' \implies (r^{-1}; me(m)) \subseteq (me(br(m)); r^{-1}))} \tag{4.1}$$

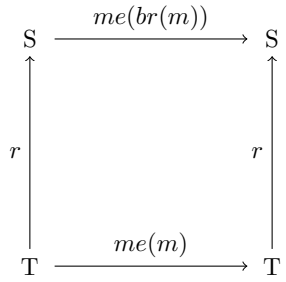


Figure 4.10: Each Machine M_i' is a Refinement of M_i

For each machine m such as $m \in RSS'$, we can find its corresponding machine in RSS such as $br(m) \in RSS$. Therefore, the composition of $me(br(m))$ with the inverse relation r^{-1} is included in the union of the composition of $me(n)$ with r^{-1} for $n \in RSS$:

$$\boxed{\forall m.(m \in RSS' \implies (me(br(m)); r^{-1}) \subseteq UNION(n).(n \in RSS | me(n); r^{-1}))} \tag{4.2}$$

We need the following Composition Property:

$$\boxed{UNION(n).(n \in RSS | me(n); r^{-1}) = (UNION(n).(n \in RSS | me(n)); r^{-1})} \tag{4.3}$$

From equation 4.3, equation 4.2 we get:

$$\boxed{\forall m.(m \in RSS' \implies (me(br(m)); r^{-1}) \subseteq (UNION(n).(n \in RSS | me(n)); r^{-1}))} \tag{4.4}$$

From the model of Merge operation in definition 4.1.4 and the equation 4.4 we derive:

$$\boxed{\forall m.(m \in RSS' \implies (me(br(m)); r^{-1}) \subseteq (mm(RSS); r^{-1}))} \tag{4.5}$$

Back to equation 4.1, we use equation 4.5 and transitivity of inclusion to obtain:

$$\boxed{\forall m.(m \in RSS' \implies (r^{-1}; me(m)) \subseteq (mm(RSS); r^{-1}))} \tag{4.6}$$

We note that the composition of each refining machine in RSS' with the inverse relation r^{-1} is included in the union of compositions of the machines of RSS' with the inverse relation:

$$\boxed{\forall m.(m \in RSS' \implies (r^{-1}; me(m)) \subseteq UNION(n).(n \in RSS' | (r^{-1}; me(n)))} \quad (4.7)$$

Using equation 4.7, we derive the following inclusion from the equation 4.6:

$$\boxed{UNION(m).(m \in RSS' | (r^{-1}; me(m))) \subseteq (mm(RSS); r^{-1})} \quad (4.8)$$

In the next step, we need the following Composition Property:

$$\boxed{UNION(m).(m \in RSS' | (r^{-1}; me(m))) = (r^{-1}; UNION(m).(m \in RSS' | me(m)))} \quad (4.9)$$

Using property 4.9, we obtain from the equation 4.8:

$$\boxed{(r^{-1}; UNION(m).(m \in RSS' | me(m))) \subseteq (mm(RSS); r^{-1})} \quad (4.10)$$

From the model of merge in definition 4.1.4, equation 4.10 is equivalent to:

$$\boxed{(r^{-1}; mm(RSS')) \subseteq (mm(RSS); r^{-1})} \quad (4.11)$$

We have therefore performed the required demonstration: after several steps of refinement, the theoretical composition of the resulting sub-machines is a refinement of the initial machine. This demonstration is a major step and is essential for this thesis.

4.2.2 Definition of New Proof Obligations

Following the proposed approach, a machine M can be decomposed into several sub-machines. Let consider the case of two resulting sub-machines M_1 and M_2:

- M defines the variables v, the invariants $I(v)$ and the events: $event_1$,...,$event_N$.
- The guard of each event $event_i$, of the machine M, is defined by G_i. So, the guards G_1,...,G_N correspond to the events $event_1$,...,$event_N$ respectively.
- M_1 defines the variables $VAR(M_1)$, the invariants $INV_1(VAR(M_1))$ and the events: $event_1$,...,$event_j$.
- The guard of each event $event_k$, of the machine M_1, is defined by H_{1k}. So, the guards H_{11},...,H_{1j} correspond to the events $event_1$,...,$event_j$ respectively.
- M_2 defines the variables $VAR(M_2)$, the invariants $INV_2(VAR(M_2))$ and the events: $event_{j+1}$,...,$event_N$.
- The guard of each event $event_l$, of the machine M_2, is defined by H_{2l}. So, the guards $H_{2(j+1)}$,...,H_{2N} correspond to the events $event_{j+1}$,...,$event_N$ respectively.

We present in the following the definition of the new proof obligations of the proposed approach. We define two types of rules to generate proof obligations as shown in figure 4.11:

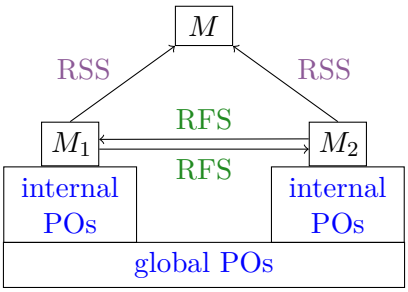


Figure 4.11: Structure of the Proof Obligations Generation

– **Local**: the local rules are those allowing to prove that a sub-machine is a refinement machine. In other words, they are the rules generating proof obligations in the classical refinement, as defined in section 2.1.2.

– **Global**: the global rules are a the new rules defined due to the definition of our approach. These rules allow to prove that the combination of the sub-machines is a correct refinement of the decomposed machine. In the following we detail the global rules.

The correction of the sub-machines combination is done throw the REFSEES link. As mentioned in section 4.1.1, the REFSEES clause allows to have a visibility on the other sub-machine. Consequently, this visibility makes it possible to generate the global proof obligations.

Deadlock Freedom: DLF

In the following, we define the deadlock freedom proof obligation rules that allow to guarantee that the system does not block.

As presented in section 2.1.2, there are two types of DLF proof obligation rules: weak and strong DLF. The weak DLF proof obligation, called $DLF1_w$, guarantee that at least one of the events in the sub-machines M_1 or M_2, resulting from the decomposition of M, can be triggered. As for the strong DLF proof obligation, named $DLF1_s$, it allows to ensure the trigger of each event in the sub-machines.

In the case of the proposed decomposition approach, the proof obligation rules $DLF1_w$ and $DLF1_s$ do not need to be defined for the RSS approach because they are hold trivially. Indeed, after a split of the initial machine M, the guard $G_i(v)$ where $i \in 1..N$ implies the guard of an event of M_1 or M_2 because the event $event_i$ is necessarily in M_1 or M_2.

We define the weak and strong DLF proof obligation rules, $DLF2'_w$ and $DLF2'_s$, in case of the refinement of M_1 and M_2 by M'_1 and M'_2 respectively where:

– M'_1 defines the variables $VAR(M'_1)$, the gluing invariants $J'_1(VAR(M_1), VAR(M'_1))$ and the events $event'_1$,...,$event'_j$.

– The events $event'_1$,...,$event'_j$, of the machine M'_1, are refining respectively the events $event_1$,...,$event_j$ of the machine M_1.

– The guard of each event $event'_k$, of the machine M'_1, is defined by H'_{1k}. So, the guards H'_{11},...,H'_{1j} correspond to the events $event'_1$,...,$event'_j$ respectively.

- M'_2 defines the variables $VAR(M'_2)$, the gluing invariants $J'_2(VAR(M_2), VAR(M'_2))$ and the events $event'_{j+1}$,...,$event'_N$.
- The events $event'_{j+1}$,...,$event'_N$, of the machine M'_2, are refining respectively the events $event_{j+1}$,...,$event_N$ of the machine M_2.
- The guard of each event $event'_l$, of the machine M'_2, is defined by H'_{2l}. So, the guards $H'_{2(j+1)}$,...,H'_{2N} correspond to the events $event'_{j+1}$,...,$event'_N$ respectively.

Table 4.2 defines the proof obligations for the sub-machine M_1 and its refinement M'_1 in case of the refining events existence. The same proof obligation rules are used for the sub-machine M_2 and its refinement M'_2.

Weak deadlock freedom of the refinement machine M'_1 in case of the refining events existence	$DLF2'_w$	$INV_1(VAR(M_1))$ $J'_1(VAR(M_1), VAR(M'_1))$ $J'_2(VAR(M_2), VAR(M'_2))$ $H_{11}(VAR(M_1)) \vee .. \vee H_{1j}(VAR(M_1))$ $\vee H_{2(j+1)}(VAR(M_2)) \vee .. \vee H_{2N}(VAR(M_2))$ $\vdash$ $H'_{11}(VAR(M'_1)) \vee .. \vee H'_{1j}(VAR(M'_1))$ $\vee H'_{2(j+1)}(VAR(M'_2)) \vee .. \vee H'_{2N}(VAR(M'_2))$
Strong deadlock freedom of the refinement machine M'_1 in case of the refining events existence	DLF2$_s$'	$INV_1(VAR(M_1)$ $J'_1(VAR(M_1), VAR(M'_1))$ $J'_2(VAR(M_2), VAR(M'_2))$ $H_{1k}(VAR(M_1))$ $\vdash$ $H'_{1k}(VAR(M'_1)$
Strong deadlock freedom of the refinement machine M'_2 in case of the refining events existence	DLF2$_s$'	$INV_2(VAR(M_2)$ $J'_1(VAR(M_1), VAR(M'_1))$ $J'_2(VAR(M_2), VAR(M'_2))$ $H_{2l}(VAR(M_2))$ $\vdash$ $H'_{1l}(VAR(M'_2))$

Table 4.2: Deadlock Freedom Proof Obligations: DLF, in case of the Existence of Refining Events

In addition to the previous defined proof obligation rules, the deadlock freedom proof obligations $DLF3'_w$ and $DLF3'_s$ are defined in case of the existence of new events in the refinement machines M'_1 and M'_2, for example:

- M'_1 can define new events $N\_event'_{11}$,...,$N\_event'_{1n}$. These events contains the guards $N'_{11}(VAR(M'_1))$, ... ,$N'_{1n}(VAR(M'_1))$ respectively.
- M'_2 can define new events $M\_event'_{21}$,...,$M\_event'_{2m}$. These events contains the guards $M'_{21}(VAR(M'_2))$, ... ,$M'_{2m}(VAR(M'_2))$ respectively.

Table 4.3 defines the proof obligations for the refinement M'_1 of the sub-machine M_1 in case of the new events existence. The same proof obligation rules are used for the refinement M'_2 of the sub-machine M_2.

Weak deadlock freedom of the refinement machine M_1' in case of the existence of refining and new events	DLF3$_w$'	$INV_1(VAR(M_1)$ $J_1'(VAR(M_1), VAR(M_1'))$ $J_2'(VAR(M_2), VAR(M_2'))$ $H_{11}(VAR(M_1)) \vee .. \vee H_{1j}(VAR(M_1))$ $\quad \vee H_{2(j+1)}(VAR(M_2)) \vee .. \vee H_{2N}(VAR(M_2))$ $\vdash$ $H'_{11}(VAR(M_1')) \vee .. \vee H'_{1j}(VAR(M_1'))$ $\quad \vee H'_{2(j+1)}(VAR(M_2')) \vee .. \vee H'_{2N}(VAR(M_2'))$ $\quad \vee N'_{11}(VAR(M_1')) \vee .. \vee N'_{1n}(VAR(M_1'))$ $\quad \vee M'_{21}(VAR(M_2')) \vee .. \vee M'_{2m}(VAR(M_2'))$
Strong deadlock freedom of the refinement machine M_1' in case of the existence of refining and new events	DLF3$_s$'	$INV_1(VAR(M_1)$ $J_1'(VAR(M_1), VAR(M_1'))$ $J_2'(VAR(M_2), VAR(M_2'))$ $H_{1k}(VAR(M_1))$ $\vdash$ $H'_{1k}(VAR(M_1')) \vee N'_{11}(VAR(M_1')) \vee .. \vee N'_{1n}(VAR(M_1'))$ $\quad \vee M'_{21}(VAR(M_2')) \vee .. \vee M'_{2m}(VAR(M_2'))$
Strong deadlock freedom of the refinement machine M_2' in case of the existence of refining and new events	DLF3$_s$'	$INV_2(VAR(M_2)$ $J_1'(VAR(M_1), VAR(M_1'))$ $J_2'(VAR(M_2), VAR(M_2'))$ $H_{2l}(VAR(M_2))$ $\vdash$ $H'_{1l}(VAR(M_2')) \vee N'_{11}(VAR(M_1')) \vee .. \vee N'_{1n}(VAR(M_1'))$ $\quad \vee M'_{21}(VAR(M_2')) \vee .. \vee M'_{2m}(VAR(M_2'))$

Table 4.3: Deadlock Freedom Proof Obligations: DLF, in case of the Existence of New Events

Variant: VAR

Let consider M_1 and M_2 the sub-machines of M, and M_1' and M_2' are respectively their refinements. Since we cannot define any new event in M_1 and M_2, let consider, for example, $event_a$ as a new defined event in the refinement machine M_1' where:

- $INV_1(VAR(M_1))$ is the invariant to preserve in M_1.
- $INV_1'(VAR(M_1), (VAR(M_1'))$ is the gluing invariant of M_1'.
- $H_a(VAR(M_1'))$ is the guard of the event $event_a$ in the machine M_1'.
- $BA_a(VAR(M_1), VAR(M_1'))$ is the Before/After predicate of the event $event_a$ in M_1'.
- $V_1(VAR(M_1))$ the variant of the machine M_1.
- $V_2(VAR(M_2))$ the variant of the machine M_2.

So, we have $RSS = \{M_1, M_2\}$ and $RSS' = \{M_1', M_2'\}$. For any machine m such as $m \in RSS'$, we have the global variant G_{Var} defined as $G_{Var} = SIGMA(m).(m : RSS'|v(m))$ where $v(m)$ is the expression of the variant clause for any machine m in RSS'. In case of the event $event_a$ it must be proved $G'_{Var} < G_{var}$.

So, the proof obligation using decreasing natural is $V_1(VAR(M_1'))+V_2(VAR(M_2')) < V_1(VAR(M_1))+ V_2(VAR(M_2))$. Following the same reasoning, the proof obligation using a finite set is $V_1(VAR(M_1'))+ V_2(VAR(M_2')) < V_1(VAR(M_1)) + V_2(VAR(M_2))$.

Variant Proof obligation using decreasing natural	VAR1	$INV_1(VAR(M_1))$ $INV_1'(VAR(M_1), VAR(M_1'))$ $H_a(VAR(M_1'))$ $BA_a(VAR(M_1), VAR(M_1'))$ $\vdash$ $V_1(VAR(M_1')) + V_2(VAR(M_2')) < V_1(VAR(M_1)) + V_2(VAR(M_2))$
Variant proof obligation using a finite set	VAR2	$INV_1(VAR(M_1))$ $INV_1'(VAR(M_1), VAR(M_1'))$ $H_a(VAR(M_1'))$ $BA_a(VAR(M_1), VAR(M_1'))$ $\vdash$ $(S_1(VAR(M_1')) \cup S_2(VAR(M_2'))) \subset (S_1(VAR(M_1)) \cup S_2(VAR(M_2)))$

Table 4.4: Variant Proof Obligations: VAR

4.3 Application of the RSS on the Railway Case Study

Back to the model of the case study, lets apply the RSS approach on the case study. Figure 4.12 shows the structure of the application of the RSS on the case study. The goal is to decompose the machine M_2 by separating Track and Train behaviours, and their functionalities using the decomposition RSS such as $RSSplit(M2) = \{Train, Track\}$.

The *Track* machine, in figure 4.13, *refsees* the *Train* machine through the *REFSEES* link. Besides, the machine *Track* contains the variables associated to the track like the blocks states variable: *block_state*. It contains also events that make these variables evolve such as *TRACKevent*.

As for the *Train* machine, in figure 4.14, it *refsees* the *Track* machine through the *REFSEES* link. It describes the train variables like *front_trainA* and the trains movement events e.g. *enter_tAblock*.

The variable *next_turn* is a shared variable of *Track* and *Train*. Partitioned events keep their guards in the sub-machines.

So, the composition of the sub-machines *Trakc* and *Train* constitutes a refinement of the machine M_2 such as $Merge(Track, Train)$ refines M_2.

