A formal approach for modeling high-level architectures of complex systems aligned with requirement models

Racem Bougacha

centralelille

CENTRALE LILLE

THÈSE

présentée en vue d'obtenir le grade de

DOCTEUR

en

Spécialité : Informatique et applications

par

Racem BOUGACHA

DOCTORAT DELIVRÉ PAR CENTRALE LILLE

Titre de la thèse :

Une approche formelle pour la modélisation d'architectures de haut niveau de systèmes complexes alignées avec les modèles d'exigences

---

Soutenue le 7 Juillet 2023 devant le jury d'examen :

President:	Sophie EBERSOLD, Professeure, *Université de Toulouse*
Examinateur:	Akram IDANI, Maître de conférences, *Grenoble INP*
Examinateur:	Sana DEBBECH, Ingénieur de recherche, *IRT Railenium*
Rapporteur:	Christian ATTIOGBE, Professeur, *Université de Nantes*
Rapporteur:	Iulian OBER, Professeur, *ISAE-SUPAERO*
Directeur de thèse:	Simon COLLART-DUTILLEUL, Directeur de recherche, *Université Gustave Eiffel*
Co-Directeur de thèse :	Régine LALEAU, Professeure, *Université Paris-Est Créteil*

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 MADIS 631

Université Gustave Eiffel — LABORATOIRE ESTAS ÉVALUATION DES SYSTÈMES DE TRANSPORTS AUTOMATISÉS ET DE LEUR SÉCURITÉ

RAILENIUM RAIL RESEARCH & INNOVATION

Abstract

Complex systems are a collection of sub-systems linked together to form an integrated whole. The sub-systems are generally heterogeneous in that they integrate various kinds of components, as mechanical, electronic, or software components, working together to perform missions. Therefore their design requires the collaboration of domain experts and the use of a common language to communicate with each other and agree on the main characteristics of the system to build. To achieve this, graphical models are generally recommended to specify, view, understand, and document the system in a simple way. However, when considering safety-critical systems where the consequences of a failure result in loss of life, significant property or environmental damage, graphical languages are not sufficient since they are only semi-formal and do not allow formal and rigorous reasoning necessary for verifying safety and security properties. Nowadays, the usefulness of formal verification and validation of system specifications is well established, at least for critical systems. Lastly, the quality of a system depends on the degree to which it fulfills its requirements. Requirements traceability is broadly recognized as a crucial element of any rigorous system development process, especially for the design of critical complex systems.

To cope with these issues, the thesis aims to define an approach of alignment between requirements models and high-level architecture models for safety-critical complex systems, thus allowing to specify traceability links between these two entities and to guarantee that high-architecture models fulfills required stakeholders needs. This is achieved by using formal specifications to verify firstly the correctness and consistency of high-level architecture models and secondly the consistency of the established alignment links. The idea behind the first point is to combine SysML, well adapted to be validated by domain experts, and the Event-B formal method for verification purposes. We propose to extend SysML with safety relevant Event-B refinement and decomposition mechanisms to model high-level architectures, and to define a set of rules enabling an automatic translation from SysML diagrams to Event-B specifications. We focus on diagrams that facilitate high-level architecture design, namely package, block-definition, state-transition and sequence diagrams. The second point aims to support traceability by defining several kinds of alignment links between requirements models and high-level architecture models. Alignment links are first graphically specified. Then they are translated into Event-B specifications. The main idea is to use the formal refinement concept to prove their correctness. As the semantics of this refinement is not the same as the standard Event-B refinement, we have defined new proof

obligations to express it, which can be discharged using AtelierB.

The proposed approach is supported by a set of tools and implemented in an Eclipse IDE plug-in. It has been evaluated on various industrial-scale case studies.

Keywords: SysML, High-level architecture, Requirement, Alignment, SysML/KAOS, Model Transformation, Formal Specification, Event-B Method.

Acknowledgment

First and foremost, I thank God for giving me strength, knowledge, ability and opportunity to undertake this research work and to continue and complete it satisfactorily. Without his blessings, this achievement would not have been possible.

Many thanks go to my thesis reviewers and the defense jury committee. Pr. Christian ATTIOGBE and Pr. Iulian OBER for accepting being my thesis reviewers and for their attention and thoughtful comments. I also thank Dr. Sana DEBBECH, Dr. Akram IDANI and Pr. Sophie EBERSOLD for accepting being my thesis examiners.

I would like to express my appreciation and gratitude to my supervisor Simon COLLART-DUTILLEUL. His valuable advice, patience, enthusiasm and constant support all the time of research allowed me to acquire new understandings and extend my experiences. Thank you for your guidance, it has been a true pleasure and I deeply hope that we can continue our collaboration.

A special gratitude is also due to my co-supervisor Pr. Régine LALEAU, for believing in me long after I had lost belief in myself, for her great advices and her guidance. I am thankful for the opportunities she provided, and for having faith in me. I am deeply grateful for the great deal of time we spent discussing many technical details of our work together.

my thanks goes to the institutes and companies that provided me all the support I needed to accomplish this work: RAILENIUM and COSYS-ESTAS, Université Gustave Eiffel. You gave me the financial, physical and intellectual support that was necessary during the whole journey of this thesis. I am very thankful to you. A very big thank goes to my colleagues and dear friends in RAILENIUM and COSYS-ESTAS lab. Thank you for always providing me with a simulating, productive and welcoming environment to which I have happily returned every day.

Similarly, I thank Dr. Slim Kallel, Associate Professor at the University of Sfax, Tunisia, the professor who introduced me to the research and taught me some of the most important basis I have. You are a strong reference in my life regarding organisation, honesty and determination. I am thankful for the support he provided, for intellectual guidance, for patient motivation, and for his good humour in my darkest hours.

I have been blessed with a very loving and supportive family. My parents, Raoudha and Mohamed, have always stressed the importance of education, which has, in some unconscious way, shaped my values and made me the person that I am today. I appreciate

all of the sacrifices that they have made for me. From my mom, I learned the art of working around obstacles, and the importance to stand up for myself. From my dad, I learned to work hard, and to be optimistic. I am grateful to the two of them for all that they have taught me about what it means to be a good person. During these years, despite our time together was brief, their contributions to my life will be felt forever. Your encouragement made me go forward and made me want to succeed. I am also grateful to my brother and sister Houssem and Wiem for being by my side through hard times. I cannot forget my dear little nieces Eya and Emna who filled me with joy and love and who always encouraged me in their innocent way.

I express my deepest gratitude to my soulmate and my loving fiancee Houda. As for you Houda, I find it difficult to express my appreciation in some lines. Thank you for sharing my wish to reach the goal of completing this thesis. I am grateful to her because she has given up so much to make my career a priority in our lives and tried to compensate for my stressful times. A tremendous help and endurance for a great number of years has helped me to complete this endeavour. So it only seems right that I dedicate this dissertation to her. My warm thanks to my parents-in-law, Rachid and Hedia for their support and kindness.

Last but not least, a warm thank goes to all my friends. Unfortunately, I cannot list everyone here. I do not want to miss anyone of you. However, it is a desire to extend my special gratitude to some of my closest friends: Sana, Jihen, Achraf, Ameni, Aslam, Fatma, Insaf, Mariem. I am grateful for all the encouragements, love, support and motivation I have received over the years and I hope I can give something back to everyone. Friends are everything.

Contents

List of Figures

List of Tables

Listings

Glossary

SysML Systems Modeling Language. XIII, XV, XIX, XX, 6, 8, 9, 16–18, 20–22, 26, 31–33, 41–45, 49–51, 54, 55, 57, 62–64, 66–73, 76–79, 81, 83–85, 87, 89–92, 96–100, 133, 138, 139, 151, 154, 160–164, 167–170, 172, 177, 178, 183, 184, 191, 193–199

SysML/KAOS . XIII–XV, 9, 10, 20–25, 46, 48, 55–57, 62–64, 91–94, 96, 97, 99, 100, 107–112, 114, 117, 118, 123–125, 177, 184, 185, 193, 194, 197–199

HLA High-Level Architecture. XIII–XV, 4–10, 26, 43, 44, 55–57, 61–71, 77–82, 86, 91, 92, 96–105, 107, 125–129, 136, 138–141, 145–148, 150, 151, 154, 156, 158, 159, 163, 164, 167, 170, 176–178, 180, 181, 183–186, 191–201

KAOS Knowledge Acquisition in autOmated Specification. 19–22, 50, 54, 57, 177

RE Requirements Engineering. XIII, 8, 10, 18–20, 48, 63, 92, 183, 193, 197, 200

INTRODUCTION

Contents

1 Context

Complex systems are a collection of sub-systems, which are independent enough to be identified. These sub-systems are generally heterogeneous in that they integrate various kinds of components, as mechanical, electronic, or software components, working together to perform a complex system mission. These systems are represented as composite systems, in the sense that the interaction between sub-systems is modeled by collaborating behaviors. Manipulation of these behaviors is the key to the resolution of composite system behavior. The goal of these manipulations is to reduce, as much as possible, the resolution of the original, composite system behavior, to the resolution of a sequence of sub-systems behaviors that can be solved independently. From literature, complex systems are a set of interconnected parts forming an integrated whole.

Nowadays, the importance of complex systems is increasing in the human life as the automation of essential tasks is taking place in the people routines. These systems may be responsible for such important tasks that their correct functioning is crucial in order to avoid severe repercussions. Complex systems are considered as safety-critical when the consequences of a failure result in loss of life, significant property or environmental damage. It requires advanced approaches for the verification of their correctness in order to avoid hazardous situations [Knight, 2002]. Some examples of these systems are Aircraft Flight Control, Railway, Medical Devices or Nuclear Systems.

The Autonomous Freight Train (AFT) project under the Autonomous Train program [1]

1. The Autonomous Train program https://railenium.eu/train-autonome/

is one of the R&D and innovation programs of IRT Railenium [2], a test and applied research center for the rail industry in France, with the cooperation of several partners (SNCF, Alstom Transport, Hitachi Rail STS, Capgemini Engineering and Apsys). They target performance improvements of the system thanks to the implementation of autonomy in railway operations. This system is considered as a safety-critical complex system, where it depends more and more on effective solutions that can address heterogeneity and the interplay of physical and software elements. Also, the use of modern verification approaches may be the differentiating factor in order to guarantee the consistency of these systems. It should be noted that AFT with the objective of avoiding the occurrence of several problems like the loss of people lives, injuries, severe environmental damage and economical loss, for instance, must guarantee the consistency of systems functionalities.

Several specialized fields are involved in the design of a complex system, making it difficult to keep a unified vision of this system and to manage its design. This leads to important and difficult problems of integration, directly related to both the huge number of basic components integrated at multiple levels, and the important scientific and technological heterogeneity of such systems (generally involving software, hardware/physical and human/organizational parts).

When attempting to operate or design complex systems such as railway systems, the main challenge is, therefore, to take into account the interrelations between sub-systems, while never considering the problem of the whole system at once. High level architecture (HLA) is a system architecture specification that defines how to create a global system composed of interacting distributed parts. The intent of HLA is a structural design that allows the reuse of capabilities available in different parts ultimately reducing the cost and time required to create a synthetic environment for a new purpose, and the possibility of distributed collaborative development of complex parts applications [Dahmann et al., 1998] such as complex systems. HLA is widely applicable across a full range of parts application areas, representing entities at many levels of resolution.

In order to prove the consistency of a system HLA and support its design, formal methods may be used. Based on mathematical foundations, formal specification methods allow system HLA modeling as a way to define and prove system properties. A recent study of a literature review conducted on 114 scientific publications on formal methods and railways presented in the European project ASTRail [ASTRAIL, 2017] claimed that "This analy-

2. Railenium. http://railenium.eu/fr/

sis has shown a dominance of the UML modeling language for high-level representation of system models, and a large variety of formal tools used, with a dominance of the tools associated to the B family (ProB and Atelier B)". The CENELEC 50128 norm [3] in the railway domain clearly recommends the use of formal methods for developing critical systems. Using formal methods, system properties and HLA correctness can be proved. In fact, the B formal method family has shown their successful application in several industrial railway projects [Behm et al., 1999, Lecomte et al., 2007].

Complex systems are generally made up heterogeneous sets of components and increasing in complexity which raises multiple problems relating to the completeness, consistency, non ambiguity and correctness of a design with respect to initial requirements. Their development process is most often challenging since it could be difficult to verify that stakeholders needs are satisfied, knowing that the consequence of a failure may be disastrous, especially for safety-critical systems. Beyond the fact that requirements models should be as precise as possible, it is necessary to verify that HLA models are aligned with them. In fact, requirement traceability is defined as "... the ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases" [Gotel and Finkelstein, 1994]. Indeed, a good alignment helps ensuring return on investment, and is key to a coherent governance. Thus, validation and verification activities take part in this flow to ensure the correctness of the design with respect to the initially specified needs.