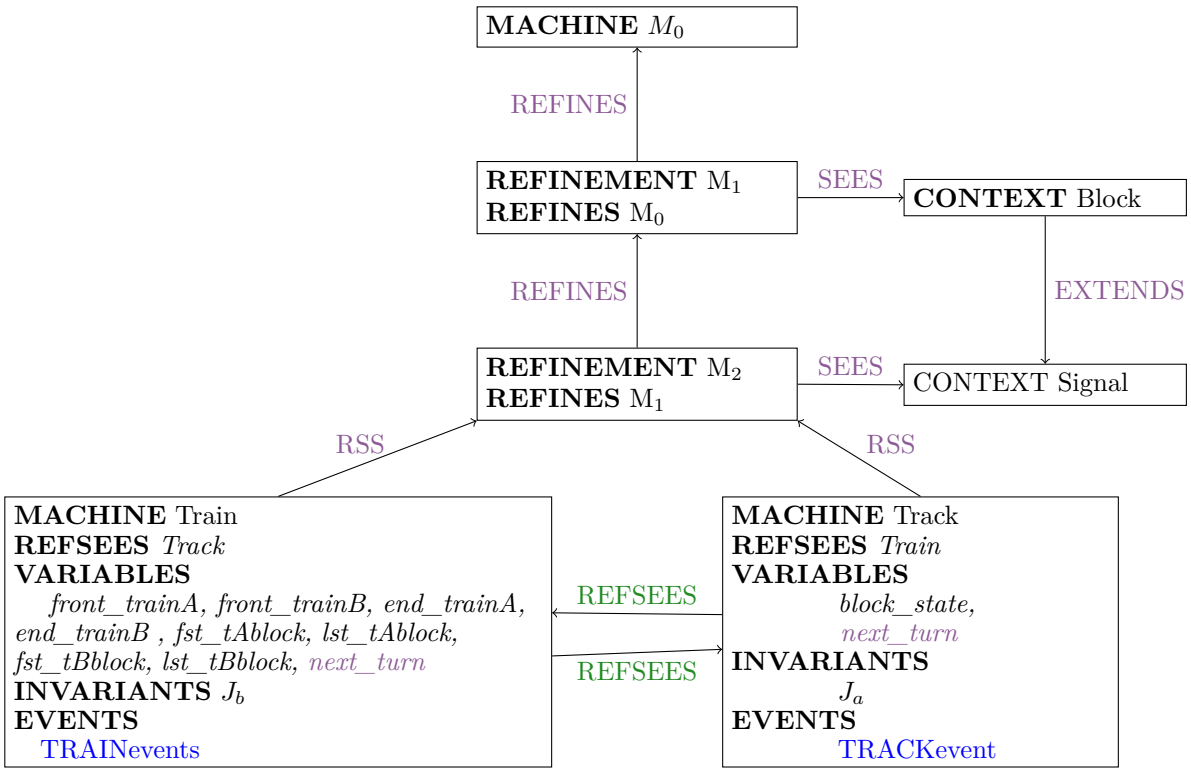


Figure 4.12: Structure of the Application of the RSS Approach on the Case Study

MACHINE Track
REFSEES Train
SEES Signal
VARIABLES
signal_state
block_state
next_turn
INVARIANTS
inv1: $signal\_state \in Signal \rightarrow SignalState$
inv2:
$\forall bk \cdot ($
$bk \in Block$
$\wedge\ next\_turn = TRAIN$
$\wedge\ signal\_state(signal\_block(bk)) = green$
$\Rightarrow block\_state(bk) = Free)$
EVENTS
Initialisation
begin
act1: ...
end
Event TRACKevent ⟨ordinary⟩ $\widehat{=}$
when
grd1: $next\_turn = TRACK$
then
act1: $next\_turn := TRAIN$
act2: $block\_state := (Block \times \{Free\}) \lhd ((lst\_tAblock \mathinner{..} fst\_tAblock \cup lst\_tBblock \mathinner{..} fst\_tBblock) \times \{Occupied\})$
act3: $signal\_state := (Signal \times \{green\}) \lhd ((signal\_block(lst\_tAblock) \mathinner{..} signal\_block(fst\_tAblock) \cup signal\_block(lst\_tBblock) \mathinner{..} signal\_block(fst\_tB2block)) \times \{red\})$
end
END

Figure 4.13: Excerpt of the Track Sub-machine after RSS

MACHINE Train
REFSEES Track
SEES Signal
VARIABLES
front_trainA
front_trainB
end_trainA
end_trainB
fst_tAblock
lst_tAblock
fst_tBblock
lst_tBblock
next_turn
EVENTS
Initialisation
begin
act1: ...
end
Event Enter_tBblock ⟨ordinary⟩ $\widehat{=}$
any
step
where
grd1: $step \in \mathbb{N}_1$
grd2: $block\_state(next\_block(fst\_tBblock)) = Free$
grd3: $front\_block(fst\_tBblock) < front\_trainB + step$
grd4: $front\_trainB + step < front\_block(next\_block(fst\_tBblock))$
grd5: $next\_turn = TRAIN$
then
act1: $next\_turn := TRACK$
act2: $front\_trainB := front\_trainB + step$
act3: $fst\_tBblock := next\_block(fst\_tBblock)$
end
END

Figure 4.14: Excerpt of the Train Sub-machine after RSS

As we can see from this decomposition example, the shared state of a system among the sub-machines, through the *Refinement Seen Split* (*RSS*) approach, allows a simple decomposition strategy which is simpler than the main existing approaches. The proposed approach satisfies the objectives of modularity as well as the syntactic and semantic coherence among the modules composing a system that are the motivations of our work considering the industrial needs.

Indeed, one of the reasons behind the definition of a new approach of decomposition is the optimisation of the number of refinement steps before the application of the decomposition. Contrary to the existing approaches of decomposition, that needs to add more refinement levels to simplify the decomposition, the *Refinement Seen Split* (*RSS*) allows the decomposition in less number of refinement steps. Moreover, after the decomposition, the system components (variables, invariants, guards, actions) are kept somewhere in one of the sub-machines. Besides, the shared variables can be refined following the decomposition strategy defined above. This is not possible in the A-style decomposition.

In addition, the *RSS* approach allows a global visibility between the different sub-systems. In other words, we reason on the global state of the system and each sub-machine is handled taking into account the other sub-machines components (variables, invariants and events). Contrary to A-style and B-style, the sub-machines are handled in a separated way and the behaviour of each sub-machine is independent from the other sub-machines.

4.4 Synthesis

Several approaches have been proposed to deal with the complex system specification issue in Event-B such as A-style and B-style. The performed analysis and study conduce to the identification of some limitations of those approaches regarding the industrial need. So, we propose a new approach: the *Refinement Seen Split* (*RSS*) based on decomposing a system into several sub-systems. A new clause REFSEES is defined to link the sub-systems to each other which allows the visibility of the state variables. This approach will ensure the preservation of invariants through the *Merge* technique.

We also define the strategy to follow for the application of the new defined approach. This strategy presents the way to decompose the state variables of the system and its events, and how to define, in each sub-system, new invariants, new state variables and new events. In addition, we demonstrate that the fact of combining -theoretically- the sub-systems constitutes a one refining component of the initial system regarding the theoretical definition of the refinement in B method. Moreover, new proof obligations are specified, through the new defined link, to ensure the behaviour preservation in each of the resulting sub-systems. For the purpose of its scaling up, the approach is applied to a railway signalling system case study.

To conclude, the *Refinement Seen Split* (*RSS*) approach allows to answer the industrial need that is based on the system/sub-systems reasoning. This need is not sufficiently covered by the existing approaches in the literature. In fact, the other approaches are based either on the partition of specific functionalities or on the split of particular behaviours. Our proposed approach is not based on the partitioning of only variables or only events but on both: the sub-systems behaviours and functionalities. Actually, the defined partition in the *Refinement Seen Split* (*RSS*) approach allows, through the decomposition strategy, to apply the decomposition in less number of refinement levels. Also, it allows to keep all the system components: variables, invariants and events. Consequently,

it does not lose the coherence of the system behaviour. In the case of *RSS*, there is no need to any external events that allow to reason locally. In A-style, the external events are defined to simulate the behaviour of the other sub-machine and then they are not taken into account in the demonstration of the composition. So, the *RSS* approach prevents the repeated occurrence of the same behaviour in different sub-machines, on the one hand. On the other hand, it avoids the guards and actions split of an event containing variables of other sub-machines because of the events atomicity. However, The fact of taking into consideration the global system to handle each sub-machine, is at the same time a difficulty for the performance of this approach.

In our approach, contrary to shared events decomposition, predicates (invariants, guards) and assignments (actions) can refer to elements that must be partitioned into different sub-components. As consequence, we don't get anymore the error "*The assignment is too complex because it refers to elements belonging to different sub-components*". An other resolved issue, is the fact of proceeding with additional refinement to avoid this error message.

This problem can be solved by passing an additional refinement step before decomposition. The user must explicitly separate the elements by this refinement by introducing an auxiliary parameter p representing the value of a variable. (Example: $v1 = v2 <=> p = v2^v1 = p$).

As far as the errors of the action act2 are concerned, no solution is proposed. The shared events decomposition Plugin is applied by partitioning the variables on two machines M1 and M2. This decomposition makes it possible to identify some limitations and inconsistencies in the behaviour of the resulting machines compared with that of the initial machine:

- The states changes of several variables in the same action is not decomposed (using the substitution "becomes such that");
- The loss of information when guards are broken down;
- The disappearance of shared invariants;
- The generation of empty events in the sub-component;
- The need for an intermediate step of manual refinement before applying the decomposition.

For shared variables decomposition, event partitioning is always possible in order to generate sub-components. However, this decomposition may be less important despite its potential: a large number of shared variables may not be of much interest, particularly for refinements that become more complex [Silva et al., 2011]. Due to the restriction of shared variables, it may be necessary to proceed with preparatory steps of refinement to resolve complex predicates (invariants, guards, axioms) or substitutions (actions) by separating the variables assigned to different sub-components. If this step is not performed, these complex predicates/assignments are automatically marked by the tool and the user intervention is required.

CONCLUSIONS AND PERSPECTIVES

Conclusions

The work presented in this thesis concerns the definition of a new approach for the decomposition in Event-B method. Its objective is to propose a methodological and operational solution to decompose a safety-critical system into several components with respect to the behaviour of the global specification. This solution must allow the designer to specify all the characteristics considered as relevant for the modelling of a system, but it must also offer the possibility of partitioning this system over the steps of refinement.

The bibliographical study, carried out on the Event-B and the approaches of the decomposition on the Event-B formal modelling, allowed us to identify the most cited and used techniques of modularisation in Event-B: decomposition by shared variables and decomposition by shared events. The first one allows to partition the system functionality and the second one decomposes the behaviour of the system. Although these approaches can split systems into multiple sub-systems, there exist some limitations and some difficulties regarding the industrial need. For instance, the difficulty of decomposing complex predicates and the need of several intermediate steps of refinement to decompose can be encountered [Abrial, 2009, Kraibi et al., 2019b].

In the context of the *PRESCOM* project, we worked on a specific industrial context: railway systems modelling. This application domain of safety-critical systems recognises the relevance of the decomposition in the railway systems modelling. It identifies some needs to separate the key features of the abstract specification system in a number of lower-level sub-systems. This separation is performed according to the intended purpose of the system modelling to get more readable and manageable specifications.

The proposition presented in this thesis for the decomposition of safety-critical system can be summarised by these four points:

– A decomposition method for partitioning systems into sub-systems, independently of any Event-B tool;

- A demonstration of the correctness of the proposed solution;
- A proposition of new additional needed proof obligations rules;
- An illustration of the *Refinement Seen Split* (*RSS*) method by its application on a concrete signalling railway system case study validated by the domain experts.

All these points are described in detail below.

The first one concerns a decomposition method for partitioning systems. In fact, the decomposition approach that we propose in this thesis allows to partition a system into multiple sub-systems. Our approach is based on the notion of refinement and the decomposing methods existing in the literature. This is done following a specific strategy in order to split correctly the invariants, variables and events of the system. The strategy defines, in general, the different cases that can occur in a model and the methodology to follow in each case. In addition, in order to keep the semantic coherence of the system behaviour, a new clause named REFSEES is defined. This clause is a semantic link between the sub-machines. It guarantees, to a certain sub-machine, the visibility of the properties, variables and invariants of the other sub-machines resulting from the decomposition of the same initial machine.

The second point tackles the correctness of the Refinement Seen Split (RSS) approach. Indeed, the formalisation of the proposed approach of decomposition requires the insurance of its correctness. As a consequence, we verify this correctness through a demonstration. Indeed, after decomposing an initial machine *M*, the merge of the resulting sub-machines *MRG* constitutes a refinement of the decomposed machine by construction. This is justified by the defined strategy to follow. It allows to keep each element of the system somewhere in the sub-machines and the clause REFSEES links between them. After this step of decomposition, each sub-machine can be refined independently, and the number of refinements varies from a sub-machine to another regarding the need. At a certain level of refinement, we demonstrate that the merge of the resulting sub-machines *MRG'* after the refinement constitutes a refinement of the first merge *MRG*. Consequently, we can deduce that the set of the sub-machine *MRG'* is a refinement of the initial machine *M*.

The third point addresses the definition of new additional proof obligation rules. Actually, the proposition of our new method of decomposition in Event-B leads to an indispensable definition of new proof obligations rules. Actually, we distinguish two types of proof obligations rules: local and global. The local proof obligations rules are those defined, classically, by the Event-B language. They are considered as internal to each sub-machine. The global proof obligations rules concern the totality of the system. In other words, it takes into consideration the merge of the sub-machines. In our work, we define new proof obligations rules for the global part. Despite the fact that a system is correct, it is not guaranteed that it does not deadlock or run indefinitely while it was not the case before. For example, a train that does not move is considered as safe, but it does not accomplish its task: transportation of passengers or goods. In practice, since we are decomposing there is a possibility to lose some behaviours of the decomposed system. Consequently, the deadlock freedom and the variant proof obligations rules are defined in order to tackle this possibility.

The fourth point deals with the application of the approach on a railway case study. In order to illustrate our contribution and compare it with the other studied methods of decomposition, we

apply our approach on the railway case study. We notice that we can apply the *Refinement Seen Split* (*RSS*) without any intermediate step of refinement in order to simplify the model. Besides, there is a communication link between those sub-machines so they can exchange the visibility of the variables states and invariants.

To summarise, refinement and decomposition are defined, in the literature, in such a way that they can coexist in the formal modelling process in order to manage the system complexity through multiple levels of abstraction. However, some existing approaches of decomposition do not rely on the refinement correctness to define the decomposition and preserve the syntactic/semantic coherence from the beginning to the lower level of modelling. In addition, the existing mechanisms of decomposition are defined as solutions associated to a specific tool *Rodin*.

We believe that certain methodological aspects relating to the Event-B method that we used such as the generation of proof obligations, the reconstruction of traces of events or the fact of making two models merge while keeping the proofs already done on these models can be reused in another model design in Event-B. It is independent on any existing tool.

Many methods have been proposed for the decomposition of the systems for Event-B. But there are few complex case studies where these techniques have been applied. We believe we have made a contribution in this direction with the decomposition of safety-critical systems.

Perspectives

The work presented in this thesis allows many research opportunities with innovative ideas that have the potential to enrich the literature. The *Refinement Seen Split* (*RSS*) approach should be applied in different types of projects for the purpose of its industrialisation and scaling up. The use of this approach in other industrial sectors can also be discussed in a future work. The implementation of the proposed decomposition technique, on the basis of the decomposition strategy, can facilitate its application on the models, especially on the big and complex systems, as well as the generation of their associated proof obligation rules.

Actually, in this work we provide a demonstration to justify that these proof obligations, when they are discharged, guarantee that the composed/merged machines, resulting from the decomposition using the *RSS* method, refine well and correctly the decomposed abstract machine. For a better understanding of these proof obligations, it would be interesting for future works to establish a formal demonstration and to reinforce this demonstration with the support tools of the method. Thus, the formalisation of this approach using Event-B theories or within a proof assistant will make it possible to obtain greater confidence in this approach.

This technique can also be automatised, in the form of plugin for instance. In addition, the decomposition can be guided by its combination with other existing techniques in the literature. These perspectives are described in details in the following.

A short-term perspective relates to the application of our approach of decomposition on other case studies from other railway projects, in the Autonomous Train project for instance, or even on specifications from different industrial areas. This allows a better understanding of the models. Furthermore, it can help to identify some area of improvement of the *RSS* approach.

Moreover, another perspective concerns the implementation of the approach in Atelier B tool,

including the new defined notions such as the clause REFSEES, as well as the integration of the new defined proof obligations. Furthermore, after its implementation, it can be possible to automatise the decomposition. Indeed, the Event-B decomposition strategy, that we propose in this thesis work, is proceeded manually on the models. So, it is planned to automatise the different decomposition modelling stages in an Event-B tool like Atelier B.

In most cases, the input requirement documents, provided by clients, are not sufficient. It is either lacking some details of the system behaviour or misunderstood by the model designers. So, these documents should be rewritten, after analysis and understanding, into reference documents, where everything is made clear and properly labelled for traceability. By nature, human is familiarised, mostly, with documents that refers to the visual supports. Hence, the decomposition of Event-B models can be guided by some graphical methods. Although they lack some semantics definitions, it is a way to facilitate the design step thanks to their visual supports. These methods can be an intermediate step and can be validated by the domain experts before the modelling of the system and its sub-systems. For example, the model transformation from UML to Event-B have been discussed in a previous work [Kraibi et al., 2019a]. This technique facilitates the behaviour modelling on Event-B through some behavioural diagram on UML, for instance, it allows to model the events sequencing.

Among the graphical methods that can be used in the future is the *Combination with the SysML-/KAOS method*. In fact, SysML-KAOS is a requirement engineering approach based on goals hierarchy [Tueno et al., 2017]. It represents the requirement document in the form of a graph. the root is the principal requirement. The children of each node are more detailed requirements of this node. These children can be linked with the parent by a specific link. This later determines the sequence, conjunction or disjunction of the requirements.
In the context of our approach of decomposition, this technique can be used to easily structure and gradually enrich the Event-B machine and its sub-machines. Actually, in parallel with our thesis work, a formalisation of SysML/KAOS requirement through and Event-B system decomposition have been discussed in [Tueno Fotso, 2019]. This formalisation is based on other decomposition technique that uses the shared variable decomposition strategy and the notion of interfaces. However, the decomposition method proposed in this work face some difficulties, compared to our approach. For example, the variables of an interface are defined as constants regarding the other interfaces, so it is difficult to animate/model-check the formal model. This approach difficulties are detailed in [Tueno Fotso, 2019].

Another perspective that can be considered is the *Combination with the Events Refinement Structure (ERS) Technique*: ERS is an Event-B approach based on events [Butler, 2009a]. It is also represented by a graph, but using the events of an Event-B model (see details in 2.2.1). This method may orient the Event-B modular modelling with the help of its notions: *AND*, *OR*, dashed link, etc.

BIBLIOGRAPHY

[Abrial et al., 2005] Abrial, J., Hallerstede, S., Mehta, F., Métayer, C., and Voisin, L. (2005). Specification of basic tools and platform. *RODIN Deliverable D10.* *cited in pages: 10 and 26*

[Abrial, 1996] Abrial, J.-R. (1996). *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, New York, NY, USA. *cited in pages: 12, 13, 24, 40, 94 and 101*

[Abrial, 2002] Abrial, J.-R. (2002). Discrete System Models. *Internal Notes (www-lsr.imag.fr/B).* *cited in pages: 61 and 101*

[Abrial, 2009] Abrial, J.-R. (2009). Event Model Decomposition. In *Technical Report/[ETH], Department of Computer Science*, volume 626. ETH Zurich. *cited in pages: 61, 64, 82, 83, 84, 91, 101, 116 and 174*

[Abrial, 2010] Abrial, J.-R. (2010). *Modeling in Event-B: System and Software Engineering.* Cambridge University Press. *cited in pages: 7, 12, 13, 40, 53, 56, 57, 58, 59, 64 and 101*

[Abrial et al., 2010] Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T. S., Mehta, F., and Voisin, L. (2010). Rodin: an Open Toolset for Modelling and Reasoning in Event-B. In *International Journal on Software Tools for Technology Transfer*, volume 12, pages 447–466. Springer. *cited in pages: 29 and 40*

[Abrial and Hallerstede, 2007] Abrial, J.-R. and Hallerstede, S. (2007). Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. In *Fundamenta Informaticae*, volume 77, pages 1–28. IOS Press. *cited in pages: 15, 57, 60, 69 and 94*

[Abrial et al., 1991] Abrial, J.-R., Lee, M. K., Neilson, D., Scharbach, P., and Sørensen, I. H. (1991). The B-Method. In *International Symposium of VDM Europe*, pages 398–405. Springer. *cited in pages: 24 and 51*

[Alkhammash et al., 2015] Alkhammash, E., Butler, M., Fathabadi, A. S., and Cîrstea, C. (2015). Building Traceable Event-B Models from Requirements. In *Science of Computer Programming*, volume 111, pages 318–338. Elsevier. *cited in pages: 7 and 53*

[Andrews, 1992] Andrews, D. (1992). VDM Specification Language Proto-Standard. Draft Standard ISO. Technical report, IEC JTC1/SC22/WG19 I-246, ISO. *cited in pages: 12 and 13*

[Atelier B, 2018] Atelier B (2018). ClearSy System Engineering. *Aix-en-Provence, France, available for download [online] http://www.atelierb.eu/en/download-atelier-b.* *cited in page: 24*

[Back, 1989] Back, R.-J. (1989). Refinement Calculus, Part II: Parallel and Reactive Programs. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, pages 67–93. Springer. *cited in page: 51*

[Behm et al., 1999] Behm, P., Benoit, P., Faivre, A., and Meynadier, J. M. (1999). Météor: A Successful Application of B in a Large Project. In Wing, J., Woodcock, J., and Davies, J., editors, *FM'99 - Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 369–387. Springer Berlin Heidelberg. *cited in pages: 24, 34, 37 and 40*

[Ben Ayed, 2016] Ben Ayed, R. (2016). *Modélisation UML/B pour la Validation des Exigences de Sécurité des Règles d'Exploitation Ferroviaires.* PhD thesis, Ecole Centrale de Lille. *cited in pages: 21 and 34*

[Ben Ayed et al., 2014] Ben Ayed, R., Collart-Dutilleul, S., Bon, P., Idani, A., and Ledru, Y. (2014). B Formal Validation of ERTMS/ETCS Railway Operating Rules. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 124–129. Springer. *cited in pages: 35 and 40*

[Ben Ayed et al., 2016] Ben Ayed, R., Collart-Dutilleul, S., and Prun, E. (2016). "Formal Methods To Tailored Solution For Single Track Low Traffic French Lines". In *International Railway Safety Council (IRSC), Paris, France.* *cited in page: 40*

[Benaissa, 2010] Benaissa, N. (2010). La Composition des Protocoles de Sécurité avec la Méthode B événementielle. *Thèse de doctorat de l'Université Henri Poincaré Nancy 1.* *cited in page: 25*

[Benaissa et al., 2016] Benaissa, N., Bonvoisin, D., Feliachi, A., and Ordioni, J. (2016). The PERF Approach for Formal Verification. In *International Conference on Reliability, Safety, and Security of Railway Systems*, pages 203–214. Springer. *cited in page: 34*

[Bézivin and Gerbé, 2001] Bézivin, J. and Gerbé, O. (2001). Towards a Precise Definition of the OMG/MDA Framework. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pages 273–280. IEEE. *cited in page: 31*

[Bjøner, 1987] Bjøner, D. (1987). On the Use of Formal Methods in Software Development. In *Proceedings of the 9th International Conference on Software Engineering*, pages 17–29. *cited in page: 23*

[Bjøner, 2019] Bjøner, D. (2019). Domain Analysis and Description Principles, Techniques, and Modelling Languages. volume 28, pages 1–67. ACM New York, NY, USA. *cited in page: 22*

[Bjrnerϕ and Havelund, 2014] Bjrnerϕ, D. and Havelund, K. (2014). 40 Years of Formal Methods-Some Obstacles and Some Possibilities? *FM.* *cited in page: 13*

[Blakstad, 2006] Blakstad, H. C. (2006). Revising Rules and Reviving Knowledge. Adapting Hierarchical and Risk Based Approaches to Safety Rule Modifications in the Norwegian Railway System. *cited in page: 31*

[Bolusset and Oquendo, 2002] Bolusset, T. and Oquendo, F. (2002). Formal Refinement of Software Architectures Based on Rewriting Logic. In *ZB2002 International Workshop on Refinement of Critical Systems: Methods, Tools and Experience, Grenoble*, volume 29, pages 1–20. *cited in page: 51*

[Bon et al., 2013] Bon, P., Collart-Dutilleul, S., and Sun, P. (2013). Study of implementation of ertms with respect to french national rules using a b centred methodology. In *Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM)*, pages 1–5. IEEE. *cited in page: 34*

[Bonvoisin, 2016] Bonvoisin, D. (2016). 25 Years of Formal Methods at RATP. *International Railway Safety Council (IRSC)*. *cited in page: 34*

[Bonvoisin and Benaissa, 2015] Bonvoisin, D. and Benaissa, N. (2015). Utilisation de la méthode de preuve formelle perf de la ratp sur le projet peee. *Revue générale des chemins de fer*, 250. *cited in page: 34*

[Boulanger, 2012] Boulanger, J. L. (2012). Industrial use of formal methods: Formal verification. *ISTE Ltd and John Wiley and Sons, Inc.* *cited in page: 25*

[Bowen and Stavridou, 1993] Bowen, J. and Stavridou, V. (1993). Safety-critical systems, formal methods and standards. *Software engineering journal*, 8(4):189–209. *cited in page: 12*

[Brien et al., 1992] Brien, S. M., Nicholls, J. E., et al. (1992). *Z base standard: Version 1.0*. Oxford University Computing Laboratory, Programming Research Group. *cited in pages: 12 and 13*

[Bushell and Stonham, 1985] Bushell, C. and Stonham, P. (1985). Jane's urban transport systems 1985. Technical report. *cited in page: 31*

[Butler, 1997] Butler, M. (1997). An approach to the design of distributed systems with b amn. In *International Conference of Z Users*, pages 221–241. Springer. *cited in page: 68*

[Butler, 2009a] Butler, M. (2009a). Decomposition Structures for Event-B. In *International Conference on Integrated Formal Methods*, pages 20–38. Springer. *cited in pages: 15, 40, 51, 52, 68, 94 and 119*

[Butler, 2009b] Butler, M. (2009b). Incremental Design of Distributed Systems with Event-B. *Engineering Methods and Tools for Software Safety and Security*, 22(131). *cited in page: 52*

[Butler and Hallerstede, 2007] Butler, M. and Hallerstede, S. (2007). The Rodin formal modelling tool. In *BCS-FACS Christmas 2007 Meeting-Formal Methods In Industry, London.* *cited in pages: 30 and 40*

[Butler et al., 2020] Butler, M., Körner, P., Krings, S., Lecomte, T., Leuschel, M., Mejia, L.-F., and Voisin, L. (2020). The First Twenty-Five Years of Industrial Use of the B-Method. In *International Conference on Formal Methods for Industrial Critical Systems*, pages 189–209. Springer. *cited in page: 14*

[Cannon et al., 2003] Cannon, D., Edel, K.-O., Grassie, S., and Sawley, K. (2003). Rail defects: an overview. *Fatigue & Fracture of Engineering Materials & Structures*, 26(10):865–886. *cited in page: 22*

[CENELEC EN50126, 2001] CENELEC EN50126 (2001). 50126: Railway Applications - The Specification and Demonstration of Reliability. *Availability, Maintainability and Safety (RAMS)*. *cited in pages: 30 and 32*

[CENELEC EN50128, 2011] CENELEC EN50128 (2011). 50128. *Railway applications-Communication, Signaling and Processing Systems-Software for Railway Control and Protection Systems*. *cited in pages: 12, 23, 30, 31 and 32*

[CENELEC EN50129, 1998] CENELEC EN50129 (1998). 50129. *Railway Applications: Safety Related Electronic Systems for Signalling, European Committee for Electrotechnical Standardization (CENELEC)*. *cited in pages: 30 and 32*

[ClearSy, 2020] ClearSy (2020). "ClearSy Systems Engineering". *http://clearsy. com/*. *cited in page: 24*

[Dghaym et al., 2017] Dghaym, D., Butler, M., and Fathabadi, A. S. (2017). Extending ERS for Modelling Dynamic Workflows in Event-B. In *Engineering of Complex Computer Systems (ICECCS), 2017 22nd International Conference on*, pages 20–29. IEEE. *cited in pages: 51 and 52*

[Dghaym et al., 2016] Dghaym, D., Trindade, M. G., Butler, M., and Fathabadi, A. S. (2016). A Graphical Tool for Event Refinement Structures in Event-B. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 269–274. Springer. *cited in pages: 51 and 52*

[Dollé et al., 2003] Dollé, D., Essamé, D., and Falampin, J. (2003). B dans le transport ferroviaire: L'expérience de siemens transportation systems. *TSI. Technique et science informatiques*, 22(1):11–32. *cited in page: 34*

[Dupuy, 2000] Dupuy, S. (2000). *Integrating semi-formal and formal notations for information system specification*. PhD thesis, Joseph Fourier University, Grenoble, France. *cited in page: 22*

[ESA, 1991] ESA (1991). 05-0 European Space Agency software engineering standards. *cited in page: 12*

[Fantechi et al., 2013] Fantechi, A., Fokkink, W., and Morzenti, A. (2013). Some Trends in Formal Methods Applications to Railway Signaling. *Formal Methods for Industrial Critical Systems*, pages 61–84. *cited in page: 23*

[Fathabadi et al., 2011] Fathabadi, A. S., Rezazadeh, A., and Butler, M. (2011). Applying Atomicity and Model Decomposition to a Space Craft System in Event-B. In *NASA Formal Methods Symposium*, pages 328–342. Springer. *cited in pages: 51 and 52*

[Guiho and Hennebert, 1990] Guiho, G. and Hennebert, C. (1990). SACEM Software Validation. In *[1990] Proceedings. 12th International Conference on Software Engineering*, pages 186–191. IEEE. *cited in page: 37*

[Hennebert and Guiho, 1993] Hennebert, C. and Guiho, G. (1993). Sacem: A fault tolerant system for train speed control. In *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*, pages 624–628. IEEE. *cited in page: 34*

[Hoang and Abrial, 2010] Hoang, T. S. and Abrial, J.-R. (2010). Event-B Decomposition for Parallel Programs. In *International Conference on Abstract State Machines, Alloy, B and Z*, pages 319–333. Springer. *cited in pages: 15, 60 and 94*

[Hoang et al., 2017] Hoang, T. S., Dghaym, D., Snook, C., and Butler, M. (2017). A composition mechanism for refinement-based methods. In *2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 100–109. IEEE. *cited in page: 69*

[Hoang et al., 2011] Hoang, T. S., Iliasov, A., Silva, R. A., and Wei, W. (2011). A Survey on Event-B Decomposition. *Electronic Communications of the EASST*, 46. *cited in pages: 15, 60, 69 and 94*

[Idani, 2006] Idani, A. (2006). *B/UML: Mise en relation de spécifications B et de descriptions UML pour l'aide à la validation externe de développements formels en B.* PhD thesis. *cited in pages: 9, 22 and 23*

[Idani et al., 2019] Idani, A., Ledru, Y., Wakrime, A. A., Ayed, R. B., and Collart-Dutilleul, S. (2019). Incremental development of a safety critical system combining formal methods and dsmls. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 93–109. Springer. *cited in page: 35*

[IEC 61508, 2010] IEC 61508 (2010). Functional safety of electrical/electronic/programmable electronic safety-related systems. *International Electrotechnical Commission, Geneva, Switzerland, December*. *cited in page: 10*

[IEEE 729, 1983] IEEE 729 (1983). Ieee standard glos., ary of software engineering terminology. Technical report, Technical Report IEEE Std 729-1983, The Institute of Electrical and .... *cited in page: 13*

[ISO 15288, 2002] ISO 15288 (2002). Iso 15288 systems engineering—system life cycle processes. *International Standards Organisation*. *cited in page: 13*

[ISO/TC269/SC1, 2017] ISO/TC269/SC1 (2017). Standard catalogue (infrastructure). *Switzerland: International Organization for Standardization*. *cited in page: 30*

[ISO/TC269/SC2, 2015] ISO/TC269/SC2 (2015). Standard catalogue (rolling stock). *Switzerland: International Organization for Standardization*. *cited in page: 30*

[Jackson, 2006] Jackson, D. (2006). Software Abstractions: Logic. *Language, and Analysis. MIT press*, 2012. *cited in page: 13*

[Jones, 1990] Jones, C. B. (1990). *Systematic software development using VDM*, volume 2. Prentice Hall Englewood Cliffs. *cited in page: 24*

[Kempen, 1993] Kempen, B. J. (1993). Modular track section for an endless conveyor. US Patent 5,178,263. *cited in page: 32*

[Kraibi et al., 2019a] Kraibi, K., Ayed, R. B., Collart-Dutilleul, S., Bon, P., and Petit, D. (2019a). Analysis and Formal Modeling of Systems Behavior Using UML/Event-B. *journal of communications*, 14(10):pp980–986. *cited in pages: 34 and 119*

[Kraibi et al., 2019b] Kraibi, K., Ayed, R. B., Rehm, J., Dutilleul, S. C., Bon, P., and Petit, D. (2019b). Event-B Decomposition Analysis for Systems Behavior Modeling. In *ICSOFT*, pages 278–286. *cited in pages: 94, 116 and 174*

[Lecomte et al., 2007] Lecomte, T., Servat, T., Pouzancre, G., et al. (2007). Formal methods in safety-critical railway systems. In *10th Brasilian symposium on formal methods*, pages 29–31. *cited in page: 32*

[Leuschel and Butler, 2003] Leuschel, M. and Butler, M. (2003). ProB: A Model Checker for B. In *FME*, volume 2805, pages 855–874. Springer. *cited in pages: 30, 45 and 81*

[Metayer et al., 2005] Metayer, C., Abrial, J.-R., and Voisin, L. (2005). Rodin deliverable 3.2: Event-b language. In *Tech. Rep. Project IST-511599*. School of Computing Science, University of Newcastle. *cited in page: 57*

[OMG, 2011] OMG (2011). Unified Modeling Language (OMG UML), superstructure. Technical report, version 2.4. 1. Tech. rep., Object Management Group. *cited in page: 22*

[Patin, 2006] Patin, F. (2006). Un Outil de Modélisation des Systèmes. In *https://www.methode-b.com/wp-content/uploads/sites/7/2013/06/CompoSys-AFADL06_17_03_2006.pdf*. Journées Approches Formelles Dans l'Assistance au Développement de Logiciels (AFADL). Clearsy. *cited in page: 24*