2 Problem statement and motivations

The design of safety-critical complex systems, such as AFT, depends on solutions that can address the interplay between their sub-systems. These sub-systems interact by exchanging information in order to perform the main goal of the global system. Therefore, a model of HLA supporting layered hierarchy of components is needed. Such HLA must enable the specification of the main functional elements of a system, together with their interfaces and interactions. It constitutes a framework common to all the domain experts involved in the design of the system. In the AFT project, graphical representations of system components are recommended to specify, view, understand, and document the system

3. The CENELEC 50128. https://standards.globalspec.com/std/2023439/afnor-nf-en-50128

in a simple way. Such representations allow all the stakeholders to discuss and agree on the main characteristics of the system to build and allow to check if its HLA corresponds to their expected requirements. Whereas, railway norms for safety engineering assert that modeling railway systems using SysML allows to generate correct-by-construction components, which is a way to ensure that the needs fulfilled by the initial model will remain respected while it is supposed to be easily understood by railway experts. From this perspective, we formulate the following research question:

RQ1: Can we provide a common modeling language for high-level architectures that can support the interaction of sub-systems?

Complex systems complexity is increasing related to both the huge number of basic components integrated at multiple levels, and the important scientific and technological heterogeneity of such systems. This raises multiple problems relating to the completeness, consistency, non ambiguity and correctness of system design. However, when systems are complex, their structure cannot be described at a single level or with a single view; multi-scale descriptions are needed to understand them. This comes from the fact that a complex system is a collection of sub-systems presented as an integrated whole working together to perform a main mission and besides, these sub-systems have their own life and can exist independently from their participation in the main mission. Based on these considerations, it is possible to formulate a second research question as:

RQ2: How can we master the complexity of these complex systems?

Generally complex system HLA graphical design is semi-formal and its semantics are given in natural language, which does not allow formal and rigorous reasoning necessary for critical systems for which safety and security are major concerns. In such circumstances, the industry needs an effective approach for the verification of the critical systems in order to guarantee their safety. In this context, the use of formal methods is strongly recommended for the specification of systems HLA during the development of railway systems for example. Formal specification allows the proof of the system consistency by modeling its structure and behavior to be formally verified. Thus, it is possible to formulate the third research question as:

RQ3: How can we provide a formal specification of high-level architectures to verify their consistency?

Beyond the fact that complex systems HLA should be consistent and verified, it is necessary to verify that its models are aligned with system requirements which should be

precise as much as possible too. Indeed, the HLA design is most often challenging since it could be difficult to verify that stakeholders needs are satisfied. That is why, while formal specification allows the proof of the system consistency, it is recommended also to prove the consistency of this alignment. Thus, validation and verification activities should take part to ensure the correctness of the design with respect to the initially specified needs. From this context, a last research question can be formulated as:

RQ4: How can we establish and verify alignment links between high-level architectures and system requirements?

3 Objectives

In the light of the aforementioned shortcomings, the core interest of this thesis is to define a holistic methodology that supports a part of complex system life cycle from stakeholders requirements definition to HLA definitions with traceability respect between HLA and requirements.

To this endeavor, in the sequel, we present the objectives of this methodology:

- Proposing a modeling language for complex systems HLA which can hamper reasoning about system properties. This modeling language should allow to provide the full range required to deal effectively with the heterogeneity of complex systems HLA elements. Such a HLA model must enable the specification of the main functional elements of a system, together with its interfaces and interactions. It constitutes a framework common to all the domain experts involved in the design of the system.
- While complex systems are represented as an integrated whole of sub-systems collaborating together to perform a main task, the aim is to define a language to model this HLA as a layered hierarchy of sub-systems.
- Defining complexity mastering mechanisms which allow step-by-step design and detailing the parent system behavior by its sub-systems behaviors interplay. This is due to the huge number of sub-systems and components a complex system can encompass. These sub-systems can also exist independently.
- Graphical representations give a simple manner to specify systems HLA and could be understood by stakeholders. However, they are less powerful, semi-formal and does not allow formal and rigorous reasoning necessary for critical systems. On the other hand, formal methods provide a formal and rigorous reasoning about system

specification but it is not an easy practice for unfamiliar users of formal methods. Therefore, we intend to define an end to end process from HLA graphical design until HLA formal specification which allows to guarantee the consistency and the completeness of the graphically designed HLA. This process is crucial for safety-critical systems.

- Proposing solutions and techniques which aim to establish alignment and traceability between HLA models and requirements. Thus, it allows to demonstrate that HLA elements are aligned with the stakeholders needs and participating in their satisfaction while the competitiveness of a system relies on the degree to which HLA models fulfill its requirements.
- Formalization of alignment links. We aim to provide a consistent and correct definition of this alignment between HLA and requirements, which involves the use of formal notation for this purpose.

4 Contributions

In this section we layout the contributions of the thesis to come up with a holistic methodology for HLA definitions with traceability respect vis-a-vis requirements. The proposed methodology realizes the set of objectives discussed in Section 3. This includes the following contributions:

— **Overview on the existing researches in the field**

A survey of the current state of the art about SysML, RE, architecture modeling and specification approaches and Event-B formal method is elaborated. Furthermore, related work on requirements models and their formalization into formal specification, design models and their formal specification translation, models refinement and decomposition and finally requirements and architecture alignment are analyzed.

— **High-level architecture modeling and formal specification**

For HLA modeling and formal specification, a model-based approach enabling the design of HLA using SysML, answering the research question RQ1 and its translation into Event-B formal method answering research question RQ3 is outlined.

- **High-level architecture modeling**

  The particularity of the proposed approach is that it provides systems/sub-systems hierarchical modeling of HLA. In order to master complexity in HLA de-

sign, the approach defines solutions and mechanisms to solve it. More precisely, we adopt Event-B refinement and decomposition mechanisms by proposing an extension for SysML to enable step-by-step design and complexity reduction. This answers research question RQ2.

- **High-level architecture formal specification**

  For a formal specification of HLA, an automatic translation from extended SysML HLA models into Event-B formal specification is proposed. This translation is a two step process, model-to-model and model-to-text, and supports also the translation of the proposed refinement and decomposition mechanisms. Using the generated formal model, we investigate a model checking and theorem proving based verification process that aims at ensuring HLA consistency and correctness.

This proposed approach is illustrated by the landing gear system case study [Boniol and Wiels, 2014] to verify and evaluate it.

— **Requirements & high-level architecture alignment**

It is necessary to verify that HLA models are aligned with requirements and to determine how HLA elements can contribute in the satisfaction of the requirements. We have chosen to use SysML/KAOS to model requirements graphically and formally. We propose a model-based approach to define several kinds of alignment links between SysML/KAOS models and SYSML HLA models and their formalization into Event-B. This approach allows to answer research question RQ4.

- **Requirements & high-level architecture graphical alignment**

  For establishing traceability, we have chosen to apply this alignment between leaf goals from requirements models and sequence diagram messages from HLA models. To this aim, different kinds of alignment links are proposed, each of them defines a specific semantics and process on how a goal in a requirements model could be satisfied by HLA elements.

- **Requirements & high-level architecture alignment formalization**

  To prove the consistency of the proposed alignment links, an Event-B formalization is proposed. The main idea is to use the Event-B refinement concept to prove the correctness of alignment links. As the semantics of this refinement is not the same as the standard Event-B refinement, to achieve it, new sets

of refinement proof obligations are specified, one for each kind of alignment. Discharging these proof obligations allows to formally verify the satisfaction of a leaf goal by a set of HLA messages.

This proposed approach of alignment is illustrated by the landing gear system case study [Boniol and Wiels, 2014] enhanced with a second example taken from the train control system case study presented in [Lamsweerde, 2008] to verify the consistency of the proposed alignment links.

— **Implementation of the methodology**

The methodology is implemented using a collaborating set of tools, frameworks and Eclipse plug-ins. This combination of tools assists designers in specifying complex systems life cycle from requirements to HLA. Moreover, it enables to verify the consistency, correctness and completeness of requirements and HLA using an automatic mappings of SysML/KAOS models and HLA models to the formal language; Event-B. Adding to that, it provides mechanisms to establish graphically and formally alignment between HLA and requirements entities in order to prove the traceability between them. A subsequent formal verification using the AtelierB tools is conducted.

5 Outline of the Thesis

Following this introductory chapter, the thesis is organized in three chapters.

— Chapter I outlines the background of this work about RE, architecture modeling and Event-B formal method which is necessary for the remainder of the thesis. Afterwards, existing researches are analyzed and a comparison is drawn. As a result of this analysis, we motivate and subsequently present the proposed methodology.

— Chapter II details the contributions of the high-level architecture modeling aligned with system requirements methodology. It consists in the application of SysML/KAOS approach, HLA modeling and its automatic translation into Event-B and we present our contribution about requirements and high-level architecture alignment. Finally, to evaluate our methodology, we illustrate it on the landing gear system case study.

— The implementation details of the methodology is discussed throughout Chapter III. This chapter presents a review of the implementation tools and mechanisms to perform each of the steps of the methodology. The main goal of this step is to support

complex systems designers with a collaborating set of tools and plug-ins within an integrated development environment as Eclipse.

Finally, in chapter CONCLUSION AND PERSPECTIVES, we will summarize and conclude the thesis. A discussion will also be held to outline several directions for future work.

6 Publications

The contributions made while pursuing the research described in this thesis have been published in proceedings of conferences and journals. Some of the chapters of this thesis are partially based upon these published papers. They are respectively listed below:

1. BOUGACHA, Racem, LALEAU, Régine et COLLART-DUTILLEUL, Simon. Formal alignment of requirements models with high-level architecture models. In : *27th International Conference on Engineering of Complex Computer Systems (ICECCS 2023)*. IEEE, 2023.
2. COLLART-DUTILLEUL, Simon, BON, Philippe, BOUGACHA, Racem et LALEAU, Régine. MODEL ENGINEERING FOR CRITICAL SYSTEMS: THE ATO OVER ETCS FOR FREIGHT TRAINS USE CASE. In : *International Journal of Transport Development and Integration*. To be published.
3. BOUGACHA, Racem, LALEAU, Régine, BON, Philippe, et al. Modeling train systems: from high-level architecture graphical models to formal specifications. In : *CRISIS2022: 17th International Conference on Risks and Security of Internet and Systems*. Springer, To be published.
4. BON, Philippe, COLLART-DUTILLEUL, Simon, et BOUGACHA, Racem. ATO OVER ETCS: A SYSTEM ANALYSIS FOR FREIGHT TRAINS. *Computers in Railways XVIII: Railway Engineering Design and Operation, 2022*, vol. 213, p. 37.
5. BOUGACHA, Racem, LALEAU, Régine, COLLART-DUTILLEUL, Simon, et al. Extending SysML with Refinement and Decomposition Mechanisms to Generate Event-B Specifications. In : *Theoretical Aspects of Software Engineering: 16th International Symposium, TASE 2022, Cluj-Napoca, Romania, July 8–10, 2022, Proceedings*. Cham : Springer International Publishing, 2022. p. 256-273.
6. Bougacha, Racem. A Formal Approach for the Modeling of High-Level Architectures Aligned with System Requirements. In: Raschke, A., Méry, D., Houdek, F. (eds)

Rigorous State-Based Methods. ABZ 2020. Lecture Notes in Computer Science, vol 12071. Springer, Cham. https://doi.org/10.1007/978-3-030-48077-6_33

Chapter I

Literature Review

Contents

1 Introduction

This chapter is dedicated to the presentation of background knowledge and the related works about specifying and verifying requirements and architecture models. Furthermore, research efforts on requirements and architecture alignment are outlined. Particularly, in this chapter we give a broad overview about SysML, requirements engineering approaches, architecture modeling approaches and Event-B formal method. Then, we analyze the state of art in existing tools and methods about formal specification of requirements models, design models and correspondent formal specifications and finally, we detail works dealing with requirements and architecture alignment. We conclude this chapter with an evaluation of the presented works and, based on these challenges, we propose a holistic methodology which deals with alignment.