[Pawlik, 2015] Pawlik, M. (2015). Control command systems impact on the railway operational safety. *Science and Transport Progress. Bulletin of Dnipropetrovsk National University of Railway Transport*, (2 (56)):58–64. *cited in page: 31*

[Pouzancre and Servat, 2005] Pouzancre, G. and Servat, T. (2005). Application Industrielle de la Méthode Formelle B. In *https://www.clearsy.com/wp-content/uploads/sites/3/pdf/documents/ClearsyCEAT-01-11-2005.pdf*. Clearsy. *cited in page: 24*

[Sabatier, 2016] Sabatier, D. (2016). Using formal proof and B method at system level for industrial projects. In *International Conference on Reliability, Safety and Security of Railway Systems*, pages 20–31. Springer. *cited in page: 40*

[Sadoun, 2014] Sadoun, D. (2014). *Des spécifications en langage naturel aux spécifications formelles via une ontologie comme modèle pivot.(From natural language specifications to formal specifications via an ontology as a pivot model).* PhD thesis, University of Paris-Sud, Orsay, France. *cited in page: 21*

[Schön, 2013a] Schön, W. (2013a). Signalisation et automatismes ferroviaires-tome 2. *Université de technologie de Compiègne.* *cited in page: 31*

[Schön, 2013b] Schön, W. (2013b). Signalisation et automatismes ferroviaires-tome 3. *Université de technologie de Compiègne.* *cited in page: 32*

[Siala et al., 2016] Siala, B., Tahar Bhiri, M., Bodeveix, J.-P., and Filali, M. (2016). Un processus de Développement Event-B pour des Applications Distribuées. *Université de Franche-Comté*. *cited in pages: 15, 69 and 94*

[Silva and Butler, 2010] Silva, R. and Butler, M. (2010). Shared Event Composition/Decomposition in Event-B. In *International Symposium on Formal Methods for Components and Objects*, pages 122–141. Springer. *cited in page: 59*

[Silva et al., 2011] Silva, R., Pascal, C., Hoang, T. S., and Butler, M. (2011). Decomposition tool for event-b. *Software: Practice and Experience*, 41(2):199–208. *cited in pages: 40, 68, 83, 85 and 115*

[Snook and Butler, 2006] Snook, C. and Butler, M. (2006). UML-B: Formal Modeling and Design Aided by UML. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(1):92–122. *cited in page: 10*

[Spivey and Abrial, 1992] Spivey, J. M. and Abrial, J. (1992). *The Z notation*. Prentice Hall Hemel Hempstead. *cited in page: 24*

[Sun et al., 2015] Sun, P., Bon, P., and Collart-Dutilleul, S. (2015). A joint development of coloured petri nets and the b method in critical systems. *cited in page: 34*

[Sun et al., 2014] Sun, P., Collart-Dutilleul, S., and Bon, P. (2014). A formal modeling methodology of the french railway interlocking system via hcpn. *WIT Transactions on The Built Environment*, 135:849–858. *cited in page: 34*

[Tueno et al., 2017] Tueno, S., Laleau, R., Mammar, A., and Frappier, M. (2017). The SysML/KAOS Domain Modeling Approach. *arXiv preprint arXiv:1710.00903*. *cited in page: 119*

[Tueno Fotso, 2019] Tueno Fotso, S. J. (2019). Vers une approche formelle d'ingénierie des exigences outillée et éprouvée. *Doctoral Dissertation*. *cited in pages: 73 and 119*

[Van Deursen et al., 2000] Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36. *cited in page: 69*

APPENDIX A

EVENT-B MODEL OF THE CASE STUDY: TRAIN CONTROL SYSTEM

A.1 Abstract Machine M0: Introduction of Trains Movements

MACHINE M0

VARIABLES

front_trainA // position of the front of tainA

end_trainA // position of the end of tainA

front_trainB // position of the front of tainB

end_trainB // position of the end of tainB

INVARIANTS

inv1: $front\_trainA \in \mathbb{N}$

inv2: $front\_trainB \in \mathbb{N}$

inv3: $end\_trainA \in \mathbb{N}$

inv4: $end\_trainB \in \mathbb{N}$

// for each train, the position of the train end must be lower than the position of the train front

inv5: $end\_trainA < front\_trainA$

inv6: $end\_trainB < front\_trainB$

// since trainA is following trainB, the position of trainA front must be lower than the position of trainB end

inv7: $front\_trainA < end\_trainB$

EVENTS

Initialisation

begin

act1:

$front\_trainA$,
end_trainA,
$front\_trainB$,
end_trainB :|
(
$front\_trainA' \in \mathbb{N}$
$\wedge\, end\_trainA' \in \mathbb{N}$
$\wedge\, front\_trainB' \in \mathbb{N}$
$\wedge\, end\_trainB' \in \mathbb{N}$
$\wedge\, end\_trainA' < front\_trainA'$
$\wedge\, end\_trainB' < front\_trainB'$
$\wedge\, front\_trainA' < end\_trainB'$
)

end

// movement of the front of trainA

Event move_front_trainA ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $front\_trainA + step < end\_trainB$

then

act1: $front\_trainA := front\_trainA + step$

end

// movement of the end of trainA

Event move_end_trainA ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainA + step < front\_trainA$

then

act1: $end\_trainA := end\_trainA + step$

end

// movement of the front of trainB

Event move_front_trainB ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$

then

act1: $front\_trainB := front\_trainB + step$

end

// movement of the end of trainB

Event move_end_trainB ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainB + step < front\_trainB$

then

act1: $end\_trainB := end\_trainB + step$

end

END

A.2 First Refinement: Definition of Blocks

A.2.1 Context of Blocks

CONTEXT Block

SETS

BlockState // a bloc state can be free or occupied

SUBSYS // track or train

CONSTANTS

Block

front_block // the front limit of the block

end_block // the rear limit of the block

next_block //the function defininf the next for each bloc

Free

Occupied

TRAIN

TRACK

AXIOMS

axm1: $BlockState = \{Free, Occupied\}$

axm2: $SUBSYS = \{TRAIN, TRACK\}$

axm3: $Block = \mathbb{N}$ // blocks are defined as Naturel

axm4: $front\_block \in Block \rightarrow \mathbb{N}$

axm5: $end\_block \in Block \rightarrow \mathbb{N}$

axm6: $next\_block = (\lambda bk \cdot bk \in Block | bk + 1)$

// the rear limit of a block is lower than its front limit

axm7:

$\forall bk \cdot ($

$bk \in Block$

$\Rightarrow end\_block(bk) < front\_block(bk)$

$)$

// the front limit of a block is equal to the rear limit of the next block

axm8:

$\forall b1, b2 \cdot ($

$b1 \in Block$

$\wedge\, b2 \in Block$

$\wedge\, next\_block(b1) = b2$

$\Rightarrow front\_block(b1) = end\_block(b2)$

$)$

axm9: $\langle$theorem$\rangle$ $ran(next\_block) = Block \setminus \{0\}$

END

A.2.2 Refinement Machine: M1

MACHINE M1
REFINES M0
SEES Block
VARIABLES

front_trainA
front_trainB
end_trainA
end_trainB
fst_tAblock // the block occupied by the head of trainA
lst_tAblock // the block occupied by the end of trainA
fst_tBblock // the block occupied by the head of trainB
lst_tBblock // the block occupied by the end of trainB
block_state // a function that gives the state of each block
next_turn // this variable allows to identify which events should be triggered: the trains events or the track events

INVARIANTS

inv1: $fst\_tAblock \in Block$
inv2: $lst\_tAblock \in Block$
inv3: $fst\_tBblock \in Block$
inv4: $lst\_tBblock \in Block$
inv5: $block\_state \in Block \rightarrow BlockState$
inv6: $next\_turn \in SUBSYS$

// for each train, the occupied block by the end of the train must be behind or equal to the occupied block by the front of the train

inv7: $lst\_tAblock <= fst\_tAblock$
inv8: $lst\_tBblock <= fst\_tBblock$

// since trainA is following trainB, the block occupied by trainA front must be behind the block occupied by trainB end

inv9: $fst\_tAblock < lst\_tBblock$

// the end of trainB must be limited by the end of its last occupied block

inv10: $end\_block(lst\_tBblock) <= end\_trainB$

// when it is the turn of the trains movements, trainB shouldn't be on the released blocks behind it

inv11:

$\forall bk \cdot ($
$bk \in Block$
$\wedge\ next\_turn = TRAIN$
$\wedge\ block\_state(bk) = Free$
$\Rightarrow bk \neq lst\_tBblock)$

EVENTS

Initialisation

begin

act1:

$front\_trainA,$
$front\_trainB,$
$end\_trainA,$
$end\_trainB,$
$fst\_tAblock,$
$lst\_tAblock,$
$fst\_tBblock,$
$lst\_tBblock,$
$block\_state,$
$next\_turn :| ($
$front\_trainA' \in \mathbb{N}$
$\wedge\ front\_trainB' \in \mathbb{N}$
$\wedge\ end\_trainA' \in \mathbb{N}$
$\wedge\ end\_trainB' \in \mathbb{N}$
$\wedge\ end\_trainA' < front\_trainA'$
$\wedge\ end\_trainB' < front\_trainB'$
$\wedge\ front\_trainA' < end\_trainB'$
$\wedge\ block\_state' \in Block \rightarrow BlockState$
$\wedge\ fst\_tAblock' \in Block$
$\wedge\ lst\_tAblock' \in Block$
$\wedge\ fst\_tBblock' \in Block$
$\wedge\ lst\_tBblock' \in Block$
$\wedge\ next\_turn' \in SUBSYS$
$\wedge\ lst\_tAblock' <= fst\_tAblock'$
$\wedge\ lst\_tBblock' <= fst\_tBblock'$
$\wedge\ fst\_tAblock' < lst\_tBblock'$
$\wedge\ end\_block(lst\_tBblock') <= end\_trainB'$
$\wedge\ \forall bk \cdot ($
$bk \in Block$
$\wedge\ next\_turn' = TRAIN$
$\wedge\ block\_state'(bk) = Free$
$\Rightarrow bk \neq lst\_tBblock'))$

end

//
////////////////////// Train Behavior ////////////////////////////////
//

// the movement of trainA front inside a block (without occupying any new block)

Event move_front_trainA ⟨ordinary⟩ $\widehat{=}$

refines move_front_trainA

any

step

where

grd1: $step \in \mathbb{N}_1$

// the front of trainA must not overstep the front limit of the block occupied by the head of this train

grd2: $front\_trainA + step < front\_block(fst\_tAblock)$

grd3: $next\_turn = TRAIN$

then

act1: $front\_trainA := front\_trainA + step$

act2: $next\_turn := TRACK$

end

// the movement of trainA end inside a block (without freeing any block)

Event move_end_trainA ⟨ordinary⟩ $\widehat{=}$

refines move_end_trainA

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainA + step < front\_trainA$

// the end of trainA must not overstep the front limit of the block occupied by the tail of this train

grd3: $end\_trainA + step < front\_block(lst\_tAblock)$

grd4: $next\_turn = TRAIN$

then

act1: $end\_trainA := end\_trainA + step$

act2: $next\_turn := TRACK$

end

// block freeing by trainA

Event free_tAblock ⟨ordinary⟩ $\widehat{=}$

refines move_end_trainA

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainA + step < front\_trainA$
// the end of trainA must overstep the front limit of the block occupied by the tail of this train
grd3: $front\_block(lst\_tAblock) < end\_trainA + step$
// the new position of trainA end must be inside the next block
grd4: $end\_trainA + step < front\_block(next\_block(lst\_tAblock))$
grd5: $next\_block(lst\_tAblock) <= fst\_tAblock$
grd6: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$
act2: $end\_trainA := end\_trainA + step$
act3: $lst\_tAblock := next\_block(lst\_tAblock)$

end

// block occupying by trainA

Event Enter_tAblock ⟨ordinary⟩ $\widehat{=}$

refines move_front_trainA

any

step

where

grd1: $step \in \mathbb{N}_1$
// the block in front of trainA must be free
grd2: $block\_state(next\_block(fst\_tAblock)) = Free$
// the new position of trainA front must be inside the new occupied block
grd3: $front\_block(fst\_tAblock) < front\_trainA + step$
grd4: $front\_trainA + step < front\_block(next\_block(fst\_tAblock))$
grd5: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$
act2: $front\_trainA := front\_trainA + step$
act3: $fst\_tAblock := next\_block(fst\_tAblock)$

end

// the movement of trainB front inside a block (without occupying any new block)

Event move_front_trainB ⟨ordinary⟩ $\widehat{=}$

refines move_front_trainB

any

step

where

grd1: $step \in \mathbb{N}_1$
// the front of trainB must not overstep the front limit of the block occupied by the head of this train
grd2: $front\_trainB + step < front\_block(fst\_tBblock)$
grd3: $next\_turn = TRAIN$

then

act1: $front\_trainB := front\_trainB + step$

act2: $next\_turn := TRACK$

end

// the movement of trainB end inside a block (without freeing any block)

Event move_end_trainB ⟨ordinary⟩ $\widehat{=}$

refines move_end_trainB

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainB + step < front\_trainB$

// the end of trainB must not overstep the front limit of the block occupied by the tail of this train

grd3: $end\_trainB + step < front\_block(lst\_tBblock)$

grd4: $next\_turn = TRAIN$

then

act1: $end\_trainB := end\_trainB + step$

act2: $next\_turn := TRACK$

end

// block freeing by trainB

Event free_tBblock ⟨ordinary⟩ $\widehat{=}$

refines move_end_trainB

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainB + step < front\_trainB$

// the end of trainB must overstep the front limit of the block occupied by the tail of this train

grd3: $front\_block(lst\_tBblock) < end\_trainB + step$

// the new position of trainB end must be inside the next block

grd4: $end\_trainB + step < front\_block(next\_block(lst\_tBblock))$

grd5: $next\_block(lst\_tBblock) <= fst\_tBblock$

grd6: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$

act2: $end\_trainB := end\_trainB + step$

act3: $lst\_tBblock := next\_block(lst\_tBblock)$

end

// block occupying by trainB

Event Enter_tBblock ⟨ordinary⟩ $\widehat{=}$

refines move_front_trainB

any

step

where

grd1: $step \in \mathbb{N}_1$

// the block in front of trainB must be free

grd2: $block\_state(next\_block(fst\_tBblock)) = Free$

// the new position of trainB front must be inside the new occupied block

grd3: $front\_block(fst\_tBblock) < front\_trainB + step$

grd4: $front\_trainB + step < front\_block(next\_block(fst\_tBblock))$

grd5: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$

act2: $front\_trainB := front\_trainB + step$

act3: $fst\_tBblock := next\_block(fst\_tBblock)$

end

//

////////////////////// Track Behavior ////////////////////////////////

//

// block state changes

Event TRACKevent ⟨ordinary⟩ $\widehat{=}$

when

grd1: $next\_turn = TRACK$

then

act1: $next\_turn := TRAIN$

act2: $block\_state := (Block \times \{Free\}) \lhd ((lst\_tAblock \mathinner{..} fst\_tAblock \cup lst\_tBblock \mathinner{..} fst\_tBblock) \times \{Occupied\})$