2 Background

2.1 Systems Modeling Language SysML

The Systems Modeling Language (SysML) [Holt and Perry, 2008, OMG, 2007] is a graphical modeling language supporting the analysis and specification of complex systems that may include hardware, software and human elements. Based on the unified modeling language (UML) [OMG, 1997] as a UML profile, SysML has been designed to be used in system engineering to satisfy the shortcomings of UML. UML was designed to be a common, semantically and syntactically rich visual modeling language. The UML profiles represent an integration of a light-weight mechanism in order to extend the languages based on the MOF (Meta Object Facility). In fact, profiles are used to customize UML for a specific domain through extension mechanisms that enrich the semantics and syntax of the language. However, the use of UML in system engineering applications has shown certain weaknesses that must be solved to provide an effective language for system engineers. As examples of shortcomings, we give:

— The need to describe the requirements directly in the model, and to ensure traceability to the architecture.

— The need to represent non-software elements and to specify their type (mechanical, circuit, hydraulic, wiring, sensor, etc.).

— The need to represent physical equations, constraints.

— The need to represent continuous flows (matter, energy, etc.).

— The need to represent logical/physical, structure/dynamic, etc. allocations.

2.1.1 Overview of SysML diagrams

SysML is composed of nine types of diagrams, each of which is dedicated to represent particular concepts in a system. It reuses some of the UML proposed 13 types of diagrams exactly as is, some have been modified and others have not been kept. These nine diagrams are classified in three categories:

— Structural diagrams

- **Block Definition Diagram (BDD):** In SysML it defines features of blocks and relationships between blocks such as associations, generalizations, and dependencies. It captures the definition of blocks in terms of properties and operations, and relationships such as a system hierarchy or a system classification tree.
- **Internal Block Diagram:** It is based on the UML composite structure and it specifies a structural aspect of the model. It is related to both the BDD and the parametric diagram and describes the internal structure of a block.

— Behavioral diagrams

- **Activity Diagram:** It shows the flows of data and control between actions. Activity modeling emphasizes the inputs, outputs, sequences, and conditions for coordinating other behaviors of a block.
- **Sequence Diagram:** It describes the flow of control between actors and systems (blocks) or between parts of a system. It shows the vertical sequence of messages exchanged between elements (lifelines) in an interaction. This sequence of messages represents "Scenarios" which highlight pertinent aspects of a particular situation.
- **State-Machine Diagram:** It models the behavior during the lifetime of a block.
- **Use Case Diagram:** It describes the usage of a system (subject) by its actors (environment) to achieve a goal, that is realized by the subject providing a set of services to selected actors.

— Transverse diagrams

- **Requirements Diagram:** It is intended to be used to represent system requirements and their relationships. A requirement may specify a function that a system must perform or a performance condition a system must achieve. SysML provides modeling constructs to represent text-based requirements and relate them to other modeling elements.
- **Parametric Diagram:** It enables constraints on system parameter values to be represented, such as performance, reliability and mass. A constraint is represented as a block that allows the definition of rules about the properties of a system or constraints that the system must obey.

2.1.2 SysML usage

SysML language is represented as a communication language between the different members of the development teams. It allows to unify the visual modeling principles using a set of diagrams, which makes it easy to learn, to use and to document. The introduction of SysML in this domain was not only for simplifying the modeling and the communication but also to offer the development community a good pillar to analyse the requirements of the system since it is the first steps of the development through a model driven process.Among the tools currently in use for SysML modeling, we can cite IBM/Rhapsody [IBM, 2013], No Magic/MagicDraw [CATIA, 2011], Papyrus (open source) [Papyrus, 2008]. These tools are used buy the AFT project partners.

2.2 Requirements Engineering

The quality of a software system is the main measure of its success, that depends on the degree to which it fulfills its requirements. Requirements definition is a careful evaluation of the need that a system should fulfill. It must describe why a system is needed, based on current or foreseen conditions, which may be internal operations or stemming from external markets. It presents system features which serve and satisfies the system context and defines how the system will be constructed [Ross and Schoman Jr, 1976]. Thus, Requirements Engineering (RE) is defined as the branch of software engineering concerned with the real-world goals. It must address the contextual goals why a software is needed, the functionalities the software has to accomplish to achieve those goals, and the constraints restricting how the software accomplishing those functions is to be designed

and implemented. Such goals, functions and constraints have to be mapped to precise specifications of software behavior [Van Lamsweerde, 2000]. They cover different types of concerns: functional concerns associated with the services to be provided and non-functional concerns associated with quality of service [Van Lamsweerde, 2001]. Goals are very important in the RE process while they allow to achieve requirements completeness, avoid irrelevant requirements, explain requirements to stakeholders, etc.

Goals modeling is based firstly on the identification of these goals, which may be explicitly stated by stakeholders or in preliminary material available to requirements engineers. Many other goals can be identified by refinement and by abstraction, just by asking **HOW** and **WHY** questions about the goals-requirements already available, others are identified by resolving conflicts among goals or obstacles to goal achievement.

Several goal oriented RE approaches and frameworks have been presented such as:

— i* [Yu, 1997]: is an agent-oriented modeling framework that can be used for requirements engineering, business process reengineering, organizational impact analysis, and software process modeling. The i* framework is used to model the environment of the system-to-be. It facilitates the analysis of the domain by allowing the modeler to represent the stakeholders of the system, their objectives, and their relationships.

— The NFR Framework [Chung et al., 2000]: focuses on the modeling and analysis of non-functional requirements. The goal of the framework is to put non-functional requirements foremost in the developer's mind.

— KAOS [Dardenne et al., 1993]: KAOS stands for *Knowledge Acquisition in autOmated Specification*, which is described as a multi-paradigm framework that allows to combine different levels of expression and reasoning: semi-formal for modeling and structuring goals, qualitative for selection among the alternatives, and formal, when needed, for more accurate reasoning.

Figure 1.1 presents the goal oriented RE frameworks used in 246 publications covered by the literature map of [Horkoff et al., 2016]. KAOS and i* appear in near to the same number of publications (13%), the most popular choice is to use goal modeling in general, without committing to a particular framework. It is also fairly common (7%) to significantly use multiple frameworks together.

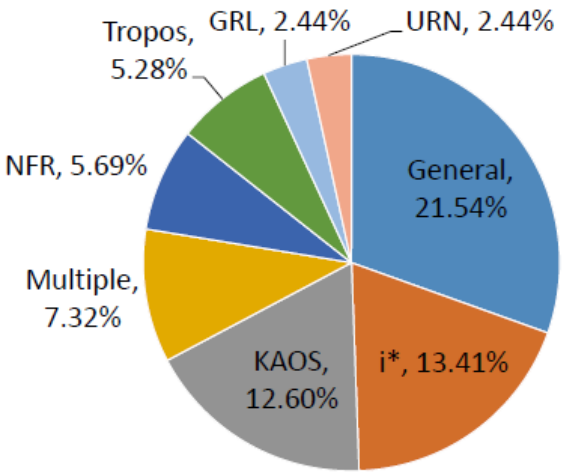

Figure 1.1 – Goal oriented RE frameworks use [Horkoff et al., 2016]

2.2.1 SysML/KAOS

SysML/KAOS [Laleau et al., 2010, Gnaho et al., 2013b] is a requirements engineering method that allows the modeling of functional and non-functional requirements of a system as goals to be achieved. The main idea is to extend SysML [OMG, 2007] with concepts from the goal model of the KAOS method [Van Lamsweerde, 2009]. The choice is on SysML/KAOS because it provides strong semantic expressiveness.

In fact, SysML is a modeling language for the analysis and specification of complex systems and it is well recommended by the AFT project partners. It is a UML profile that uses certain UML diagrams and also offers extensions such as modeling requirements. Despite, the concepts provided by SysML to represent requirements, they are not as extensive as in the other requirements methods because their semantics are not always clearly defined. That is why, the first step of SysML/KAOS approach is to extend the SysML requirements with concepts of the KAOS goal model, since it is the most important aspect of KAOS.

The KAOS approach [Van Lamsweerde, 2009] is a framework for eliciting, specifying, and analysing goals, requirements, scenarios, and responsibility assignments. It defines a requirements modeling language for the representation of requirements to be satisfied by the system and of expectations with regards to the environment through a hierarchy of goals. KAOS is based on the decomposition and refinement of goals to support the entire requirements development and acquisition process. The choice of KAOS is motivated by, firstly, it permits the expression of several models (goal, agent, object, behavioral models)

and relationships between them. Secondly, KAOS provides a powerful and extensive set of concepts to specify goal models. This allows the design of goal hierarchies with a high level of expressiveness that can be considered at different levels of abstraction. The most principal element is the **Goal model**. It shows the system functional and non-functional goals that contribute to each other through **AND/OR** refinements links from most abstract system goal down to software requirements and environment assumptions represented by the leaf goals. Refinement and abstraction paths in a goal model are build once preliminary goals are identified by recursively asking **HOW** and **WHY** questions about available goals, respectively. Knowing that preliminary goals may be obtained by analysing the strategic business objectives of the system-as-is by identifying the domain-specific objectives to be preserved across system versions, and by addressing the reported problems and complains about the system-as-is. A system-as-is is a reference model of the current way in which a group of actors deal with a particular situation and how it exists before the machine is built into it [Van Lamsweerde, 2009].

Figure 1.2 presents the extensions of SysML. Grey boxes represent the initial SysML concepts, while the white boxes represent the extended KAOS concepts. This figure presents a meta-model of SysML/KAOS.

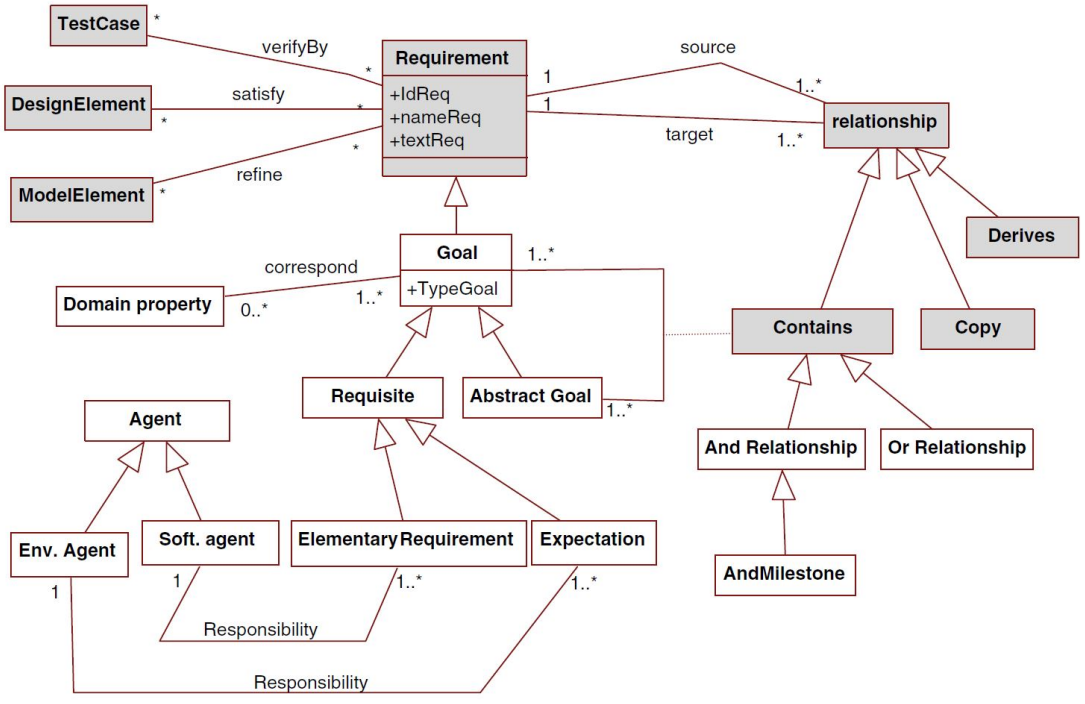

Figure 1.2 – Meta-model of the extended SysML [Laleau et al., 2010]

2.2.1.1 Functional Requirements Modeling

A functional requirement describes an expected behavior of the system, upon the occurrence of a specific condition. The SysML/KAOS functional requirements modeling language combines the SysML requirement/elements traceability with the expressiveness of the KAOS requirements modeling language. It allows the representation of the functional requirements of a system and the expectations regarding the environment in the form of hierarchies of goals. This hierarchy is defined using three types of operators **AND, OR** and **MILESTONE**. The **AND** operator is used when the only condition for the achievement of a goal is the achievement of each of its sub-goals. When the necessary and sufficient condition for the achievement of a goal is limited to the achievement of one of its sub-goals, then the operator **OR** is used. The **MILESTONE** operator makes it possible to sequence a set of sub-goals whose the satisfaction of the parent goal requires their realisation in order. SysML/KAOS also considers the data refinement that occurs when goals appearing in one level of refinement are re-expressed, within a subsequent level of refinement, due to the refinement of some data elements involved in their specification. To take into account the complexity of the systems, the SysML/KAOS method considers that the "first" constructed functional goal diagram, or diagram of the highest level, is that of the main system. The breakdown into sub-goals ends when a goal is no longer refined. Then, this goal can be either an elementary requirement which is placed under the responsibility of a system agent or an expectation which is placed under the responsibility of an agent of the system environment.