end

END

A.2.3 Animation of the Refinement Machine

ProB File for the Animation

```
MACHINE M1_prob
SETS /* enumerated */
  SUBSYS={TRAIN,TRACK};
  BlockState={Free,Occupied}
CONCRETE_CONSTANTS
  next_block,
  front_block,
  end_block,
  Block3
ABSTRACT_VARIABLES
  lst_t1block,
  fst_t1block,
  fst_t2block,
  next_turn,
  lst_t2block,
  block_state,
  front_train2,
  front_train1,
  end_train1,
  end_train2
/* PROMOTED OPERATIONS
  move_front_train1_TRNevt,
  move_end_train1_TRNevet,
  free_t1block_TRNevt,
  Enter_t1block_TRNevt,
  move_front_train2_TRNevt,
  move_end_train2_TRNevt,
  free_t2block_TRNevt,
  Enter_t2block_TRNevt,
  TRACKevent */
AXIOMS
    /* @track:axm3 */ Block = ℕ
  & /* @track:axm4 */ front_block ∈ Block → ℕ
  & /* @track:axm5 */ end_block ∈ Block → ℕ
  & /* @track:axm6 */ next_block = (λbk.bk ∈ Block|bk + 1)
  & /* @track:axm7 */ ∀bk·(bk ∈ Block ∩ 0 .. 10 ⇒ end_block(bk) <
  front_block(bk))
  & /* @track:axm8 */ ∀b1·(b1 ∈ Block ∩ 0 .. 10 ⇒ /* @track:axm8 */ ∀b2·(b2
  ∈ Block ∩ 0 .. 10 ∧ next_block(b1) = b2 ⇒ front_block(b1) =
  end_block(b2)))
  & /* @track_prob:prob1 */ end_block = (λb.b ∈ Block|b * 2)
  & /* @track_prob:prob2 */ front_block = (λb.b ∈ Block|(b + 1) * 2)
INVARIANT
    /* @Modele00_r1:inv1 */ fst_t1block ∈ Block
  & /* @Modele00_r1:inv2 */ lst_t1block ∈ Block
  & /* @Modele00_r1:inv3 */ fst_t2block ∈ Block
  & /* @Modele00_r1:inv4 */ lst_t2block ∈ Block
  & /* @Modele00_r1:inv5 */ block_state ∈ Block → BlockState
  & /* @Modele00_r1:inv7 */ lst_t1block ≤ fst_t1block
  & /* @Modele00_r1:inv8 */ lst_t2block ≤ fst_t2block
```

```
48    & /* @Modele00_r1:inv9 */ fst_t1block < lst_t2block
49    & /* @Modele00_r1:inv10 */ end_block(lst_t2block) ≤ end_train2
50    & /* @Modele00_r1:inv11 */ (
51       next_turn = TRAIN
52       ⇒
53       /* @Modele00_r1:inv11 */ ∀bk·(bk ∈ Block ∧ block_state(bk) = Free ⇒
 •       bk ≠ lst_t2block)
54      )
55    & /* @Modele00:inv1 */ front_train1 ∈ ℕ
56    & /* @Modele00:inv2 */ front_train2 ∈ ℕ
57    & /* @Modele00:inv3 */ end_train1 ∈ ℕ
58    & /* @Modele00:inv4 */ end_train2 ∈ ℕ
59    & /* @Modele00:inv5 */ end_train1 < front_train1
60    & /* @Modele00:inv6 */ end_train2 < front_train2
61    & /* @Modele00:inv7 */ front_train1 < end_train2
62  THEOREMS
63       /* @track:axm9 */ ran(next_block) = Block \ {0}
64  INITIALISATION
65      EVENT /* of machine Modele00_r1_prob */
66      BEGIN
67           front_train1,front_train2,end_train1,end_train2,fst_t1block,lst_t1
 •           block,fst_t2block,lst_t2block,block_state,next_turn :
 •           (front_train1 ∈ ℕ ∧ (front_train2 ∈ ℕ ∧ (end_train1 ∈ ℕ ∧
 •           (end_train2 ∈ ℕ ∧ (end_train1 < front_train1 ∧ (end_train2 <
 •           front_train2 ∧ (front_train1 < end_train2 ∧ (block_state ∈ Block
 •           → BlockState ∧ (fst_t1block ∈ Block ∧ (lst_t1block ∈ Block ∧
 •           (fst_t2block ∈ Block ∧ (lst_t2block ∈ Block ∧ (lst_t1block ≤
 •           fst_t1block ∧ (lst_t2block ≤ fst_t2block ∧ (fst_t1block <
 •           lst_t2block ∧ (end_block(lst_t2block) ≤ end_train2 ∧ (next_turn =
 •           TRAIN ⇒ ∀bk·(bk ∈ Block ∧ block_state(bk) = Free ⇒ bk ≠
 •           lst_t2block))))))))))))))))))
68      REFINES
69        EVENT /* of machine Modele00_r1 */
70        BEGIN
71          skip
72        REFINES
73          EVENT /* of machine Modele00 */
74          BEGIN
75               front_train1,end_train1,front_train2,end_train2 :
 •               (front_train1 ∈ ℕ ∧ (end_train1 ∈ ℕ ∧ (front_train2 ∈ ℕ ∧
 •               (end_train2 ∈ ℕ ∧ (end_train1 < front_train1 ∧ (end_train2 <
 •               front_train2 ∧ front_train1 < end_train2))))))
76          END
77        END
78      END
79  EVENTS
80    move_front_train1_TRNevt(step) =
81      EVENT move_front_train1_TRNevt = /* of machine Modele00_r1_prob */
82      ANY step
83      WHERE
84
```

```
84
85          /* @Modele00_r1_prob:grd1 */ step ∈ ℕ₁
86        & /* @Modele00_r1_prob:grd2 */ front_train1 + step <
  •       front_block(fst_t1block)
87        & /* @Modele00_r1_prob:grd3 */ next_turn = TRAIN
88      THEN
89           front_train1 := front_train1 + step
90        ||
91           next_turn := TRACK
92      REFINES
93        EVENT move_front_train1_TRNevt = /* of machine Modele00_r1 */
94        BEGIN
95          skip
96        REFINES
97          EVENT move_front_train1 = /* of machine Modele00 */
98          WHEN
99              /* @Modele00:grd2 */ front_train1 + step < end_train2
100         THEN
101           skip
102         END
103       END
104     END;
105
106   move_end_train1_TRNevet(step) =
107     EVENT move_end_train1_TRNevet = /* of machine Modele00_r1_prob */
108     ANY step
109     WHERE
110         /* @Modele00_r1_prob:grd1 */ step ∈ ℕ₁
111       & /* @Modele00_r1_prob:grd2 */ end_train1 + step < front_train1
112       & /* @Modele00_r1_prob:grd3 */ end_train1 + step <
  •       front_block(lst_t1block)
113       & /* @Modele00_r1_prob:grd4 */ next_turn = TRAIN
114     THEN
115          end_train1 := end_train1 + step
116       ||
117          next_turn := TRACK
118     REFINES
119       EVENT move_end_train1_TRNevet = /* of machine Modele00_r1 */
120       BEGIN
121         skip
122       REFINES
123         EVENT move_end_train1 = /* of machine Modele00 */
124         BEGIN
125           skip
126         END
127       END
128     END;
129
130   free_t1block_TRNevt(step) =
131     EVENT free_t1block_TRNevt = /* of machine Modele00_r1_prob */
132     ANY step
```

```
133     WHERE
134         /* @Modele00_r1_prob:grd1 */ step ∈ ℕ₁
135       & /* @Modele00_r1_prob:grd2 */ end_train1 + step < front_train1
136       & /* @Modele00_r1_prob:grd3 */ front_block(lst_t1block) < end_train1
  •       + step
137       & /* @Modele00_r1_prob:grd4 */ end_train1 + step <
  •       front_block(next_block(lst_t1block))
138       & /* @Modele00_r1_prob:grd5 */ next_block(lst_t1block) ≤ fst_t1block
139       & /* @Modele00_r1_prob:grd6 */ next_turn = TRAIN
140     THEN
141         next_turn := TRACK
142       ||
143         end_train1 := end_train1 + step
144       ||
145         lst_t1block := next_block(lst_t1block)
146     REFINES
147       EVENT free_t1block_TRNevt = /* of machine Modele00_r1 */
148       BEGIN
149         skip
150       REFINES
151         EVENT move_end_train1 = /* of machine Modele00 */
152         BEGIN
153           skip
154         END
155       END
156     END;
157
158   Enter_t1block_TRNevt(step) =
159     EVENT Enter_t1block_TRNevt = /* of machine Modele00_r1_prob */
160     ANY step
161     WHERE
162         /* @Modele00_r1_prob:grd1 */ step ∈ ℕ₁
163       & /* @Modele00_r1_prob:grd2 */ block_state(next_block(fst_t1block))
  •       = Free
164       & /* @Modele00_r1_prob:grd3 */ front_block(fst_t1block) <
  •       front_train1 + step
165       & /* @Modele00_r1_prob:grd4 */ front_train1 + step <
  •       front_block(next_block(fst_t1block))
166       & /* @Modele00_r1_prob:grd5 */ next_turn = TRAIN
167     THEN
168         next_turn := TRACK
169       ||
170         front_train1 := front_train1 + step
171       ||
172         fst_t1block := next_block(fst_t1block)
173     REFINES
174       EVENT Enter_t1block_TRNevt = /* of machine Modele00_r1 */
175       BEGIN
176         skip
177       REFINES
```

```
178         EVENT move_front_train1 = /* of machine Modele00 */
179         WHEN
180             /* @Modele00:grd2 */ front_train1 + step < end_train2
181         THEN
182           skip
183         END
184       END
185     END;
186
187   move_front_train2_TRNevt(step) =
188     EVENT move_front_train2_TRNevt = /* of machine Modele00_r1_prob */
189     ANY step
190     WHERE
191         /* @Modele00_r1_prob:grd1 */ step ∈ ℕ₁
192       & /* @Modele00_r1_prob:grd2 */ front_train2 + step <
  •       front_block(fst_t2block)
193       & /* @Modele00_r1_prob:grd3 */ next_turn = TRAIN
194     THEN
195          front_train2 := front_train2 + step
196       ||
197          next_turn := TRACK
198     REFINES
199       EVENT move_front_train2_TRNevt = /* of machine Modele00_r1 */
200       BEGIN
201         skip
202       REFINES
203         EVENT move_front_train2 = /* of machine Modele00 */
204         BEGIN
205           skip
206         END
207       END
208     END;
209   move_end_train2_TRNevt(step) =
210     EVENT move_end_train2_TRNevt = /* of machine Modele00_r1_prob */
211     ANY step
212     WHERE
213         /* @Modele00_r1_prob:grd1 */ step ∈ ℕ₁
214       & /* @Modele00_r1_prob:grd2 */ end_train2 + step < front_train2
215       & /* @Modele00_r1_prob:grd3 */ end_train2 + step <
  •       front_block(lst_t2block)
216       & /* @Modele00_r1_prob:grd4 */ next_turn = TRAIN
217     THEN
218          end_train2 := end_train2 + step
219       ||
220          next_turn := TRACK
221     REFINES
222       EVENT move_end_train2_TRNevt = /* of machine Modele00_r1 */
223       BEGIN
224         skip
225       REFINES
```

```
226
227         EVENT move_end_train2 = /* of machine Modele00 */
228         BEGIN
229           skip
230         END
231       END
232     END;
233
234   free_t2block_TRNevt(step) =
235     EVENT free_t2block_TRNevt = /* of machine Modele00_r1_prob */
236     ANY step
237     WHERE
238         /* @Modele00_r1_prob:grd1 */ step ∈ ℕ₁
239       & /* @Modele00_r1_prob:grd2 */ end_train2 + step < front_train2
240       & /* @Modele00_r1_prob:grd3 */ front_block(lst_t2block) < end_train2
  •       + step
241       & /* @Modele00_r1_prob:grd4 */ end_train2 + step <
  •       front_block(next_block(lst_t2block))
242       & /* @Modele00_r1_prob:grd5 */ next_block(lst_t2block) ≤ fst_t2block
243       & /* @Modele00_r1_prob:grd6 */ next_turn = TRAIN
244     THEN
245         next_turn := TRACK
246       ||
247         end_train2 := end_train2 + step
248       ||
249         lst_t2block := next_block(lst_t2block)
250     REFINES
251       EVENT free_t2block_TRNevt = /* of machine Modele00_r1 */
252       BEGIN
253         skip
254       REFINES
255         EVENT move_end_train2 = /* of machine Modele00 */
256         BEGIN
257           skip
258         END
259       END
260     END;
261
262   Enter_t2block_TRNevt(step) =
263     EVENT Enter_t2block_TRNevt = /* of machine Modele00_r1_prob */
264     ANY step
265     WHERE
266         /* @Modele00_r1_prob:grd1 */ step ∈ ℕ₁
267       & /* @Modele00_r1_prob:grd2 */ block_state(next_block(fst_t2block))
  •       = Free
268       & /* @Modele00_r1_prob:grd3 */ front_block(fst_t2block) <
  •       front_train2 + step
269       & /* @Modele00_r1_prob:grd4 */ front_train2 + step <
  •       front_block(next_block(fst_t2block))
270       & /* @Modele00_r1_prob:grd5 */ next_turn = TRAIN
271     THEN
```

```
272           next_turn := TRACK
273         ||
274           front_train2 := front_train2 + step
275         ||
276           fst_t2block := next_block(fst_t2block)
277       REFINES
278         EVENT Enter_t2block_TRNevt = /* of machine Modele00_r1 */
279         BEGIN
280           skip
281         REFINES
282           EVENT move_front_train2 = /* of machine Modele00 */
283           BEGIN
284             skip
285           END
286         END
287       END;
288
289     TRACKevent =
290       EVENT TRACKevent = /* of machine Modele00_r1_prob */
291       WHEN
292           /* @Modele00_r1_prob:grd1 */ next_turn = TRACK
293       THEN
294           next_turn := TRAIN
295         ||
296           block_state := Block × {Free} <+ (lst_t1block .. fst_t1block ∪
  •           lst_t2block .. fst_t2block) × {Occupied}
297       REFINES
298         EVENT TRACKevent = /* of machine Modele00_r1 */
299         BEGIN
300           skip
301         END
302       END
303   END
304
```

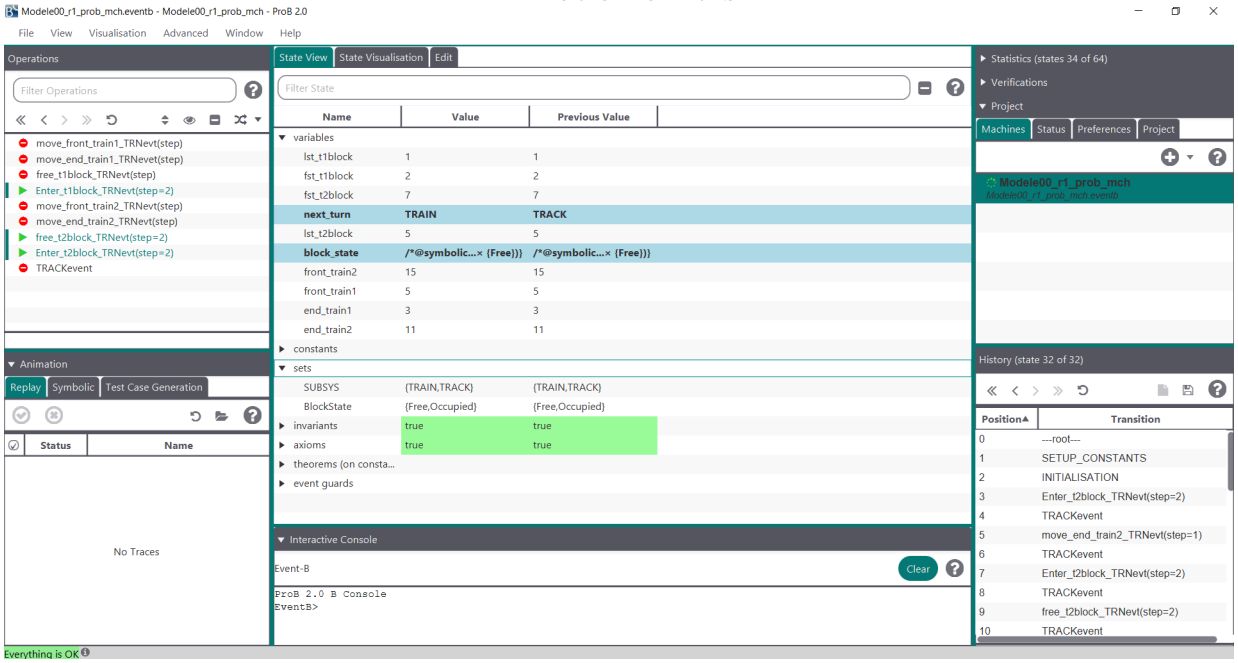
JSON File for the Animation

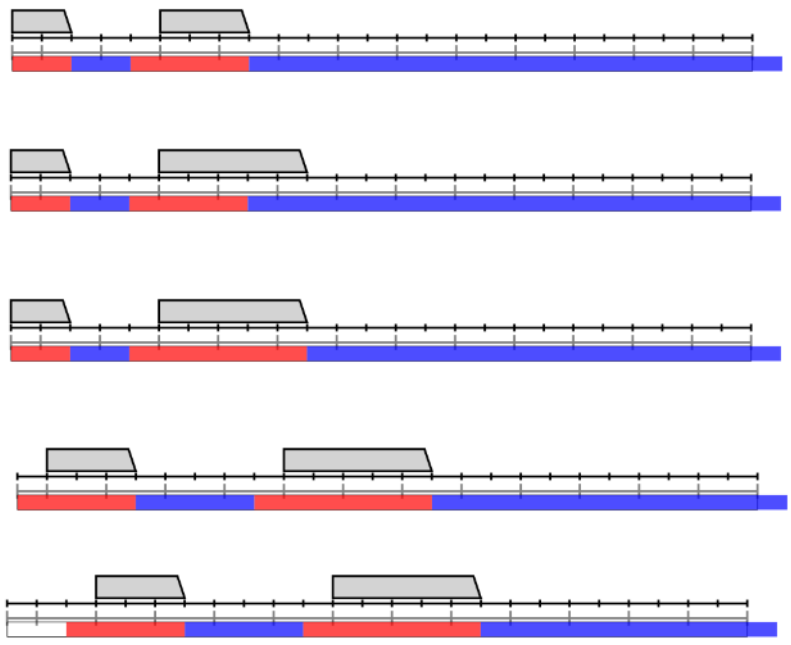
```
{
  "svg":"Track.svg",
  "items":[

    {
      "repeat": ["1","2"],
      "id":"train_polygon%0",
      "attr":"points",
      "value":"svg_train(real(end_train%0), real(1+front_train%0-end_train%0) , 4.0,
      1.0, 3.0)",
      "comment":"show train position using a slanted polygon"
    },

    {
      "id":"track_polyline",
      "attr":"points",
      "value":"svg_axis(0..25 ,4.0,100.0,1.0)",
      "comment":"show ticks for Track units"
    },
    {
      "id":"ttd_polyline",
      "attr":"points",
      "value":"svg_axis({0} \\/ ran(%tt.(tt:0..12|1+end_block(tt))),4.0,100.0,2.0)",
      "comment":"show ticks for TTD Limits"
    },
    {
      "id":"occupied_ttd_polygon",
      "attr":"points",
      "value":"svg_set_polygon((end_block\\/(end_block;succ))[(0..12 <|
      block_state)~[{Occupied}]],4.0,100.0,2.0)",
      "comment":"show occupied TTD zones"
    },
    {
      "id":"cleared_ttd_polygon",
      "attr":"points",
      "value":"svg_set_polygon((end_block\\/(end_block;succ))[(1..12 <|
      block_state)~[{Free}]],4.0,100.0,2.0)",
      "comment":"show free TTD zones"
    }
    ],
  "events":[
    ]
}