2.2.1.2 Non-Functional Requirements Modeling

Non-functional requirements capture the properties or constraints under which the system to be designed must operate, such as performance, quality or safety aspects. [Gnaho and Semmak, 2010], [Matoussi et al., 2011b] and [Gnaho et al., 2013a] propose solutions to define non-functional requirements and their impact on functional requirements

2.2.1.3 Domain Model Modeling

Modeling domain knowledge is one of the crucial factors to perform high quality requirements elicitation. SysML/KAOS goal model lacks of enough information to precisely describe the structural part of systems. Some research works such as KAOS use object

models to describe this part. [Tueno et al., 2017b, Tueno et al., 2017c] propose a meta-model to represent the domain model. In fact, they present a complementary work to the [Mammar and Laleau, 2016] works by modeling the domain using ontology. The ontology is defined as an explicit specification of a conceptualization.

2.2.2 SysML/KAOS to Event-B

A major remaining weakness in the development chain is the gap between textual or semi-formal requirements and formal models. This gap becomes larger and most of the approaches stop at requirement phase, so designers are obliged to use another method to develop their systems. Therefore, it is difficult to validate specifications with regard to requirements. SysML/KAOS combines requirements engineering methods with formal methods and defines a mapping rules to create a B specification from goal models.

2.2.2.1 SysML/KAOS functional goal formalization

The formalization of SysML/KAOS functional goal models is presented in [Matoussi et al., 2011a]. The proposed rules allow to generate an Event-B model whose structure reflects the hierarchy of the model of functional goals: a component is associated with each level of refinement of the hierarchy. This component defines the skeleton of an event for each goal of the level of refinement. The Event-B transformation of refinement patterns associated with goals is based on the classical set of inference rules from Event-B. Systematic proof obligations were identified for each goal refinement pattern.

We give as an example, the transformation of the MILESTONE refinement pattern. For an abstract event EvG and concrete events EvG1 and EvG2:

— THE MILESTONE GOAL REFINEMENT PATTERN: a syntactic extension of the Event-B refinement proof rule is presented in order to provide a way to refine an abstract event by a sequence of new events. For abstract event EvG, it is refined as follows: (EvG1 ; EvG2) Refines EvG. In fact, in addition to the feasibility proof obligation, this refinement pattern formalization requires to discharge these different proof obligations:

- The ordering constraint expresses the ”milestone” characteristic between the Event-B events.

$$\text{G1-PostCond} \Rightarrow \text{G2-Guard (PO1)}.$$

- The guard strengthening ensures that the concrete guard of the sequence (the guard of the first event in the sequence) implies the abstract guard.

  G1-Guard ⇒ G-Guard (PO2).

- The correct refinement ensures that the sequence (the action of the last event in the sequence) transforms the concrete variables in a way which does not contradict the abstract event.

  G2-PostCond ⇒ G-PostCond (PO3).

These proof obligations are introduced in the AtelierB [AtelierB, 1990] tool by extending the **ref** keyword which represents the Event-B refinement mechanism. This extension is stated as follows:

— MILESTONE refinement of an abstract event G by a sequence of concrete events G1, G2,..., GN is denoted by the following notation:
G1 ref_milestone G ; G2 ref_milestone G;...; GN ref_milestone G.

More details about these proof obligations generation can be found in [Matoussi et al., 2011a] and [ANR, 2014]. This work is carried out within the framework of the FORMOSE project [ANR-14-CE28-0009, 2014] funded by the French National Research Agency (ANR).

2.2.2.2 SysML/KAOS non-functional goal formalization

[Matoussi et al., 2011b] present a continuity of the work of [Gnaho and Semmak, 2010] to translate non-functional goals and their impacts to Event-B in order to enrich the formal specification created from functional requirements. They create a set of traceability rules to facilitate the management of evolution of these goals with different Event-B elements.

2.2.2.3 From domain model to Event-B specification

To provide a complete extraction of the structural part of the Event-B specification obtained from SysML/KAOS goal models and the initialisation of state variables, [Fotso et al., 2018d] proposes a set of rules to translate SysML/KAOS domain models (presented in 2.2.1.3) into Event-B specifications. These rules have been defined with Event-B and verified with RODIN [Abrial, 2010].

New elements may appear in the Event-B specification obtained from SysML/KAOS models when specifying the body of events and/or by using formal validation and/or verification tools. Moreover, modeling is often done through several backwards and forwards between the Event-B specification and SysML/KAOS models. [Fotso et al., 2018a] defines a set of rules allowing the back propagation, within domain models for every added element in the structural part of the Event-B specification. In fact, they describe these propagation rules and how they are specified in Event-B, proving their consistency using the RODIN tool.

2.2.2.4 Formalization of SysML/KAOS Goal Assignments with Event-B Component Decompositions

The use of formal methods for verification and validation of critical and complex systems is important, but can be extremely tedious without modularisation mechanisms. Several systems break down into sub-systems (enabling the distribution of work between several agents: hardware, software and human). SysML/KAOS goal models allow the capture of assignments of requirements to agents responsible for their achievement. Each agent is associated with a sub-system. [Fotso et al., 2018c] propose an approach to ensure that a requirement assigned to a sub-system is well achieved by the sub-system. This approach uses formal decomposition mechanisms [Abrial and Hallerstede, 2007] to construct, from the formal specification of a high-level system, the interface of each of its sub-systems. The interface of a sub-system describes the requirements that the high-level system expects from the sub-system. Proof obligations are defined to ensure that the invariants of each sub-system are consistent with that of the high-level system. The approach thus ensures that each sub-system achieves its expected goals with respect to constraints set by the high-level system.

2.3 Architecture Modeling

An architecture is a description of elements within a product and the interactions between them. These elements are grouped in a manner which fulfills some tasks that single element can not fulfill individually. It designates also how communication and interaction between elements is achieved.

System architecture is a conceptual model that describes the structure and behavior of multiple components and sub-systems like multiple software applications, network devices,

hardware, and even other machinery of a system. It is Architecture Description Language (ADL) [Clements, 1996] which describes the entire system architecture.

Software architecture refers to the process of creating high level structure of a software system. It is about the complete structure/architecture of the overall system means it converts software characteristics like scalability, security, reusability, extensibility, modularity, maintainability, etc. into structured solutions to meet the business requirement. High level architecture (HLA) [Dahmann et al., 1997] is a standard software architecture specification that defines how to create a global simulation composed of interacting distributed simulations. It contains major functional elements, interfaces, and design rules, providing a common framework within which specific system architectures can be defined.

Therefore, a system is generally understood to be an assemblage of components that integrates various mechanical, electronic, and information technology parts. These systems are classified as complex systems that can address heterogeneity and the interplay of physical and software elements. Several modeling languages have been proposed to reason about heterogeneous properties, and to develop optimized system-level solutions by assessing multidisciplinary design trade-offs. A number of these modeling languages have been standardized, such as Systems Modeling Language (SysML) [OMG, 2007] which focuses on the "big picture" architectural views, whereas others, such as Architecture Analysis and Design Language (AADL) [Feiler et al., 2006a] addresses the more detailed platform-oriented and physical aspects of such systems.

2.3.1 Architecture Analysis and Design Language AADL

Architecture Analysis & Design Language (AADL) [Feiler et al., 2006a, Feiler et al., 2006b] is a textual and graphical language used to design and analyze the software and hardware architecture of real-time systems and their performance-critical characteristics. It is aimed at supporting the avionics, aerospace, and automotive industry. From these experiments in avionics, flight control, and robotics applications, this language proved its richness and it offers expressive capabilities that go beyond the domain of avionics and it is extensible to other software and hardware systems. AADL can be expressed in text, in XML, as well as a graphic representation. It can be used by many different tools due to these representations, graphical or not. The development of profiles for UML also allows to integrate AADL within UML modeling tools. AADL was created to facilitate the interoperability of the various tools, this is why its reference syntax is textual. The XML

representation makes it easier to create parsers for existing applications. The graphical notation arises in addition to the textual notation, to facilitate the description of the architectures. It gives a better representation than text or XML, but it is less expressive. As an architecture description language, AADL describes the components connected to each other to form an architecture. An AADL description consists of a set of component declarations. These declarations can be instantiated to form the modeling of an architecture.

2.3.1.1 Components

AADL components are defined in two parts: the interface and the implementations. An AADL component has an interface (component type) to which correspond zero, one or more implementations. The interface presents a specification of the component. This specification is used by other system components to interact with the specified component. The type of an AADL component consists of three parts: interface elements, flows and properties. The implementation describes the internal structure of the component. Generally, a component is described as a set of sub-components. These sub-components are instances of interfaces or implementations of other components. An implementation also contains the connections that link the sub-components.

Component categories. AADL defines three components categories: (1) Hardware components describe elements of the execution platform (processors, memories, buses, etc.). (2) Software components describe the software entities that form an application (processes, sub-program, data, etc.). Finally, (3) systems that allow to regroup different components into logical entities to structure the architecture.

Interfaces and connections. An interface of a component provides "interface elements" (communication ports, parameters, etc.). Components communicate with each other by connecting their respective interface elements. An interface element models a feature that is visible to other components. In AADL, these elements are named entities allowing a component to exchange data and signals with the outside.

Annexes and properties. Annexes are another way to associate information with elements of a description. They allow to incorporate elements written in a language other than AADL into the model of an application. The use of annexes allows to extend the standard AADL syntax in order to specify the behavior of components.

AADL introduces the notion of properties. Properties are characteristics associated with different entities (components, connections, interface elements, etc.). These are attributes which allow to specify characteristics or constraints applying to the architecture elements: frequency of a processor, worst execution time of a process, bandwidth of a bus... A set of standard properties is defined in the language; but it is possible to define properties specific to a given application.

2.3.1.2 AADL tools

OSATE the Open Source AADL Tool Environment [Feiler and Greenhouse, 2005] was developed by the Software Engineering Institute on the Eclipse platform and Eclipse Modeling Framework (EMF) [Budinsky et al., 2004]. It supports the full language standard, text and graphical editing, semantic checking, translation into the standardized XML for AADL. The graphical editing capability is provided through TOPCASED [Gaufillet, 2005]. It is available at http://www.aadl.info. Also, the UML profile allows UML tool vendors to provide support for the AADL. Current commercial design tools can also be modified and extended to support the AADL. Ellidiss (www.ellidiss.com) has extended their HOOD development environment to support modeling in AADL as well as HOOD and UML, and to import/export AADL models. A number of toolsets based on AADL are becoming available. Ocarina [Vergnaud, 2005] is a tool suite to manipulate AADL models and generate distributed applications by automatically producing Object Request Broker (ORB) based middleware based on AADL models of the distributed application. The Furness toolset [Sokolsky, 2005] uses AADL as a front-end for the Algebra of Communicating Shared Resources (ACSR) [Clarke et al., 1993] for formal analysis of concurrent resource utilization and scheduling. Cheddar [Singhoff et al., 2005] is a real time scheduling tool designed for checking task temporal constraints of real time applications.

2.3.1.3 AADL usage

AADL allows to describe architectures with a very concrete approach. The language offers a set of component categories of three main types: software, hardware and systems. AADL focuses on the architectural aspects: it allows the description of the components and their connections, but does not allow to represent their behavioral implementation, nor the semantics of the manipulated data. These aspects can be added using the annexe mechanism, or by associating external descriptions using properties. In the same way, the

various constraints applying to the system or the deployment of the applications on the hardware can be expressed by means of properties.

2.3.2 Modeling and Analysis of Real-Time and Embedded systems MARTE profile

MARTE (Modeling and Analysis of Real-Time and Embedded systems) [OMG, 2008] is an Object Management Group (OMG) standard UML profile, inspired by the uml profile for Schedulability, Performance and Time [OMG, 2005]. The MARTE profile provides constructs to accurately model non-functional properties, time, and resources. It allows to model hardware and software execution platforms, and to allocate elements of a real-time application embedded on these platforms. It also allows to apply quantitative analyzes (scheduling or performance analysis) from these models, and validate the system design. These extensions are generic and are not dedicated to any particular execution model, analysis technique or implementation technology. MARTE is therefore applicable to a wide variety of engineering methodologies and processes. It can be considered as being to the field of embedded real time while UML is to the field of software.

2.3.2.1 MARTE architecture

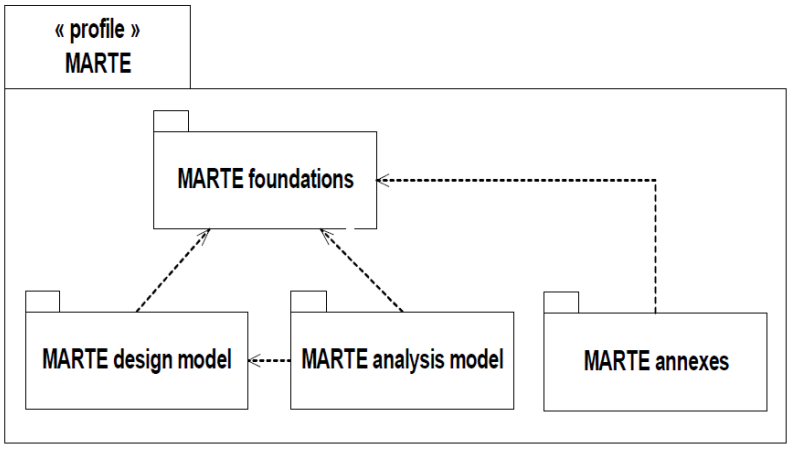

Figure 1.3 – Overall architecture of MARTE

The profile is structured around two concerns, one to model the features of real-time and embedded (RTE) systems and the other to annotate application models to support analysis of system properties. Figure 1.3 presents the structure of this profile. The MARTE design model package provides a domain-specific language for modeling phenomena specific

to RTE systems. The MARTE analysis model package is a viewpoint-based sub-profile suitable for model analysis. These two major parts share common concerns with describing time and the use of concurrent resources, which are contained in the shared package called MARTE foundations. A fourth package contains the annexes profiles defined in MARTE which define a complementary cross-cutting modeling constructs, as well as a predefined model libraries that may be used by modelers to denote their real-time and embedded applications [Kordon et al., 2013, Mallet, 2015].

— **MARTE Foundations.** The foundation package is divided into five chapters:

- CoreElements define configurations and modes, which are key parameters for analysis.
- NonFunctionalProperties sub-profile allows to describe the quantitative as well as the qualitative aspects of properties.
- Time sub-profile defines the concept of time, which is of top priority for the embedded real-time systems.
- Generic Resource Modeling sub-profile provides annotations to satisfies platform modeling.
- Allocation gives a compatible way to make this deployment.

— **MARTE design model.** The design model package has four chapters:

- High level application modeling provides a set of extensions to the UML that allow to annotate the elements of a model
- Generic component modeling reviews the composite structures of UML and extends one of its sub-models in order to better address the domain-specific requirements in terms of component-based modeling.
- Software Resource Modeling (SRM), and Hardware Resource Modeling (HRM): these two sub-profiles specialize the Generic Resource Modeling sub-profile in order to provide a basis for the modeling of software and hardware platforms.

— **MARTE analysis model.** The analysis model package also has a sub-profile that defines generic elements to perform model-driven analysis on real-time and embedded systems. This generic sub-profile is specialized to address schedulability analysis and performance analysis.

— **MARTE annexes.** The MARTE annexes integrate the set of extensions defined in the norm, the set of the libraries and the models. They propose an advanced textual

editor of the value description language (VSL) and encompass two other sub-profiles: the repetitive structure modeling (RSM) sub-profile and MARTE library sub-profile. The RSM sub-profile allows to describe systems-on-chip of the massively parallel computing type, such as systems that implement image processing algorithms. The MARTE library sub-profile defines a model library that contains a set of primitive types and structured types in a normative annex.

2.3.2.2 MARTE usage

MARTE supports different domains modeling and/or different analysis techniques while its time model is rich and it combines physical and logical clocks. However, MARTE is not expected to be used as a whole in a single specification. It is expected to be the base of several complementary methodologies that cover different aspects of a system. This chapter covers one aspect and proposes a partial usage to capture important features. It is not intended to offer a comprehensive cover of all aspects. Several research works have compared the MARTE profile with some popular existing profiles and standards such as AADL, for example the [Faugere et al., 2007] work. Designers can use MARTE to model their AADL applications at earlier design stages. Similarly, models can be conceived for different views related to time properties, performance and scheduling. Once the applications have been developed, designers can take advantage of existing AADL validation and verification techniques and tools. These validation/verification aspects would come as a compliment to current MARTE aspects. MARTE has become the preferred defacto industry standard for the modeling of RTE systems while it shares common concepts with ADLs such as AADL, other standards and UML profiles.

2.3.3 Combining architecture modeling languages

2.3.3.1 Combining SysML with AADL

SysML and AADL are two standardized modeling languages specified for designing system architectures, but none of them provide the full range required to deal effectively with a specific kind of system architecture. [Behjati et al., 2011] propose an approach to combine these two languages, since they are both widely used in industry with adequate tool support. In fact, as shown in Figure 1.4, SysML and AADL are mutually complementary. SysML supports requirements engineering, traceability, and precise modeling of diverse

physical phenomena. On the other hand, AADL is oriented to model real-time embedded systems. It provides a software to hardware bindings in such systems allowing analyses of different system properties such as performance, timing, etc... The contribution behind this approach is then to extend SysML using the UML profile extension mechanism to cover all AADL concepts. This profile is called **Extended SysML for Architecture Analysis Modeling (ExSAM)**.

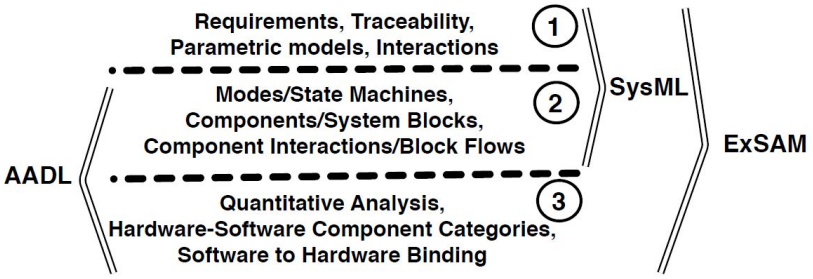

Figure 1.4 – The relationship between SysML, AADL, and the combined profile ExSAM. [Behjati et al., 2011]

Figure 1.4, level 2 shows the overlapping concepts between these two languages. However, they have different meanings, usages or design rationales. That's why this profile should resolve inconsistencies or augment either side of the mappings with constraints to account for semantic differences. The main idea of this profile is to satisfy SysML limitations in order to address important AADL concepts.

AADL provides two mechanisms for declaring components: using the component type construct or using the component implementation construct. SysML blocks are used to model both AADL component types and AADL component implementations using two newly defined stereotypes *«ComponentType»* and *«ComponentImpl»*. In fact, using blocks for modeling components allows to easily use other SysML constructs (e.g. parts and ports) to model AADL constructs (e.g. subcomponents and ports) associated with a component.

AADL software component categories and AADL hardware component categories are-defined as stereotypes with a set of attributes representing the properties of the corresponding component category.

The applicability and usefulness of ExSAM were investigated through two case studies. One benchmark case study showed that ExSAM can fully cover all AADL aspects and one large-scale industrial case study, showed that ExSAM was sufficient to satisfy the modeling needs of industrial partners, while AADL and SysML alone were not.

2.3.3.2 Combining SysML with MARTE

The lack of a common design language between different disciplines hampers the reasoning about system properties. Designers of one part of the system may make wrong assumptions concerning some other parts that result of the increasing development costs due to long feedback cycles. [Espinoza et al., 2009] investigate the possibility of combining these two languages, SysML and MARTE, since they are both widely used in industry. SysML [OMG, 2007] provides constructs to specify traceable requirements, structure and behavior of system blocks, as well as a parametric formalism to specify equation-based analytical models. MARTE (modeling and Analysis Real-Time and Embedded systems) [OMG, 2008] deals with time- and resource-constrained aspects, and includes a detailed taxonomy of hardware and software patterns along with their non-functional attributes to enable state of the art quantitative analyses. The purpose of [Espinoza et al., 2009] paper is to identify some scenarios in which the usage of the combination of the two profiles is of relevant added value in the embedded systems domain. SysML and MARTE consider characteristics of the embedded systems domain at different abstraction levels, architectural styles, and particularly for specific purposes or application areas.

Modeling capabilities of both SysML and MARTE are rich enough for a wide range of design approaches. In particular, SysML does not define any specific viewpoint, but it provides a means to specify how views are built, and to relate any user-specific view to a given viewpoint. Although MARTE does not provide any concrete model element to define viewpoints, it has an implicit conception of viewpoints rooted in its design rationale. Indeed, some of the MARTE constructs have been designed to define domain-specific viewpoints. Therefore, SysML and MARTE can be used in a complementary way. While SysML provides means to create viewpoints in a general way, MARTE provides particular viewpoints. As an example for Requirements Management/Traceability, SysML requirements diagrams explicitly show the various kinds of relationships between different requirements. On the other hand, MARTE offers key features to specify non-functional requirements in general and timing requirements in particular.

Integration strategies for combining the SysML and MARTE profiles were presented in [Espinoza et al., 2009] work. Both provide essential ingredients to model embedded systems. The intent of this paper is to offer a better understanding of their conceptual domains, and to help in using both profiles in a single model by avoiding semantic and syntactical mismatches.

2.4 EVENT-B Formal Method

In software engineering, formal methods [Wing, 1990, Rodhe and Karresand, 2015] are techniques that allow rigorous mathematical logic reasoning over computer programs or electronic devices to demonstrate their validity with respect to a certain specification. Formal methods allow a complete check of the entire system states and the properties that can be proved in the system are valid for all possible inputs. When formal methods can not be used throughout the development process (due to system complexity, lack of tools or other reasons), they can still be used on parts of the system.

EVENT-B is a formal method introduced by Jean-Raymond Abrial [Abrial, 2010] specialized in modeling dynamic systems, which has been used in safety-critical systems such as the Paris Metro line. As an evolution of the B-Method (called Classical B) [Abrial and Hoare, 1996], it specifies discrete systems based on mathematical notations, predicate logic and set theory. It is used to model systems, which could contain different types of components: software, hardware or humans, and to define the interactions between these components. An EVENT-B specification is composed of a set of elements of two kinds: Machine and Context. The Machine represents the dynamic part of the model and it regroups the behavioral properties of the system whereas the Context contains the static part of the model. A model can contain machines only, or contexts only, or both.

CONTEXT	MACHINE
<identifiant_contexte>	<identifiant_machine>
EXTENDS	REFINES
<liste_identifiant_contextes>	<identifiant_machine>
SETS	SEES
<liste_identifiant_ensembles>	<liste_identifiant_contextes>
CONSTANTS	VARIABLES
<liste_identifiant_constantes>	<liste_identifiant_variables>
AXIOMS	INVARIANTS
<label>: <prédicat>	<label>: <prédicat>
...	...
THEOREMS	THEOREMS
<label>: <prédicat>	<label>: <prédicat>
...	...
END	VARIANT
	<variant>
	EVENTS
	<liste_événements>
	END

Figure 1.5 – EVENT-B components structure

2.4.1 Machine

A machine is identified by a name and contains various clauses organized as follows:

— REFINES: a machine could be refined by another machine, this refinement is de-

fined in the clause REFINES and it is used to gradually introduce the details and complexity into a model.

— SEES: a machine can see one or several contexts thus by adding the name of the context in the clause SEES.

— VARIABLES: clause VARIABLES represents the state variable of the model. The changing of the variable values reflects the state in which the system is located.

— INVARIANTS: clause INVARIANTS lists the various predicates, which the variables must obey, at least, the typing of variables declared in the clause VARIABLES.

— THEOREMS: clause THEOREMS lists the various theorems, which have to be proved within the machine.

— VARIANT: The VARIANT clause appears in a machine containing some convergent events.

— EVENTS: clause EVENTS regroups events that specify the evolution of the state variables defined in the specification of the system. All variables should be initialized in the machine, a particular event called INITIALISATION is defined for this purpose.

2.4.2 Context

A context has an identifier, which must be distinct from all other component (machine or context) names within the same model. It contains various clauses organized as follows:

— EXTENDS: a context can extend optionally one or several other contexts by adding its name in the clause EXTENDS. This means that the present context can use all sets and constants of the extended contexts and of their extended contexts.

— SETS: clause SETS describes a set of abstract and enumerated types, which are the basic types of the specification.