```

SVG File for the Animation

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg
   xmlns="http://www.w3.org/2000/svg"
   width="750"
   height="300"
   viewBox="0 5 150 65"
   version="1.1"
   id="svg5154">

  <polygon id = "train_polygon1"
       points="0,0 100,0"
       style="stroke-width: 0.3"
       stroke="black" fill="lightgray"
       transform="translate(10,16.8)" />
  <polygon id = "train_polygon2"
       points="0,0 100,0"
       style="stroke-width: 0.3"
       stroke="black" fill="lightgray"
       transform="translate(10,16.8)" />

  <polygon id = "track_polyline"
       points="0,0 1,0, 1,1 1,0 50,0 50,1 50,0 100,0"
       style="stroke-width: 0.3"
       stroke="black" fill="none"
       transform="translate(10,20.5)" />
  <polygon id = "ttd_polyline"
       points="0,0 100,0"
       style="stroke-width: 0.3"
       stroke="gray" fill="none"
       transform="translate(10,22.5)" />


  <rect id = "ttd_rect"
       style="stroke-width: 0.1"
       width="100" height="2" x="0"  y="0"
       stroke="black" fill="none"
       transform="translate(10,23)" />
  <polygon id = "occupied_ttd_polygon"
       points="0,0 0,2 10,2 10,0 70,0 70,1 90,1 90,0"
       stroke="none" fill="red"
       opacity="0.70"
       style="stroke-width: 0.2"
       transform="translate(10,23)" />
  <polygon id = "cleared_ttd_polygon"
       points="0,0 10,2 20,2 20,0 90,0 90,1 100,1 100,0"
       stroke="none" fill="blue"
       opacity="0.70"
       style="stroke-width: 0.2"
       transform="translate(10,23)" />