— CONSTANTS: clause CONSTANTS represents the various constants introduced in the context.

— AXIOMS: clause AXIOMS contains all the properties of the constants and their types.

— THEOREMS: lists the various theorems, which have to be proved within the context.

A context could be seen by a machine in order to use its sets and constants to type the machine variables.

EVENT-B is based on two main mechanisms to master the complexity of a system which are refinement and model decomposition.

2.4.3 EVENT-B **refinement**

EVENT-B refinement is a process that allows to gradually introduce the different parts that constitute the system starting from an abstract model to a more concrete one. At each refinement level, system details are gradually added into the concrete model which must preserve the functionality and properties of the more abstract models. A machine refinement consists in adding new variables and/or replacing existing variables with new ones. A link between the abstract machine variables and the refinement machine variables is explicitly defined by gluing invariants. Events can be refined and new ones can be introduced. The refinement of an event has to verify that the guard of the refined event should be stronger than the guard of the abstract one and the effect of the refined action should be stronger than the effect of the abstract one. A context can also be extended with other contexts by adding new modeling elements (sets, constants and axioms).

The EVENT-B method is a state-based specification method. EVENT-B refinement ensures that specified state variables of the concrete specification are consistent with specified state variables in the abstract specification. In fact, event based specification refinement is expressed in a sequence of events accepted by the specification, while state-based specifications are expressed in terms of the effect of events on state variables. Here, two properties are verified : consistency and accessibility. The refinement consistency is verified if a sequence of events is accepted by the concrete specification and has an effect on state variables, then a corresponding sequence of events is accepted by the abstract specification and the effect of this sequence on variables states matches. The refinement accessibility consists of if an abstract event is allowed in a state, a corresponding concrete event must be allowed in a corresponding state.

2.4.4 EVENT-B **model decomposition**

EVENT-B model decomposition is a powerful mechanism to scale the complexity of the design of large and complex systems. An EVENT-B model can be decomposed into several simple sub-components which can be refined separately and more comfortably than

the whole. Many approaches allow to decompose an EVENT-B model, particularly, the shared-variable decomposition [Abrial, 2009] and the shared-event decomposition [Butler, 2009b]. Shared-variable decomposition is suitable for shared memory parallel systems, whereas shared-event decomposition is more suitable for distributed system development. The shared-variable decomposition approach consists in distributing the events of a model over the selected sub-components. It allows the introduction of shared variables and external events. These ensure that the behavior of shared variables is preserved in all sub-components. After that, further refinements then concentrate on how each sub-component processes shared state variables. The shared-event decomposition is a set of events that are synchronized and shared by sub-components. This approach defines a partial version of a global event in each sub-machine, when the variables of a global event are distributed between separate sub-machines. This is to simulate the action of the global event on the considered variables. The recomposition of the refined sub-components gives rise to a component which should refine the initial abstract component.

2.4.5 EVENT-B **Proof obligation**

The proof of the correctness of the EVENT-B models is one of the most important aspects of EVENT-B method. To ensure this correctness, a set of proof obligations (denoted PO) must be discharged. These proof obligations concern different aspects of the model such as the verification of the invariants properties of an EVENT-B machine or the proof of the correctness of a refinement. There is a different types of proof obligations.

— Invariant preservation: An invariant is a property that the system must always preserve while its evolution. For this, each triggering of any event must always verify this property preservation.

— Feasibility: Each event in an EVENT-B machine must always be feasible. Feasibility requires that a new value actually exists for each event having a substitution. This means, for a variable value before the triggering of an event, it should satisfy the guard of the event and satisfies the system invariant.

— Theorems: Recall that theorems in EVENT-B are formulas that can be useful for rewriting the invariant in a specific form or proving lemmas. The proof obligation consists in proving that these formulas are deduced from the invariant of the machine as well as the set of predicates in the AXIOMS and THEOREMS clauses of the context.

— Abstract event refinement: This proof obligation guarantees that the abstract event is correctly refined by the concrete event.

— Guard strengthening: The purpose of this proof obligation is to make sure that the concrete guards in a concrete event are stronger than the abstract ones in the abstract event. This ensures that when a concrete event is enabled, so is the corresponding abstract one.

— Well-definedness: This proof obligation rule ensures that a potentially ill-defined axiom, theorem, invariant, guard, action, variant, or witness is indeed well defined.

More details about proof obligations are found in [Abrial, 2010].

2.4.6 Tools Environments for Event-B

The Event-B modeling language is supported by the AtelierB [AtelierB, 1990] environment, by the Rodin platform [Abrial et al., 2010] and by the ProB model checker and animator [ProB, 2003]. These environments provide facilities for editing machines, refinements, contexts and projects, for generating proof obligations corresponding to a given property, for proving proof obligations in an automatic or/and interactive process and for animating models. The internal prover is shared by AtelierB and Rodin and there are hints generated by the prover interface for helping the interactive proofs. However, the refinement process of machines should be progressive when adding new elements to a given current model and the goal is to distribute the complexity of proofs through the proof-based refinement. These tools are based on logical and semantical concepts of Event-B models (machines, contexts, refinement).

3 State of the art

This section gives an overview of state-of-the-art contributions for designing system models and their alignment with stakeholders needs. Particularly, we are interested in (i) design models and their formal specification, (ii) requirements and architecture alignment. The first part presents works about architecture modeling and their translation into formal specification. These works present approaches using UML and its variants (SysML, MARTE, ...) which are aligned with our proposed methodology and the translation of graphical models into formal specifications, precisely Event-B, which is recommended for

critical systems. The second part presents research works dealing with alignment and traceability issue between design models and requirements. These works define relationships and traces established between UML, SysML, MARTE, ... design models and requirements presented as models or textual specifications.

3.1 Design Model and Formal Specification

3.1.1 From Design Model to Formal Specification

Formal methods are specifically used for complex systems and more precisely for safety critical systems because they need to be robust and reliable. Formalizing and abstracting functionalities through specifications, in a "mathematical" way, make the verification process easier to be applied in order to rigorously check the correctness of these systems. The use of formal methods is thus recommended and a recent study of technological efforts concerning the use of formal methods in railways performed in 2017 [ASTRAIL, 2017] claims that “this analysis has shown a large variety of formal tools used, with a dominance of the tools associated to the B family”. This is reaffirmed in [Bonvoisin, 2016]. However, using formal methods leads to complex models which may be difficult to read, understand and modify.

Graphical languages are used for visualizing, specifying, constructing and documenting software systems in a simpler way. [ASTRAIL, 2017] also concludes by claiming: “This analysis has shown a dominance of the UML modeling language for high-level representation of system models ...". However, the semantics of UML are given in natural language, which does not allow formal and rigorous reasoning necessary for critical systems for which safety and security are major concerns. So the complementarity between these two techniques and how to link them one to the other are the objective of several research works, methods and tools.

3.1.1.1 UML-B

UML provides graphical models that facilitate communication of ideas and system process understanding but lacks of formal semantics. The B method [Abrial and Hoare, 1996] allows rigorous formal verification and animation but requires significant effort in training to overcome the mathematical barrier that many practitioners perceive. To cope with these problems, [Snook and Butler, 2006] propose a derivation of the B notations as

an action and constraint language for UML via the UML profiles mechanism and define the semantics of UML entities via a translation into B. UML-B profile provides specializations of UML entities to support model refinement. The result is a formally precise variant of UML that can be used for refinement based object-oriented behavioral modeling. The currently developed version of this profile uses the Event-B method and is integrated into Rodin [Abrial et al., 2010].

This approach is based on UML class and state-machine diagrams. Class diagrams allow to specify structures of the system. Associations are translated into relationships between the sets and class methods are translated into events which allows to manage these sets. State-machine diagrams define the system components behaviors. In Event-B, state-machine transitions are translated into events. state-machine states can be translated into constants, and a state variable which takes the value of the current state.

The integration of the tool supporting this approach in Rodin includes the UML drawing tools and a translator to generate Event-B models. This tool allows to automatically generate the corresponding Event-B models, every time a drawing is saved. The Event-B verification tools (syntax checker and prover) then run automatically.

3.1.1.2 The B4MSecure Platform

The B4MSecure platform [Idani and Ledru, 2015] is a model-based approach which allows a graphical modeling and formal reasoning on both functional and security models of a system. This approach is based on UML for the functional description and SecureUML [Lodderstedt et al., 2002] for the access control rules. It generates B formal specification which allows rigorous verification of the system functional and security models. These models are firstly validated separately, and then integrated in order to verify their interactions and complementarity. The B4MSecure platform allows, on the one hand, the separation of concerns, and on the other hand, the identification of links between functional and security models. In fact, it allows a graphical modeling using UML class diagram customized with access control policy concepts introduced by a UML profile for RBAC (Role Based Access Control) inspired from SecureUML. Also, it assures the translation of both models into B specifications in order to formally verify them.

This platform produces one B model, from the functional UML class diagram, gathering its structural properties and all basic operations (constructors, destructors, getters and setters) and a second B model is produced from the UML class diagram customization with

RBAC concepts to represent the security applied on functional models. Also, additional invariants and user-defined operations can be added manually to take benefit of a proof tool like AtelierB in order to validate the consistency of the functional specification. The platform provides an annotation mechanism which allows the integration of B invariants and specification of operations in the graphical model. This functionality is useful to avoid inconsistent evolutions of the graphical and formal methods.

3.1.1.3 The CHESS toolset

[Cicchetti et al., 2012] and [Mazzini et al., 2016] present the CHESS toolset, a tooled MDE approach for cross-domain modeling of industrial complex systems. These works are based on an extension of UML, SysML and MARTE modeling languages which allow the specification of well-defined design views, each of them addresses a particular aspect of the problem. The CHESS toolset allows code generation toward multiple target languages and property description, verification, preservation and dependability using a dedicated UML profile. This methodology supports the separation of concerns principle, strictly separating the functional aspects of a component from the non-functional ones. In fact, this approach relies on SysML for the modeling of requirements and for the system level design, on UML for modeling software aspects of the system, and on MARTE for describing real-time aspects, staying as close as possible to the standard modeling languages. In particular a profile has been defined on top of UML to model failures definition and their intra/inter-components propagation, while SysML has been extended to offer support for contract-based design. MARTE has been used and extended to be able to model real-time properties such as schedulability, end-to-end response time and different scheduling algorithms for multicore deployments.

The CHESS methodology enables early verification, as possible inconsistencies and integration issues will be raised at the earliest stages of the process. It also supports system-software co-engineering as a seamless process, by keeping traceability between system level entities and requirements on one side and the corresponding software and hardware level entities on the other side.

3.1.1.4 Coupling of formal methods for industrial systems specification

[Fayolle, 2017] approach proposes to couple a formal graphical notation called Algebraic State Transition Diagrams (ASTD) with an Event-B specification in order to provide a

better representation of the software behavior. The behavior is captured by ASTDs, based on automata and process algebra operators, while the data model is described by means of an EVENT-B specification. A set of properties can be checked. For instance: if a transition can be executed according to the ASTD specification, then the corresponding event in the EVENT-B specification can be executed, or an EVENT-B machine corresponding to an ASTD must be able to execute all the sequences of events corresponding to the sequences of transitions of the ASTD.

3.1.1.5 SYSML to formal specification

[Mentré, 2016] present an approach to transform SYSML diagrams into a B specification. This approach proposes the use a graphical design of the architecture of a B software using SYSML model and automatically transforms this model into a B specification. This work focuses on the architecture of the system and its contained software. Therefore, it considers only Block Description Diagram (BDD) and Internal Block Diagram (IBD). This approach prototype was built upon the Eclipse environment, with Papyrus [Papyrus, 2008] SYSML editor and Acceleo [Musset et al., 2006] model transformation language.

[Salunkhe et al., 2021] propose a transformation using Triple Graph Grammars (TGGs) [Schürr, 1994] from SYSML to EVENT-B. The aim is to identify the subset of the EVENT-B language and SYSML language, which is necessary and appropriate for the transformation, then search for the semantic similarities between both constructs and finally define a transformation from SYSML to EVENT-B using MDE techniques. This work is an advancement of the MBSE approach explained in [Berglehner et al., 2019]. In [Berglehner et al., 2019], the SYSML state-machines are transformed to equivalent UML-B state-machines using UML-B plug-in [Snook and Butler, 2006]. Later, UML-B state-machines are used to generate EVENT-B code, and then the safety requirements are verified. The overall approach is time-consuming and increases the overall life cycle cost. From these challenges the work of [Salunkhe et al., 2021] is motivated. In fact, it proposes a methodology and tool-chain to automate the transformation of SYSML specification models into EVENT-B models. Also, the traceability behind this transformation should be maintained between informal requirements and the modeled system, specifically for the safety properties. The main objective of this approach is to verify SYSML models against such safety requirements using formal methods with some tool support and reduce the efforts involved in the manual transformation of a SYSML semi-formal model to an EVENT-B model. The implemented

prototype is limited to the simple and most relevant concepts of SysML state-machines. The transformation rules are defined using eMoflon-IBeX [Weidmann et al., 2019].

The work of [Poorhadi et al., 2022] proposes an integrated approach to combine SysML graphical modeling with Event-B for formal specification and verification. The aim of this approach is to reason about safety and security interactions at system modeling stage. It also allows to visualize and formalize the analysis of the impact of security attacks on system safety. The SysML subset used in this work contains block definition diagrams, state-machine diagrams and sequence diagrams. The translation of these SysML elements into Event-B aims at achieving two goals. Firstly, it checks the correctness of the SysML diagrams and verify their consistency. Secondly, it analyzes the impact of cyber-attacks on system safety. Safety properties are defined as invariants in the final model. Proofs should be discharged without introducing any further assumptions.

3.1.1.6 Discussion

The UML-B and B4MSecure approaches propose a derivation of formal specification from graphical modeling. These approaches are based on UML class and state-machine diagrams and use UML profiles mechanisms to conduct the transformation into B notations. However, these approaches reason about graphical design formally, but they require a basic designer knowledge about the B notations to use correctly the UML-B profile and to add safety invariants.

The CHESS toolset allows code generation toward multiple target languages and property description, verification, preservation and dependability through a dedicated UML profile. However, this approach do not allow system/sub-systems model decomposition and refinement mechanisms which are particularly well suited to HLA modeling.

[Fayolle, 2017] proposes to link ASTDs with Event-B specifications in order to provide a better representation of system behavior. However, the use of SysML rather than other graphical modeling language is advantageous since SysML offers a set of concepts more relevant to model systems. Moreover SysML is recommended by the AFT project partners.

The SysML to formal specification approaches presented in 3.1.1.5 provide a model transformation into formal specifications (B and Event-B). Their aim is to reason about safety and security interactions at system modeling stage. However these approaches lack of system/sub-systems decomposition and refinement mechanisms which are particularly recommended to HLA modeling. Furthermore they do not allow to demonstrate require-

ments/HLA alignments and how HLA elements participate in satisfying stakeholders needs.

3.1.2 Refinement of design models

Refinement of design models can be used in a top down way in order to reduce system complexity by gradually enriching abstract models with more details, while ensuring that the detailed description preserves the original abstraction. Models refinement can also be used in a bottom up way, to reverse engineer existing systems in order to enrich the abstractions encoded in the concrete models.

Event-B refinement is used to relate system models at different abstraction levels. These abstraction-refinement concepts can also be applied in UML models [Said et al., 2009]. The notion of refined classes and inherited attributes in UML-B corresponds to the variables refinement in Event-B. In UML-B refinement, a machine that refines a more abstract machine may contain refined classes where each refined class refines a class of its abstract machine. A refined class can inherit attributes of its abstract class and can drop some of the attributes of its abstract class. A refined class can introduce new attributes.

State-machines are refined by building abstract states with nested sub-states. In UML-B refinement, a machine can contain refined state-machines and refined states. The structure of a refined state-machine is an elaboration of the structure of its abstraction in two possible ways: each transition is replaced by one or more transitions and an abstract state can be refined by a nested state-machine. These nested state-machines are modeled in state-machine diagrams different from their parent state-machine diagrams. In a nested state-machine, a transition with an initial source state contains at most one incoming transition to the super-state and a transition with a final target state contains at most one outgoing transition from the super-state.

[Fayolle, 2017] uses two refinement mechanisms. The first one is to refine an ASTD specification and the other one is to refine data specification in Event-B. ASTD refinement [Frappier et al., 2014] is defined as follows: a concrete ASTD must preserve the traces accepted by the refined ASTD. It allows the addition and removal of states and transitions. The Event-B refinement in this approach is the classical one.

Papers [Lima et al., 2017] and [Miyazawa and Cavalcanti, 2014] propose a definition of guidelines of usage for construction of meaningful SysML models and a semantics for SysML models. This work focuses on a set of SysML diagrams which are block definition, internal block, state machine, activity, and sequence diagrams and refinement-based

analysis and verification is supported by providing a semantics for these SysML elements. Using these semantics, notions of refinement of complete SysML models are defined and can be used to support refinement laws that support the transformation of diagrammatic models.

Discussion Event-B refinement is an important mechanism to gradually introduce the different parts that constitute a system starting from an abstract model to a more concrete one. The refinement mechanisms allow to enrich models with more details but it does not allow a decomposition between different models describing systems components. That is why, a decomposition mechanism between a system and its corresponding sub-systems is required to simulate the behavior of each sub-system independently of its parent system and to manage the interplay of the sub-systems behaviors which satisfy the parent system tasks.

3.1.3 Event-B Model Decomposition

Model decomposition is a powerful tool to scale the design of large and complex systems. The main idea of the decomposition is to cut a model *M* into sub-models *M1, ..., Mn*, which can be refined separately and more comfortably than the whole. Many approaches have been proposed to deal with the Event-B decomposition issue: generic instantiation [Abrial and Hallerstede, 2007], the shared variable decomposition [Abrial, 2009], the shared event decomposition [Butler, 2009a], the modularization [Hoang et al., 2011], etc.

3.1.3.1 Shared Variables Decomposition

Abrial et al. [Abrial and Hallerstede, 2007, Abrial, 2009] present decomposition as a part of the state information (variables) that is shared between sub-components. Further refinements then concentrate on how each component processes shared state information. This approach proposes to handle the variables shared between several events, using external variables and events.

3.1.3.2 Shared Events Decomposition

[Butler, 2009a] present a decomposition approach, using shared events. A shared-event decomposition is a set of events that are synchronised and shared by sub-components

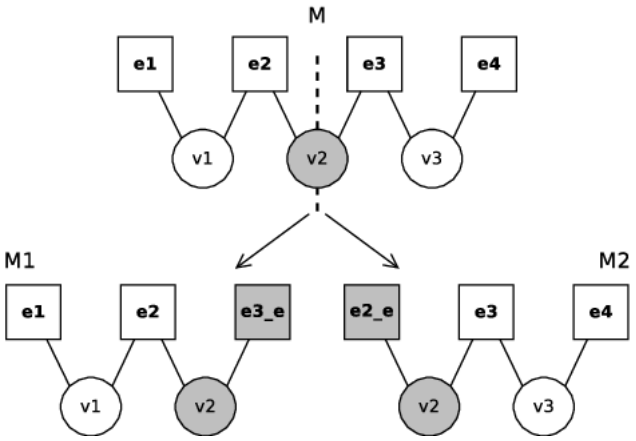

Figure 1.6 – Shared Variables Decomposition

[Hoang et al., 2011]. This approach consists of defining a partial version of a global event in each sub-machine, when the variables of a global event are distributed between separate sub-machines, to simulate the action of the global event on the considered variables.

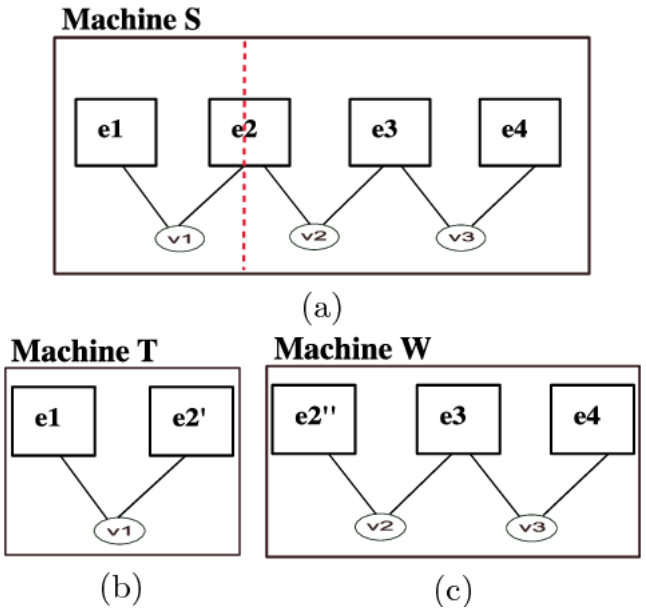

Figure 1.7 – Shared Events Decomposition

3.1.3.3 SysML/KAOS Event-B model decomposition

SysML/KAOS goal models allow the capture of assignments of requirements to agents responsible for their achievement. Each agent is associated with a sub-system. [Fotso et al., 2018c] propose the use of a formal decomposition strategy shown in Figure 1.8, applied at the most concrete level of the Event-B specification of the high-level system (parent

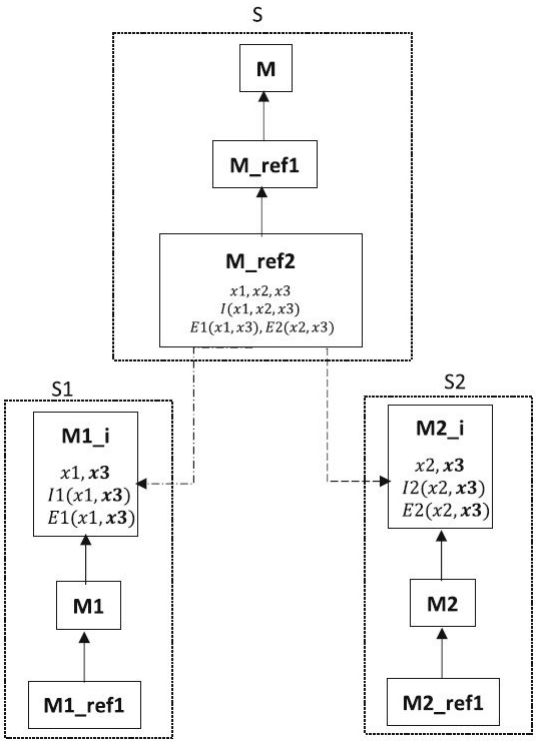

Figure 1.8 – [Fotso et al., 2018c] EVENT-B model decomposition

component), to build sub-systems interfaces. The definition of an EVENT-B components called *interfaces* will bridge the gap between system and sub-systems specifications. This interface defines events that correspond to the goals that the system assigns to the sub-systems. It also defines variables involved in these events and their constraints. Invariant predicates that define properties involved in sub-components also should be assigned to the corresponding sub-component. To ensure that the sub-system specification conforms to the interface specification, the most abstract level of the formal specification of a sub-system is defined as a refinement of the sub-system interface. For an interface corresponding to the agent, internal events of the interface are the correspondences goals assigned to the agent. The variables of the interface are the ones involved in its internal events. If an interface variable appears in another interface, then it is an external variable. Otherwise, it is an internal variable. External events are defined in the interface, to emulate how external variables are handled in other interfaces. Each external event is an abstraction of an internal event defined in another interface. In this approach, [Fotso et al., 2018c] advocate also a set of conditions that are necessary and sufficient to decompose the invariants involving variables assigned to different interfaces.

This approach allows to avoid the difficulties lying in the definition of external events

such as redundancy of the same behavior associated with an external variable in each interface where the external variable appears and the partitioning of guards and actions of an event to consider only the variables of the interface where the external event must be defined. This is achieved by defining a link between sub-systems interfaces and the most concrete component of the high-level system specification.

3.1.3.4 Discussion

These approaches allow to reduce the complexity of the final level of modeling critical systems, which can lead to complex and voluminous models, using model decomposition techniques. Although, the real difficulty lies in the determination of the refinement level from which to introduce the decomposition. Regarding the shared event approach, it may be difficult, once the distribution of variables has been done, to separate the guards and actions of events in order to construct the partial events (a variable cannot appear in two different sub-machines). Also, not all actions are accepted to be decomposed and variables partitioning is not always possible. Regarding invariants, the user selects which invariant predicate should be assigned to which sub-component. However, for the shared variable decomposition, shared variables and external events must be present in the resulting sub-components and cannot be refined when refining these sub-components. Adding to that, these approaches are about only formal specifications and their decomposition with no easy-way to give a simpler presentation for no formal specifications stakeholders such as system graphical modeling and decomposition.

The approach of [Fotso et al., 2018c] focuses on ensuring that a requirement assigned to a sub-system is well achieved by this sub-system. The approach uses a formal model decomposition strategy and proof obligations to guarantee that sub-system goals are consistent and meet system requirements expressed in SysML/KAOS models that are translated into Event-B specifications. Although, this approach provides formal specifications and graphical model decomposition but it is limited only to RE step and does not provide a complete view about the software design process and more precisely about its architecture modeling.