    <text  text-align="left"  x="5"  y="41"
       font-size ="2" fill="gray"  font-family="sans-serif">
       <tspan x="15" dy = "0.6em" id="visb_debug_messages">.TXT</tspan>
    </text>
</svg>

```

Animation on VisB





A.3 Second Refinement: Definition of Signals

A.3.1 Context of Signals

CONTEXT Signal
EXTENDS Block
SETS
SignalState // a signal can be red or green
CONSTANTS
Signal
signal_block // each signal is associated to one block
red
green
AXIOMS
axm1: $SignalState = \{red, green\}$
axm2: $Signal = \mathbb{N}$

// each signal is associated to one block: signal $i \Leftrightarrow$ block i
axm3: $signal\_block = (\lambda bk \cdot bk \in Block | bk)$
END

A.3.2 Refinement Machine: M2

MACHINE M2
REFINES M1
SEES Signal
VARIABLES

signal_state // variable of the signal state
front_trainA
front_trainB
end_trainA
end_trainB
fst_tAblock
lst_tAblock
fst_tBblock
lst_tBblock
block_state
next_turn

INVARIANTS

inv1: $signal\_state \in Signal \rightarrow SignalState$ // each signal can be red or green
inv2:
$\forall bk \cdot ($
$bk \in Block$
$\wedge\ next\_turn = TRAIN$
$\wedge\ signal\_state(signal\_block(bk)) = green$
$\Rightarrow block\_state(bk) = Free)$

EVENTS
Initialisation
begin
act1:
$signal\_state$,
$front\_trainA$,
$front\_trainB$,
end_trainA,
end_trainB,
$fst\_tAblock$,
$lst\_tAblock$,
$fst\_tBblock$,
$lst\_tBblock$,
$block\_state$,
$next\_turn :| ($
$front\_trainA' \in \mathbb{N}$
$\wedge\ front\_trainB' \in \mathbb{N}$
$\wedge\ end\_trainA' \in \mathbb{N}$

$\land\ end\_trainB' \in \mathbb{N}$
$\land\ end\_trainA' < front\_trainA'$
$\land\ end\_trainB' < front\_trainB'$
$\land\ front\_trainA' < end\_trainB'$
$\land\ block\_state' \in Block \to BlockState$
$\land\ fst\_tAblock' \in Block$
$\land\ lst\_tAblock' \in Block$
$\land\ fst\_tBblock' \in Block$
$\land\ lst\_tBblock' \in Block$
$\land\ next\_turn' \in SUBSYS$
$\land\ lst\_tAblock' <= fst\_tAblock'$
$\land\ lst\_tBblock' <= fst\_tBblock'$
$\land\ fst\_tAblock' < lst\_tBblock'$
$\land\ end\_block(lst\_tBblock') <= end\_trainB'$
$\land\ signal\_state' \in Signal \to SignalState$
$\land\ \forall bk \cdot ($
$bk \in Block$
$\land\ next\_turn' = TRAIN$
$\land\ signal\_state'(signal\_block(bk)) = green$
$\Rightarrow block\_state'(bk) = Free))$

end

//
////////////////////// Train Behavior //////////////////////////////////
//

Event move_front_trainA ⟨ordinary⟩ $\widehat{=}$
refines move_front_trainA
any
step
where
grd1: $step \in \mathbb{N}_1$
grd2: $front\_trainA + step < front\_block(fst\_tAblock)$
grd3: $next\_turn = TRAIN$
then
act1: $front\_trainA := front\_trainA + step$
act2: $next\_turn := TRACK$
end

Event move_end_trainA ⟨ordinary⟩ $\widehat{=}$
refines move_end_trainA
any
step
where
grd1: $step \in \mathbb{N}_1$
grd2: $end\_trainA + step < front\_trainA$
grd3: $end\_trainA + step < front\_block(lst\_tAblock)$

grd4: $next\_turn = TRAIN$

then

act1: $end\_trainA := end\_trainA + step$

act2: $next\_turn := TRACK$

end

Event free_tAblock ⟨ordinary⟩ $\widehat{=}$

refines free_tAblock

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainA + step < front\_trainA$

grd3: $front\_block(lst\_tAblock) < end\_trainA + step$

grd4: $end\_trainA + step < front\_block(next\_block(lst\_tAblock))$

grd5: $next\_block(lst\_tAblock) <= fst\_tAblock$

grd6: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$

act2: $end\_trainA := end\_trainA + step$

act3: $lst\_tAblock := next\_block(lst\_tAblock)$

end

Event Enter_tAblock ⟨ordinary⟩ $\widehat{=}$

refines Enter_tAblock

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $signal\_state(next\_block(fst\_tAblock)) = green$

grd3: $front\_block(fst\_tAblock) < front\_trainA + step$

grd4: $front\_trainA + step < front\_block(next\_block(fst\_tAblock))$

grd5: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$

act2: $front\_trainA := front\_trainA + step$

act3: $fst\_tAblock := next\_block(fst\_tAblock)$

end

Event move_front_trainB ⟨ordinary⟩ $\widehat{=}$

refines move_front_trainB

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $front\_trainB + step < front\_block(fst\_tBblock)$

grd3: $next\_turn = TRAIN$

then

act1: $front\_trainB := front\_trainB + step$

act2: $next\_turn := TRACK$

end

Event move_end_trainB ⟨ordinary⟩ $\widehat{=}$

refines move_end_trainB

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainB + step < front\_trainB$

grd3: $end\_trainB + step < front\_block(lst\_tBblock)$

grd4: $next\_turn = TRAIN$

then

act1: $end\_trainB := end\_trainB + step$

act2: $next\_turn := TRACK$

end

Event free_tBblock ⟨ordinary⟩ $\widehat{=}$

refines free_tBblock

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainB + step < front\_trainB$

grd3: $front\_block(lst\_tBblock) < end\_trainB + step$

grd4: $end\_trainB + step < front\_block(next\_block(lst\_tBblock))$

grd5: $next\_block(lst\_tBblock) <= fst\_tBblock$

grd6: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$

act2: $end\_trainB := end\_trainB + step$

act3: $lst\_tBblock := next\_block(lst\_tBblock)$

end

Event Enter_tBblock ⟨ordinary⟩ $\widehat{=}$

refines Enter_tBblock

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $signal\_state(next\_block(fst\_tBblock)) = green$

grd3: $front\_block(fst\_tBblock) < front\_trainB + step$

grd4: $front\_trainB + step < front\_block(next\_block(fst\_tBblock))$

grd5: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$
act2: $front\_trainB := front\_trainB + step$
act3: $fst\_tBblock := next\_block(fst\_tBblock)$

end

//
////////////////////// Track and Signals Behavior //////////////////////
//

// block and signals states changes

Event TRACKevent ⟨ordinary⟩ $\widehat{=}$

refines TRACKevent

when

grd1: $next\_turn = TRACK$

then

act1: $next\_turn := TRAIN$
act2: $block\_state := (Block \times \{Free\}) \lhd ((lst\_tAblock \mathinner{..} fst\_tAblock \cup lst\_tBblock \mathinner{..} fst\_tBblock) \times \{Occupied\})$
// signals states changes
act3: $signal\_state := (Signal \times \{green\}) \lhd ((signal\_block(lst\_tAblock) \mathinner{..} signal\_block(fst\_tAblock) \cup signal\_block(lst\_tBblock) \mathinner{..} signal\_block(fst\_tBblock)) \times \{red\})$

end

END

APPENDIX B

DECOMPOSITION OF THE CASE STUDY USING A-STYLE

B.1 Additional Refinement Step of the Case Study

MACHINE M3
REFINES M2
SEES Signal
VARIABLES

signal_state
front_trainA
front_trainB
end_trainA
end_trainB
fst_tAblock
lst_tAblock
fst_tBblock
lst_tBblock
block_state
// definition of two new variables
for the refinement of the shared variable *next_turn*
ww
tt

INVARIANTS

inv1: $ww \in \{0,1\}$ // variables typing
inv2: $tt \in \{0,1\}$ // variables typing

// refinement of *next_turn*

inv3: $ww = tt \Rightarrow next\_turn = TRAIN$

inv4: $ww \neq tt \Rightarrow next\_turn = TRACK$

EVENTS

Initialisation

begin

act1: $ww := 0$

act2: $tt := 1$

act3:

$signal\_state$,
$front\_trainA$,
$front\_trainB$,
end_trainA,
end_trainB,
$fst\_tAblock$,
$lst\_tAblock$,
$fst\_tBblock$,
$lst\_tBblock$,
$block\_state :|$ (
$front\_trainA' \in \mathbb{N}$
$\wedge\ front\_trainB' \in \mathbb{N}$
$\wedge\ end\_trainA' \in \mathbb{N}$
$\wedge\ end\_trainB' \in \mathbb{N}$
$\wedge\ end\_trainA' < front\_trainA'$
$\wedge\ end\_trainB' < front\_trainB'$
$\wedge\ front\_trainA' < end\_trainB'$
$\wedge\ block\_state' \in Block \rightarrow BlockState$
$\wedge\ fst\_tAblock' \in Block$
$\wedge\ lst\_tAblock' \in Block$
$\wedge\ fst\_tBblock' \in Block$
$\wedge\ lst\_tBblock' \in Block$
$\wedge\ lst\_tAblock' <= fst\_tAblock'$
$\wedge\ lst\_tBblock' <= fst\_tBblock'$
$\wedge\ fst\_tAblock' < lst\_tBblock'$
$\wedge\ end\_block(lst\_tBblock') <= end\_trainB'$
$\wedge\ signal\_state' \in Signal \rightarrow SignalState$
)

end

Event move_front_trainA ⟨ordinary⟩ $\widehat{=}$

refines move_front_trainA

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $front\_trainA + step < front\_block(fst\_tAblock)$

grd3: $ww = tt$

then

act1: $front\_trainA := front\_trainA + step$

act2: $tt := 1 - tt$

end

Event move_end_trainA ⟨ordinary⟩ $\widehat{=}$

refines move_end_trainA

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainA + step < front\_trainA$

grd3: $end\_trainA + step < front\_block(lst\_tAblock)$

grd4: $ww = tt$

then

act1: $end\_trainA := end\_trainA + step$

act2: $tt := 1 - tt$

end

Event free_tAblock ⟨ordinary⟩ $\widehat{=}$

refines free_tAblock

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainA + step < front\_trainA$

grd3: $front\_block(lst\_tAblock) < end\_trainA + step$

grd4: $end\_trainA + step < front\_block(next\_block(lst\_tAblock))$

grd5: $next\_block(lst\_tAblock) <= fst\_tAblock$

grd6: $ww = tt$

then

act1: $tt := 1 - tt$

act2: $end\_trainA := end\_trainA + step$

act3: $lst\_tAblock := next\_block(lst\_tAblock)$

end

Event Enter_tAblock ⟨ordinary⟩ $\widehat{=}$

refines Enter_tAblock

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $block\_state(next\_block(fst\_tAblock)) = Free$

grd3: $front\_block(fst\_tAblock) < front\_trainA + step$

grd4: $front\_trainA + step < front\_block(next\_block(fst\_tAblock))$

grd5: $ww = tt$

then

act1: $tt := 1 - tt$

act2: $front\_trainA := front\_trainA + step$

act3: $fst\_tAblock := next\_block(fst\_tAblock)$

end

Event move_front_trainB ⟨ordinary⟩ $\widehat{=}$

refines move_front_trainB

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $front\_trainB + step < front\_block(fst\_tBblock)$

grd3: $ww = tt$

then

act1: $front\_trainB := front\_trainB + step$

act2: $tt := 1 - tt$

end

Event move_end_trainB ⟨ordinary⟩ $\widehat{=}$

refines move_end_trainB

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainB + step < front\_trainB$

grd3: $end\_trainB + step < front\_block(lst\_tBblock)$

grd4: $ww = tt$

then

act1: $end\_trainB := end\_trainB + step$

act2: $tt := 1 - tt$

end

Event free_tBblock ⟨ordinary⟩ $\widehat{=}$

refines free_tBblock

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainB + step < front\_trainB$

grd3: $front\_block(lst\_tBblock) < end\_trainB + step$

grd4: $end\_trainB + step < front\_block(next\_block(lst\_tBblock))$

grd5: $next\_block(lst\_tBblock) <= fst\_tBblock$

grd6: $ww = tt$

then

act1: $tt := 1 - tt$

act2: $end\_trainB := end\_trainB + step$

act3: $lst\_tBblock := next\_block(lst\_tBblock)$

end

Event Enter_tBblock ⟨ordinary⟩ $\widehat{=}$

refines Enter_tBblock

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $block\_state(next\_block(fst\_tBblock)) = Free$

grd3: $front\_block(fst\_tBblock) < front\_trainB + step$

grd4: $front\_trainB + step < front\_block(next\_block(fst\_tBblock))$

grd5: $ww = tt$

then

act1: $tt := 1 - tt$

act2: $front\_trainB := front\_trainB + step$

act3: $fst\_tBblock := next\_block(fst\_tBblock)$

end

// block and signals states changes

Event TRACKevent ⟨ordinary⟩ $\widehat{=}$

refines TRACKevent

when

grd1: $ww \neq tt$

then

act1: $ww := 1 - ww$

act2: $block\_state := (Block \times \{Free\}) \lhd ((lst\_tAblock \mathinner{..} fst\_tAblock \cup lst\_tBblock \mathinner{..} fst\_tBblock) \times \{Occupied\})$

// signals states changes

act3: $signal\_state := (Signal \times \{green\}) \lhd ((signal\_block(lst\_tAblock) \mathinner{..} signal\_block(fst\_tAblock) \cup signal\_block(lst\_tBblock) \mathinner{..} signal\_block(fst\_tB2block)) \times \{red\})$

end

END

B.2 Application of the A-style Plugin

B.2.1 First Sub-system: Train

MACHINE M3_Astyle_train

SEES Signal

VARIABLES

ww Shared variable, DO NOT REFINE

tt Shared variable, DO NOT REFINE

end_trainA Private variable

front_trainA Private variable

front_trainB Private variable

end_trainB Private variable

lst_tAblock Shared variable, DO NOT REFINE

fst_tAblock Shared variable, DO NOT REFINE

lst_tBblock Shared variable, DO NOT REFINE

fst_tBblock Shared variable, DO NOT REFINE

signal_state Shared variable, DO NOT REFINE

INVARIANTS

typing_ww: ⟨theorem⟩ $ww \in \mathbb{Z}$

typing_tt: ⟨theorem⟩ $tt \in \mathbb{Z}$

typing_end_trainA: ⟨theorem⟩ $end\_trainA \in \mathbb{Z}$

typing_front_trainA: ⟨theorem⟩ $front\_trainA \in \mathbb{Z}$

typing_end_trainB: ⟨theorem⟩ $end\_trainB \in \mathbb{Z}$

typing_lst_tAblock: ⟨theorem⟩ $lst\_tAblock \in \mathbb{Z}$

typing_fst_tAblock: ⟨theorem⟩ $fst\_tAblock \in \mathbb{Z}$

typing_signal_state: ⟨theorem⟩ $signal\_state \in \mathbb{P}(\mathbb{Z} \times SignalState)$

typing_lst_tBblock: ⟨theorem⟩ $lst\_tBblock \in \mathbb{Z}$

typing_fst_tBblock: ⟨theorem⟩ $fst\_tBblock \in \mathbb{Z}$

typing_front_trainB: ⟨theorem⟩ $front\_trainB \in \mathbb{Z}$

Modele00_inv1: $front\_trainA \in \mathbb{N}$

Modele00_inv2: $front\_trainB \in \mathbb{N}$

Modele00_inv3: $end\_trainA \in \mathbb{N}$

Modele00_inv4: $end\_trainB \in \mathbb{N}$

Modele00_inv5: $end\_trainA < front\_trainA$

Modele00_inv6: $end\_trainB < front\_trainB$

Modele00_inv7: $front\_trainA < end\_trainB$

Modele00_r1_inv1: $fst\_tAblock \in Block$

Modele00_r1_inv2: $lst\_tAblock \in Block$

Modele00_r1_inv3: $fst\_tBblock \in Block$

Modele00_r1_inv4: $lst\_tBblock \in Block$

Modele00_r1_inv7: $lst\_tAblock \leq fst\_tAblock$

Modele00_r1_inv8: $lst\_tBblock \leq fst\_tBblock$

Modele00_r1_inv9: $fst\_tAblock < lst\_tBblock$

Modele00_r1_inv10: $end\_block(lst\_tBblock) \leq end\_trainB$

Modele00_r22_inv1: $signal\_state \in Signal \rightarrow SignalState$

M3_inv1: $ww \in \{0, 1\}$

M3_inv2: $tt \in \{0, 1\}$

EVENTS

Initialisation

begin

act1: $ww := 0$

act2: $tt := 1$

act3:

$signal\_state,$
$front\_trainA,$
$front\_trainB,$
$end\_trainA,$
$end\_trainB,$
$fst\_tAblock,$
$lst\_tAblock,$
$fst\_tBblock,$
$lst\_tBblock :|$
$(\exists block\_state' \cdot front\_trainA' \in \mathbb{N}$
$\wedge\ front\_trainB' \in \mathbb{N}$
$\wedge\ end\_trainA' \in \mathbb{N}$
$\wedge\ end\_trainB' \in \mathbb{N}$
$\wedge\ end\_trainA' < front\_trainA'$
$\wedge\ end\_trainB' < front\_trainB'$
$\wedge\ front\_trainA' < end\_trainB'$
$\wedge\ block\_state' \in Block \rightarrow BlockState$
$\wedge\ fst\_tAblock' \in Block$
$\wedge\ lst\_tAblock' \in Block$
$\wedge\ fst\_tBblock' \in Block$
$\wedge\ lst\_tBblock' \in Block$
$\wedge\ lst\_tAblock' \leq fst\_tAblock'$
$\wedge\ lst\_tBblock' \leq fst\_tBblock'$
$\wedge\ fst\_tAblock' < lst\_tBblock'$
$\wedge\ end\_block(lst\_tBblock') \leq end\_trainB'$
$\wedge\ signal\_state' \in Signal \rightarrow SignalState)$

end

Event move_front_trainA ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $front\_trainA + step < front\_block(fst\_tAblock)$
grd3: $ww = tt$

then

act1: $front\_trainA := front\_trainA + step$
act2: $tt := 1 - tt$

end

Event move_end_trainA ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $end\_trainA + step < front\_trainA$
grd3: $end\_trainA + step < front\_block(lst\_tAblock)$
grd4: $ww = tt$

then

act1: $end\_trainA := end\_trainA + step$
act2: $tt := 1 - tt$

end

Event free_tAblock ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $end\_trainA + step < front\_trainA$
grd3: $front\_block(lst\_tAblock) < end\_trainA + step$
grd4: $end\_trainA + step < front\_block(next\_block(lst\_tAblock))$
grd5: $next\_block(lst\_tAblock) \leq fst\_tAblock$
grd6: $ww = tt$

then

act1: $tt := 1 - tt$
act2: $end\_trainA := end\_trainA + step$
act3: $lst\_tAblock := next\_block(lst\_tAblock)$

end

Event Enter_tAblock ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $signal\_state(next\_block(fst\_tAblock)) = green$
grd3: $front\_block(fst\_tAblock) < front\_trainA + step$
grd4: $front\_trainA + step < front\_block(next\_block(fst\_tAblock))$
grd5: $ww = tt$

then

act1: $tt := 1 - tt$

act2: $front\_trainA := front\_trainA + step$
act3: $fst\_tAblock := next\_block(fst\_tAblock)$

end

Event move_front_trainB ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $front\_trainB + step < front\_block(fst\_tBblock)$
grd3: $ww = tt$

then

act1: $front\_trainB := front\_trainB + step$
act2: $tt := 1 - tt$

end

Event move_end_trainB ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $end\_trainB + step < front\_trainB$
grd3: $end\_trainB + step < front\_block(lst\_tBblock)$
grd4: $ww = tt$

then

act1: $end\_trainB := end\_trainB + step$
act2: $tt := 1 - tt$

end

Event free_tBblock ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $end\_trainB + step < front\_trainB$
grd3: $front\_block(lst\_tBblock) < end\_trainB + step$
grd4: $end\_trainB + step < front\_block(next\_block(lst\_tBblock))$
grd5: $next\_block(lst\_tBblock) \leq fst\_tBblock$
grd6: $ww = tt$

then

act1: $tt := 1 - tt$
act2: $end\_trainB := end\_trainB + step$
act3: $lst\_tBblock := next\_block(lst\_tBblock)$

end

Event Enter_tBblock ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $signal\_state(next\_block(fst\_tBblock)) = green$
grd3: $front\_block(fst\_tBblock) < front\_trainB + step$
grd4: $front\_trainB + step < front\_block(next\_block(fst\_tBblock))$
grd5: $ww = tt$

then

act1: $tt := 1 - tt$
act2: $front\_trainB := front\_trainB + step$
act3: $fst\_tBblock := next\_block(fst\_tBblock)$

end

Event TRACKevent ⟨ordinary⟩ $\widehat{=}$

External event, DO NOT REFINE

when

grd1: $ww \neq tt$

then

act1: $ww := 1 - ww$
act3: $signal\_state := (Signal \times \{green\}) \lhd ((signal\_block(lst\_tAblock)..signal\_block(fst\_tAblock) \cup signal\_block(lst\_tBblock)\,..\,signal\_block(fst\_tBblock)) \times \{red\})$

end

END

B.2.2 Second Sub-system: Track

MACHINE M3_Astyle_track

SEES Signal

VARIABLES

ww Shared variable, DO NOT REFINE

tt Shared variable, DO NOT REFINE

lst_tAblock Shared variable, DO NOT REFINE

fst_tAblock Shared variable, DO NOT REFINE

signal_state Shared variable, DO NOT REFINE

block_state Private variable

lst_tBblock Shared variable, DO NOT REFINE

fst_tBblock Shared variable, DO NOT REFINE

INVARIANTS

typing_ww: ⟨theorem⟩ $ww \in \mathbb{Z}$

typing_tt: ⟨theorem⟩ $tt \in \mathbb{Z}$

typing_lst_tAblock: ⟨theorem⟩ $lst\_tAblock \in \mathbb{Z}$

typing_fst_tAblock: ⟨theorem⟩ $fst\_tAblock \in \mathbb{Z}$

typing_signal_state: ⟨theorem⟩ $signal\_state \in \mathbb{P}(\mathbb{Z} \times SignalState)$

typing_block_state: ⟨theorem⟩ $block\_state \in \mathbb{P}(\mathbb{Z} \times BlockState)$

typing_lst_tBblock: ⟨theorem⟩ $lst\_tBblock \in \mathbb{Z}$

typing_fst_tBblock: ⟨theorem⟩ $fst\_tBblock \in \mathbb{Z}$

Modele00_r1_inv1: $fst\_tAblock \in Block$

Modele00_r1_inv2: $lst\_tAblock \in Block$

Modele00_r1_inv3: $fst\_tBblock \in Block$

Modele00_r1_inv4: $lst\_tBblock \in Block$

Modele00_r1_inv5: $block\_state \in Block \rightarrow BlockState$

Modele00_r1_inv7: $lst\_tAblock \leq fst\_tAblock$

Modele00_r1_inv8: $lst\_tBblock \leq fst\_tBblock$

Modele00_r1_inv9: $fst\_tAblock < lst\_tBblock$

WD_Modele00_r1_inv10: ⟨theorem⟩ $lst\_tBblock \in dom(end\_block)$

WD_Modele00_r1_inv10_1: ⟨theorem⟩ $end\_block \in \mathbb{Z} \nrightarrow \mathbb{Z}$

Modele00_r22_inv1: $signal\_state \in Signal \rightarrow SignalState$

M3_inv1: $ww \in \{0, 1\}$

M3_inv2: $tt \in \{0, 1\}$

EVENTS

Initialisation

begin

act1: $ww := 0$

act2: $tt := 1$

act3:
$signal\_state$,
$fst\_tAblock$,
$lst\_tAblock$,
$fst\_tBblock$,
$lst\_tBblock$,
$block\_state :| ($
$\exists front\_trainA', front\_trainB', end\_trainA', end\_trainB' \cdot front\_trainA' \in \mathbb{N}$
$\land front\_trainB' \in \mathbb{N}$
$\land end\_trainA' \in \mathbb{N}$
$\land end\_trainB' \in \mathbb{N}$
$\land end\_trainA' < front\_trainA'$
$\land end\_trainB' < front\_trainB'$
$\land front\_trainA' < end\_trainB'$
$\land block\_state' \in Block \to BlockState$
$\land fst\_tAblock' \in Block$
$\land lst\_tAblock' \in Block$
$\land fst\_tBblock' \in Block$
$\land lst\_tBblock' \in Block$
$\land lst\_tAblock' \leq fst\_tAblock'$
$\land lst\_tBblock' \leq fst\_tBblock'$
$\land fst\_tAblock' < lst\_tBblock'$
$\land end\_block(lst\_tBblock') \leq end\_trainB'$
$\land signal\_state' \in Signal \to SignalState)$

end

Event move_front_trainA ⟨ordinary⟩ $\widehat{=}$
External event, DO NOT REFINE

any
step
front_trainA

where
typing_front_trainA: ⟨theorem⟩ $front\_trainA \in \mathbb{Z}$
grd1: $step \in \mathbb{N}_1$
grd2: $front\_trainA + step < front\_block(fst\_tAblock)$
grd3: $ww = tt$

then
act2: $tt := 1 - tt$

end

Event move_end_trainA ⟨ordinary⟩ $\widehat{=}$
External event, DO NOT REFINE

any
step
end_trainA
front_trainA

where

typing_end_trainA: ⟨theorem⟩ $end\_trainA \in \mathbb{Z}$
typing_front_trainA: ⟨theorem⟩ $front\_trainA \in \mathbb{Z}$
grd1: $step \in \mathbb{N}_1$
grd2: $end\_trainA + step < front\_trainA$
grd3: $end\_trainA + step < front\_block(lst\_tAblock)$
grd4: $ww = tt$

then

act2: $tt := 1 - tt$

end

Event free_tAblock ⟨ordinary⟩ $\widehat{=}$

External event, DO NOT REFINE

any

step
end_trainA
front_trainA

where

typing_end_trainA: ⟨theorem⟩ $end\_trainA \in \mathbb{Z}$
typing_front_trainA: ⟨theorem⟩ $front\_trainA \in \mathbb{Z}$
grd1: $step \in \mathbb{N}_1$
grd2: $end\_trainA + step < front\_trainA$
grd3: $front\_block(lst\_tAblock) < end\_trainA + step$
grd4: $end\_trainA + step < front\_block(next\_block(lst\_tAblock))$
grd5: $next\_block(lst\_tAblock) \leq fst\_tAblock$
grd6: $ww = tt$

then

act1: $tt := 1 - tt$
act3: $lst\_tAblock := next\_block(lst\_tAblock)$

end

Event Enter_tAblock ⟨ordinary⟩ $\widehat{=}$

External event, DO NOT REFINE

any

step
front_trainA

where

typing_front_trainA: ⟨theorem⟩ $front\_trainA \in \mathbb{Z}$
grd1: $step \in \mathbb{N}_1$
grd2: $signal\_state(next\_block(fst\_tAblock)) = green$
grd3: $front\_block(fst\_tAblock) < front\_trainA + step$
grd4: $front\_trainA + step < front\_block(next\_block(fst\_tAblock))$
grd5: $ww = tt$

then

act1: $tt := 1 - tt$
act3: $fst\_tAblock := next\_block(fst\_tAblock)$

end

Event move_front_trainB ⟨ordinary⟩ $\widehat{=}$

External event, DO NOT REFINE

any

step

front_trainB

where

typing_front_trainB: ⟨theorem⟩ $front\_trainB \in \mathbb{Z}$

grd1: $step \in \mathbb{N}_1$

grd2: $front\_trainB + step < front\_block(fst\_tBblock)$

grd3: $ww = tt$

then

act2: $tt := 1 - tt$

end

Event move_end_trainB ⟨ordinary⟩ $\widehat{=}$

External event, DO NOT REFINE

any

step

end_trainB

front_trainB

where

typing_end_trainB: ⟨theorem⟩ $end\_trainB \in \mathbb{Z}$

typing_front_trainB: ⟨theorem⟩ $front\_trainB \in \mathbb{Z}$

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainB + step < front\_trainB$

grd3: $end\_trainB + step < front\_block(lst\_tBblock)$

grd4: $ww = tt$

then

act2: $tt := 1 - tt$

end

Event free_tBblock ⟨ordinary⟩ $\widehat{=}$

External event, DO NOT REFINE

any

step

end_trainB

front_trainB

where

typing_end_trainB: ⟨theorem⟩ $end\_trainB \in \mathbb{Z}$

typing_front_trainB: ⟨theorem⟩ $front\_trainB \in \mathbb{Z}$

grd1: $step \in \mathbb{N}_1$

grd2: $end\_trainB + step < front\_trainB$

grd3: $front\_block(lst\_tBblock) < end\_trainB + step$

grd4: $end\_trainB + step < front\_block(next\_block(lst\_tBblock))$

grd5: $next\_block(lst\_tBblock) \leq fst\_tBblock$

grd6: $ww = tt$

then

act1: $tt := 1 - tt$

act3: $lst\_tBblock := next\_block(lst\_tBblock)$

end

Event Enter_tBblock ⟨ordinary⟩ $\widehat{=}$

External event, DO NOT REFINE

any

step

front_trainB

where

typing_front_trainB: ⟨theorem⟩ $front\_trainB \in \mathbb{Z}$

grd1: $step \in \mathbb{N}_1$

grd2: $signal\_state(next\_block(fst\_tBblock)) = green$

grd3: $front\_block(fst\_tBblock) < front\_trainB + step$

grd4: $front\_trainB + step < front\_block(next\_block(fst\_tBblock))$

grd5: $ww = tt$

then

act1: $tt := 1 - tt$

act3: $fst\_tBblock := next\_block(fst\_tBblock)$

end

Event TRACKevent ⟨ordinary⟩ $\widehat{=}$

when

grd1: $ww \neq tt$

then

act1: $ww := 1 - ww$

act2: $block\_state := (Block \times \{Free\}) \lhd ((lst\_tAblock \mathinner{..} fst\_tAblock \cup lst\_tBblock \mathinner{..} fst\_tBblock) \times \{Occupied\})$

act3: $signal\_state := (Signal \times \{green\}) \lhd ((signal\_block(lst\_tAblock) \mathinner{..} signal\_block(fst\_tAblock) \cup signal\_block(lst\_tBblock) \mathinner{..} signal\_block(fst\_tBblock)) \times \{red\})$

end

END

APPENDIX C

DECOMPOSITION OF THE CASE STUDY USING REFINEMENT SEES SPLIT (RSS)

C.1 First Sub-system: Train

MACHINE Train
REFSEES Track
SEES Signal
VARIABLES
front_trainA
front_trainB
end_trainA
end_trainB
fst_tAblock
lst_tAblock
fst_tBblock
lst_tBblock
next_turn
EVENTS
Initialisation
begin
act1:
$front\_trainA$,
$front\_trainB$,
end_trainA,
end_trainB,
$fst\_tAblock$,
$lst\_tAblock$,

$fst\_tBblock$,
$lst\_tBblock$,
$next\_turn :|$ (
$front\_trainA' \in \mathbb{N}$
$\wedge\ front\_trainB' \in \mathbb{N}$
$\wedge\ end\_trainA' \in \mathbb{N}$
$\wedge\ end\_trainB' \in \mathbb{N}$
$\wedge\ end\_trainA' < front\_trainA'$
$\wedge\ end\_trainB' < front\_trainB'$
$\wedge\ front\_trainA' < end\_trainB'$
$\wedge\ fst\_tAblock' \in Block$
$\wedge\ lst\_tAblock' \in Block$
$\wedge\ fst\_tBblock' \in Block$
$\wedge\ lst\_tBblock' \in Block$
$\wedge\ next\_turn' \in SUBSYS$
$lst\_tAblock <= fst\_tAblock$
$lst\_tBblock <= fst\_tBblock$
$fst\_tAblock < lst\_tBblock$
$end\_block(lst\_tBblock) <= end\_trainB$

end

Event move_front_trainA ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $front\_trainA + step < front\_block(fst\_tAblock)$
grd3: $next\_turn = TRAIN$

then

act1: $front\_trainA := front\_trainA + step$
act2: $next\_turn := TRACK$

end

Event move_end_trainA ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $end\_trainA + step < front\_block(lst\_tAblock)$
grd3: $next\_turn = TRAIN$

then

act1: $end\_trainA := end\_trainA + step$
act2: $next\_turn := TRACK$

end

Event free_tAblock ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $front\_block(lst\_tAblock) < end\_trainA + step$
grd3: $end\_trainA + step < front\_block(next\_block(lst\_tAblock))$
grd4: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$
act2: $end\_trainA := end\_trainA + step$
act3: $lst\_tAblock := next\_block(lst\_tAblock)$

end

Event Enter_tAblock ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $block\_state(next\_block(fst\_tAblock)) = Free$
grd3: $front\_block(fst\_tAblock) < front\_trainA + step$
grd4: $front\_trainA + step < front\_block(next\_block(fst\_tAblock))$
grd5: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$
act2: $front\_trainA := front\_trainA + step$
act3: $fst\_tAblock := next\_block(fst\_tAblock)$

end

Event move_front_trainB ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $front\_trainB + step < front\_block(fst\_tBblock)$
grd3: $next\_turn = TRAIN$

then

act1: $front\_trainB := front\_trainB + step$
act2: $next\_turn := TRACK$

end

Event move_end_trainB ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$
grd2: $end\_trainB + step < front\_block(lst\_tBblock)$
grd3: $next\_turn = TRAIN$

then

act1: $end\_trainB := end\_trainB + step$

act2: $next\_turn := TRACK$

end

Event free_tBblock ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $front\_block(lst\_tBblock) < end\_trainB + step$

grd3: $end\_trainB + step < front\_block(next\_block(lst\_tBblock))$

grd4: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$

act2: $end\_trainB := end\_trainB + step$

act3: $lst\_tBblock := next\_block(lst\_tBblock)$

end

Event Enter_tBblock ⟨ordinary⟩ $\widehat{=}$

any

step

where

grd1: $step \in \mathbb{N}_1$

grd2: $block\_state(next\_block(fst\_tBblock)) = Free$

grd3: $front\_block(fst\_tBblock) < front\_trainB + step$

grd4: $front\_trainB + step < front\_block(next\_block(fst\_tBblock))$

grd5: $next\_turn = TRAIN$

then

act1: $next\_turn := TRACK$

act2: $front\_trainB := front\_trainB + step$

act3: $fst\_tBblock := next\_block(fst\_tBblock)$

end

END

C.2 Second Sub-system: Track

MACHINE Track

REFSEES Train

SEES Signal

VARIABLES

signal_state

block_state

next_turn

INVARIANTS

inv1: $signal\_state \in Signal \to SignalState$

inv2:

$\forall bk \cdot ($
$bk \in Block$
$\land\ next\_turn = TRAIN$
$\land\ signal\_state(signal\_block(bk)) = green$
$\Rightarrow block\_state(bk) = Free)$

EVENTS

Initialisation

begin

act1:

$signal\_state,$
$block\_state,$
$next\_turn :| ($
$\land\ block\_state' \in Block \to BlockState$
$\land\ next\_turn' \in SUBSYS$
$\land\ signal\_state \in Signal \to SignalState$
$\forall bk \cdot ($
$bk \in Block$
$\land\ next\_turn = TRAIN$
$\land\ block\_state(bk) = Free$
$\Rightarrow bk \neq lst\_tBblock)$
$\forall bk \cdot ($
$bk \in Block$
$\land\ next\_turn = TRAIN$
$\land\ signal\_state(signal\_block(bk)) = green$
$\Rightarrow block\_state(bk) = Free))$

end

Event TRACKevent ⟨ordinary⟩ $\widehat{=}$

when

grd1: $next\_turn = TRACK$

then

act1: $next\_turn := TRAIN$

act2: $block\_state := (Block \times \{Free\}) \lhd ((lst\_tAblock \mathinner{..} fst\_tAblock \cup lst\_tBblock \mathinner{..} fst\_tBblock) \times \{Occupied\})$

act3: $signal\_state := (Signal \times \{green\}) \lhd ((signal\_block(lst\_tAblock) \mathinner{..} signal\_block(fst\_tAblock) \cup signal\_block(lst\_tBblock) \mathinner{..} signal\_block(fst\_tB2block)) \times \{red\})$

end

END

RÉSUMÉ LONG EN FRANÇAIS

Le travail présenté dans cette thèse concerne la définition d'une nouvelle approche de la décomposition en méthode B événementiel. Son objectif est de proposer une solution méthodologique et opérationnelle pour décomposer un système critique pour la sécurité en plusieurs composants par rapport au comportement de la spécification globale. Cette solution doit permettre à l'ingénieur de préciser toutes les caractéristiques considérées comme pertinentes pour la modélisation d'un système, mais elle doit aussi offrir la possibilité de partitionner ce système au fil des étapes de raffinement.

L'étude bibliographique, réalisée sur le B événementiel et les approches de la décomposition sur la modélisation formelle en B événementiel, nous a permis d'identifier les techniques de modularisation les plus citées et utilisées en B événementiel : décomposition par variables partagées et décomposition par événements partagés. Le premier permet de partitionner la fonctionnalité du système et le second décompose le comportement du système. Bien que ces approches puissent diviser les systèmes en plusieurs sous-systèmes, il existe certaines limites et certaines difficultés concernant les besoins industriels. Par exemple, la difficulté de décomposer des prédicats complexes et la nécessité de plusieurs étapes intermédiaires de raffinement pour décomposer peuvent être rencontrées [Abrial, 2009, Kraibi et al., 2019b].

Dans le cadre du projet *PRESCOM*, nous avons travaillé sur un contexte industriel spécifique: la modélisation des systèmes ferroviaires. Ce domaine d'application des systèmes critiques pour la sécurité reconnaît la pertinence de la décomposition dans la modélisation des systèmes ferroviaires. Il identifie certains besoins pour séparer les principales caractéristiques du système de spécification abstraite dans un certain nombre de sous-systèmes de niveau inférieur. Cette séparation est effectuée conformément à l'objectif prévu de la modélisation du système pour obtenir des spécifications plus lisibles et gérables.

La proposition présentée dans cette thèse pour la décomposition d'un système critique pour la sécurité peut être résumée par ces quatre points :

– Une méthode de décomposition pour partitionner les systèmes en sous-systèmes, indépendamment de tout outil Event-B ;

- Démonstration de la correction de la solution proposée ;
- Proposition de nouvelles règles d'obligations de preuve supplémentaires nécessaires ;
- Illustration de la méthode *Refinement Seen Split* (*RSS*) par son application sur un cas concret de signalisation ferroviaire validée par les experts du domaine.

Tous ces points sont décrits en détail ci-dessous.

Le premier concerne une méthode de décomposition pour partitionner les systèmes. En fait, l'approche de décomposition que nous proposons dans cette thèse permet de partitionner un système en plusieurs sous-systèmes. Notre approche est basée sur la notion de raffinement et les méthodes de décomposition existantes dans la littérature. Ceci se fait selon une stratégie spécifique afin de fractionner correctement les invariants, les variables et les événements du système. La stratégie définit, en général, les différents cas qui peuvent survenir dans un modèle et la méthodologie à suivre dans chaque cas. De plus, afin de conserver la cohérence sémantique du comportement du système, une nouvelle clause nommée REFSEES est définie. Cette clause est un lien sémantique entre les sous-machines. Il garantit, à une certaine sous-machine, la visibilité des propriétés, les variables et les invariants des autres sous-machines résultantes de la décomposition de la même machine initiale.

Le deuxième point porte sur la correction de l'approche Refinement Seen Split (RSS). En effet, la formalisation de l'approche de décomposition proposée nécessite l'assurance de sa correction. En conséquence, nous vérifions cette exactitude par une démonstration. En effet, après décomposition d'une machine initiale *M*, la fusion des sous-machines résultantes *MRG* constitue un raffinement de la machine décomposée par construction. Cela se justifie par la stratégie définie à suivre. Il permet de conserver chaque élément du système quelque part dans les sous-machines et la clause REFSEES relie entre elles. Après cette étape de décomposition, chaque sous-machine peut être raffinée indépendamment, et le nombre de raffinements varie d'une sous-machine à l'autre en fonction du besoin. À un certain niveau de raffinement, nous démontrons que la fusion des sous-machines résultantes *MRG'* après le raffinement constitue un raffinement de la première fusion *MRG*. Par conséquent, nous pouvons en déduire que l'ensemble de la sous-machine *MRG'* est un raffinement de la machine initiale *M*.

Le troisième point concerne la définition de nouvelles règles d'obligation de preuve supplémentaires. En effet, la proposition de notre nouvelle méthode de décomposition en B événementiel conduit à une définition indispensable de nouvelles règles d'obligations de preuve. En réalité, nous distinguons deux types de règles d'obligation de preuve : locale et globale. Les règles d'obligations de preuve locales sont celles définies, classiquement, par le langage B événementiel. Ils sont considérés comme internes à chaque sous-machine. Les règles d'obligations de preuve globales concernent l'ensemble du système. En d'autres termes, il prend en considération la fusion des sous-machines. Dans notre travail, nous définissons de nouvelles règles d'obligation de preuve pour la partie globale. Malgré le fait qu'un système soit correct, il n'est pas garanti qu'il ne se bloque pas ou ne s'exécute pas indéfiniment alors que ce n'était pas le cas auparavant. Par exemple, un train qui ne bouge pas est considéré comme sûr, mais il n'accomplit pas sa tâche : le transport de passagers ou de marchandises. En pratique, puisque nous décomposons, il y a une possibilité de perdre certains comportements du système décomposé. Par conséquent, la liberté de blocage et les règles

d'obligations de preuve variantes sont définies afin de faire face à cette possibilité.

Le quatrième point concerne l'application de l'approche sur une étude de cas ferroviaire. Afin d'illustrer notre contribution et de la comparer avec les autres méthodes de décomposition étudiées, nous appliquons notre approche à l'étude de cas ferroviaire. On remarque que l'on peut appliquer le *Refinement Seen Split* (*RSS*) sans aucune étape intermédiaire de raffinement afin de simplifier le modèle. De plus, il existe un lien de communication entre ces sous-machines afin qu'elles puissent échanger la visibilité des états des variables et des invariants.

Pour résumer, le raffinement et la décomposition sont définis, dans la littérature, de manière à pouvoir coexister dans le processus de modélisation formelle afin de gérer la complexité du système à travers de multiples niveaux d'abstraction. Cependant, certaines approches de décomposition existantes ne reposent pas sur l'exactitude du raffinement pour définir la décomposition et préserver la cohérence syntaxique/sémantique du début au niveau inférieur de la modélisation. De plus, les mécanismes de décomposition existants sont définis comme des solutions associées à un outil spécifique *Rodin*.

Nous pensons que certains aspects méthodologiques liés à la méthode B événementiel que nous avons utilisée comme la génération d'obligations de preuve, la reconstruction de traces d'événements ou le fait de faire fusionner deux modèles tout en conservant les preuves déjà effectuées sur ces modèles peuvent être réutilisés dans un autre modèle de conception dans B événementiel. Il est indépendant de tout outil existant.

De nombreuses méthodes ont été proposées pour la décomposition des systèmes pour le B événementiel. Mais il existe peu d'études de cas complexes où ces techniques ont été appliquées. Nous pensons avoir apporté une contribution dans cette direction avec la décomposition des systèmes critiques pour la sécurité.

Towards a Modular Architecture of Formal Modelling:
System/Sub-systems Decomposition in Event-B

Abstract: The activities of analysis and modelling of critical systems, such as railway systems, are large-scale tasks requiring rigorous mechanisms. Founded on mathematical bases, formal methods can help to rigorously conduct these activities and reduce the ambiguity of the specifics of these systems. The Event-B method is one of the most widely used and recommended methods for system modelling. The central mechanism of Event-B system modelling is refinement. Indeed, the refinement consists in detailing abstract specifications in order to obtain more concrete specifications. In addition, the refinement process must be proven in order to ensure consistency and correctness of the system modelling between two levels of refinement. Although the Event-B method has a refinement mechanism to move from an abstract level to a finer level of granularity, the formal models for such systems are often complex and large. In addition, it is difficult to communicate around these models between the different trades (business experts in the field, system engineers, subsystems engineers, etc.) and to manage the different provided bricks of the system. This requires, in the majority of cases, a manual intervention ensuring the synergy between these actors.

In order to have better communication and management, the decomposition has emerged as a technique that complements refinement. This mechanism aims to reduce the complexity of the initial model by its partitioning into sub-models, and consequently to facilitate formal verification activities. In the literature, the proposed decomposition approaches have some limitations regarding the industrial needs expressed in the context of critical systems, among others, system/sub-systems reasoning. The application of these approaches on such systems is particularly difficult and requires intermediate stages of refinement. Indeed, these decomposition methods can lead to a loss of some properties of the system such as security properties and/or an inconsistency of the behaviour expressed in the sub-models with that of the initial model.

On the basis of this problematic, the thesis subject is focused on the definition of a new modular approach for modelling critical systems based on the Event-B decomposition. This approach focuses on the decomposition of a system into several sub-systems while preserving the behaviour of the overall system. This is ensured by the definition of new semantic links as well as new rules for the generation of the associated proof obligations. The correctness of the proposed approach is ensured by demonstrating that the set of resulting components, after decomposition, constitutes a refinement of the initial decomposed system. This methodology is illustrated by a concrete case study from the rail sector.

Keywords: Decomposition, Refinement, Formal Modelling, Verification and Validation, Event-B, Railway System.

Vers une Architecture Modulaire de Modélisation Formelle :
Décomposition Système/Sous-systèmes en B Événementiel

Résumé : Les activités d'analyse et de modélisation des systèmes critiques, tels que les systèmes ferroviaires, constituent des tâches d'envergure nécessitant des mécanismes rigoureux. Fondées sur des bases mathématiques, les méthodes formelles peuvent aider à mener rigoureusement ces activités et à réduire l'ambiguïté des spécificités de ces systèmes. La méthode B événementiel fait partie des méthodes les plus utilisées et recommandées pour la modélisation système. Le mécanisme central d'une modélisation système en B événementiel est le raffinement. En effet, le raffinement consiste à détailler des spécifications abstraites afin d'obtenir des spécifications plus concrètes. En outre, le processus de raffinement doit être prouvé afin d'assurer la cohérence et la correction de la modélisation du système entre deux niveaux de raffinement. Bien que la méthode B événementiel dispose d'un mécanisme de raffinement permettant de passer d'un niveau abstrait à un niveau de granularité plus fine, les modèles formels pour de tels systèmes sont souvent complexes et volumineux. En outre, il est difficile de communiquer autour de ces modèles entre les différents corps de métier (les experts métier du domaine, les ingénieurs système, les ingénieurs sous-systèmes, etc.) et de gérer les différentes briques du système fournies. Ceci nécessite, dans la majorité des cas, une intervention manuelle assurant la synergie entre ces acteurs.

Dans le but d'avoir une meilleure communication et gestion, la décomposition est apparue comme technique qui complète le raffinement. Ce mécanisme a pour but de diminuer la complexité du modèle initial en le partitionnant en sous-modèles, et de faciliter par conséquent les activités de vérification formelle. Les approches de décomposition proposées dans la littérature ont quelques limitations au vu des besoins industriels exprimés dans le cadre des systèmes critiques, entre autres, le raisonnement système/sous-systèmes. L'application de ces approches sur de tels systèmes est particulièrement difficile et exige des étapes intermédiaires de raffinement. En effet, ces méthodes de décomposition peuvent entraîner une perte de quelques propriétés du système comme les propriétés de sécurité ou une incohérence du comportement exprimé dans les sous-modèles avec celui du modèle initial.

Sur la base de cette problématique, le sujet de thèse est focalisé sur la définition d'une nouvelle approche modulaire de modélisation des systèmes critiques basée sur la décomposition en B événementiel. Cette approche porte sur la décomposition d'un système en plusieurs sous-systèmes en préservant le comportement du système global. Cela est assuré par la définition de nouveaux liens sémantiques ainsi que de nouvelles règles pour la génération des obligations de preuve associées. La correction de l'approche proposée est assurée en démontrant que l'ensemble des composants résultants, après la décomposition, constitue un raffinement du système initial décomposé. Cette méthodologie est illustrée par un cas d'étude concret issu du secteur ferroviaire.

Mots clés : Décomposition, Raffinement, Modélisation Formelle, Vérification et Validation, B Événementiel, Système Ferroviaire.