3.2 Requirements and Architecture Alignment

Requirements traceability is defined as the ability to follow the life of a requirement in both backward and forward directions. A RE process should be defined, which allows

to identify, control and monitor requirements and its changes in each project step. This requirements tracing is based on relationships between requirements themselves or between requirements and design artifacts. Several research works have proposed methodologies to build these kinds of relationships.

3.2.1 SysML relationships between requirements and SysML elements

SysML [OMG, 2007] specifies a set of requirements relationships which allow to relate requirements to other requirements or to other design model elements.

- «Copy» relationship is a dependency between a supplier requirement and a client requirement that specifies that the text of the client requirement is a read-only copy of the text of the supplier requirement.
- «DeriveReqt» relationship is a dependency between two requirements in which a client requirement can be derived from the supplier requirement.
- «Satisfy» relationship is a dependency between a requirement and a model element that fulfills the requirement.
- «Refine» relationship can be used to describe how a model element or set of elements can be used to further refine a requirement. For example, a use case or activity diagram may be used to refine a text-based functional requirement.
- «Verify» relationship. A Verify relationship is a dependency between a requirement and a test case or other model element that can determine whether a system fulfills the requirement.

3.2.2 The MeMVaTEx methodology: from requirements to models in automotive application design

Authors in [Albinet et al., 2008] present a model-based methodology named **MeMVaTex** for requirements expression, traceability and verification. The methodology relies on the Electronic Architecture & Software Tools – Architecture Description Language (EAST-ADL2) framework and two of the UML profiles: MARTE for real-time embedded systems and SysML for system requirements modeling. The methodology defines the different models used at each abstraction level of the process. The results are a requirement model and a solution model which relates to the requirements. From the EAST-ADL2 framework, a decomposition of the design process into abstraction levels is adopted. For each level,

requirement models and solution models are separately built. The real-time aspects and non functional constraints are modeled within the UML MARTE profile. V&V techniques can then be connected to these models to express the satisfaction of the requirements by the proposed solution. Traceability management mechanisms defined in this approach are used for relating requirements of the same abstraction level, requirements through successive abstraction levels and requirements to other elements from solution models or V&V means. Traceability links used in MeMVaTEx are those proposed by SysML, but they concern MeMVaTEx requirement and design elements from SysML. In a same EAST-ADL2 level, requirements can only have «DeriveReqt» dependency relationship between requirements for a requirement A (the client) refined into a requirement B (the supplier) and the requirement containment relationship for the decomposition of a parent requirement into several ones. For considering requirements of different levels, the previous relationships are used with the «copy» dependency relationship which is related to requirements that appear in a level and that are unchanged when considering the next EAST-ADL2 level. To relate requirements to other design elements, the «satisfy» dependency relationship is used between a requirement and a model element that fulfills this requirement. The «verify» relationship between a requirement and a test case that can determine whether a system fulfils the requirement.

3.2.3 KAOS: Connecting the goal model with other system views

The core of a KAOS goal model consists of an annotated refinement graph where potential conflicts may be indicated [Van Lamsweerde, 2009]. In addition to refinement and conflict links, the goal model shows interface links with other sub-models of the system model.

— The concern link presents an interface between the goal model and the KAOS object model. A goal concerns a conceptual object if its specification refers to this object. However, a conceptual item referenced in a goal/property specification has been determined to be an object. This object should be an entity, an association, an agent, or an event.

— The responsibility link presents an interface between the goal model and the KAOS responsibility model. It relies on agent capabilities which are defined in terms of ability to monitor or control object attributes and associations defined in the object model. A goal assigned to some agent must be realizable by this agent in view of its

capabilities.

— Operationalization link refers to the process of mapping leaf goals, under responsibility of single agents, to operations ensuring them. Each such operation is performed by the responsible agent under restricted conditions for satisfaction of its underlying goals.

— coverage link relies on making a goal underline a positive scenario. A specific instance of agent behaviors is captured through scenarios and general class behaviors are captured through state machines. A sequence diagram illustrating typical interaction sequences among agent instances is in general more easily elaborated by focusing on pairwise interactions one agent pair after the other. When the goal underlying the scenario is an achieve goal and the scenario is sequentially composed of cohesive parts, the end of each part is considered as a milestone for reaching the goal target.

3.2.4 UML, MARTE, SysML/Requirements Traceability

[Marques et al., 2014] present a model-driven requirement engineering approach for the embedded software domain. It is based on an integrated set of UML, MARTE and SysML standard notations in order to improve requirements specification and traceability. This approach named MDEReq supports modeling and management of functional and non-functional requirements also it gives for designers an effective control of requirement changes, and its impacts on other requirements or design artifacts in the whole development process. To represent requirements, SysML requirement diagram is used. It allows to represent functional and non-functional requirements. It allows also to model requirements derivation and requirements hierarchy relationships using «derive» and «composite» stereotypes. UML models are used to build a functional, structural and behavioral view of the system, using respectively, use case, class and sequence diagrams. From the MARTE Foundations, Time and Generic Resource Modeling (GRM) are used to indicate classes that represent the interactions with external devices and to represent a clock and delay aspects. From the MARTE design model, the high Level Application Modeling (HLAM) sub-package is used to indicate that a class represents a concurrent unit and to detail timing aspects into a sequence diagram.

These artifacts are related to the requirements using «satisfy» and «refine» SysML relationship. «refine» is used to indicate that a requirement is detailed by a use case, while «satisfy» identifies that an artifact must satisfy the associated requirement. A sequence

diagram is used to satisfy a requirement behavior. During the validation activity, test cases are defined and related to requirements through «verify» relationships. This approach proposes three different matrixes, which allow tracing requirements in different abstraction levels and in different design phases. During the elicitation activity, a traceability matrix is generated in which relationships between two requirements can be traced according to derive and composite relations. The matrix relating requirements to design artifacts is generated during the analysis and specification activity through «refine» and «satisfy» relationships. Finally, the tracing matrix built during the validation activity indicates which test cases are used to verify each requirement. When a requirement changes, related test cases should be checked.

3.2.5 Generation and validation of traces between requirements and architecture based on formal trace semantics

[Goknil et al., 2014] present an approach for automatic trace generation and validation between requirements and architecture. Requirements relations and architecture verification techniques are used. A trace meta-model is defined with commonly used trace types. The semantics of traces and requirements relations are used for generating and validating traces with a tool support. The tool provides a generation and validation of traces by using requirements relations and/or verification of architecture and generation and validation of requirements relations by using traces. The tool is based on model transformation in ATL and term-rewriting logic in Maude [Clavel et al., 2003].

The aim of this approach is to improve the literature observed practices by providing a degree of automation that allows faster trace generation and improves the precision of traces by validating them. Two types of traces between requirements and architecture which are «satisfies» and «allocatedTo».

— «satisfies» traces are established automatically based on architecture verification and reasoning over existing traces. It relates a set of architectural elements to a single requirement. The «satisfies» traces are established after verifying the requirement over the architectural model.

— «allocatedTo» traces are usually assigned manually by the software architect. They express the expectation of the software architect that a certain set of architectural elements is responsible for fulfilling a given requirement. Since the «allocatedTo» traces are manually assigned, they may be invalid and/or incomplete.

The basic mechanisms of automatic generation of satisfies traces, trace validation, and inference based on requirements relations can be combined in various scenarios that the software architect can follow.

1. Generating traces by using verification of architecture.
2. Validating traces by using verification of architecture.
3. Generating/validating traces by using requirements relations.
4. Generating/Validating requirements relations by using traces between requirement and architecture.

To resume, this approach improves the process of collecting traceability information. First, traces can be generated automatically by checking if a requirement is satisfied by the architecture. This makes the process of establishing traces faster and less error prone compared to manually assigning traces. Second, traces are validated by using verification techniques and constraints ensuring that requirements relations are reflected in the software architecture. This eliminates false positive traces and helps in identifying missed traces. As an additional result, the requirements model may be improved by detecting invalid requirements relations and discovering new relations. The process is generally semi-automatic and iterative since the software architect has to decide on the outcome of the supporting tools.

3.2.6 Traceability Between Restricted Natural Language Requirements and AADL Models

[Wang et al., 2019] propose an approach to bridge the gap between natural language requirements (NLRs) and AADL models. First, this approach proposes a requirement modeling method based on the restricted natural language, which is named as RM-RNL. The RM-RNL can eliminate the ambiguity of NLRs and barely change engineers habits of requirement specification. Second, it presents a method to automatically generate the initial AADL models from the RM-RNLs and to automatically establish traceability links between the elements of the RM-RNL and the generated AADL models. Third, the initial AADL models are refined through patterns to achieve the change of requirements and traceability links. This paper focuses on three traceability scenarios. The first traceability scenario (TS-1) is based on the RM-RNL, which automatically generates the AADL models and requirement traceability links through model transformations. TS-2 describes the requirement changes, that is, change the elements of the RM-RNL; therefore, the AADL

models and the traceability links should be regenerated. TS-3 describes the refinement of the AADL models. It should maintain the change of the requirements and the traceability links at the same time. To establish these links, requirement traceability information model called RAInterM is created. RAInterM defines a "TraceType" enumeration to specify links types, which are: «Generation automatically» which relates the component to a requirement through model transformation. «ImplementedBy» relates requirement to system fragments, implementation plans, code source, etc. «MappedTo» relates requirement to a particular attribute, operation, state, or value of the artifact. «Satisfy» which relates requirement to the component that fulfills it, «Refine» which relates a requirement to its refined requirement and finally «Verify» which relates requirements to test cases.

In this approach the process of automatically generate the requirement traceability links actually consists of three steps. First, Traceability links are established between the RM-RNL and the RAInterM model. Then, Traceability links are established between a RAInterM model and the AADL models. Finally, Traceability links are established between the RM-RNL and the AADL models through merging the generated traceability links in the former two steps.

3.2.7 Discussion

The proposed approaches provide solutions to assure requirements traceability vis-a-vis design model elements. However, not all of them can allow the formal verification of the requirements, the design and the traceability links. In fact, SysML, [Albinet et al., 2008], KAOS and [Marques et al., 2014] approaches support requirements and/or architectural design modeling. They provide a mechanism to align requirements with design elements but they lack of formal verification techniques to prove the consistency of the different models and also of the alignment links. These approaches only use test cases associated with requirements to verify the alignment. The work of [Goknil et al., 2014] presents an approach for automatic trace generation and validation between requirements and architecture. This approach is based on formal trace semantics, yet, it does not provide formal reasoning about architecture models and it provides only two types of generated links: automatically «satisfies» and manually «allocatedTo». These links do not give semantics about the manner of how this satisfaction is applied or the order of architecture elements execution which assures the requirement satisfaction. The work of [Wang et al., 2019] produces AADL design model directly from requirements models, then alignment links exist

intuitively between the different models and there is no difficulties to identify them. Also, this approach takes as input only textual specification to model requirements and it does not support any other kind of artifacts which specify requirements. Finally, this approach also does not provide formal reasoning about requirements and architecture models.

4 Synthesis

While the quality of a system is the main measure of its success, which depends on the extent to which it meets its requirements. Also, in large complex system design, requirements engineering experts may be different of HLA design experts then two different kinds of models will be designed for the same system with no correspondence. In our work, the choice is on SysML/KAOS to model requirements and formally verify them while SysML/KAOS provides strong semantic expressiveness, refinement and decomposition mechanisms well suited with Event-B. Then, the thesis focuses on HLA modeling with design model refinement and decomposition and its formal verification using Event-B. The choice for this step is on SysML and Event-B formal method because SysML is aligned with SysML/KAOS, offering a set of concepts more relevant to model complex systems and it is recommended by the AFT project partners. Event-B, on the other hand, is recommended for critical systems. To reveal SysML extensions with refinement and decomposition mechanisms, a purpose to combine SysML and the Event-B is stemming from the need to master the complexity of such systems that allow a step-by-step design and make proofs easier. Afterward, alignment links are established between SysML/KAOS requirements models and SysML HLA models to display a traceable requirements satisfaction by HLA elements. Finally, the thesis proposes a plugin that supports these works, implemented using available free software and frameworks (EMF, Papyrus, AtelierB, etc.).

5 Conclusion

In this chapter, we have presented related works about requirements modeling and its formal specification and HLA modeling with its refinement and decomposition which are important mechanisms to manage HLA complexity. We have presented works dealing with the formalization of the HLA models into formal specifications. Finally, we have presented works that propose solutions for requirements and architecture alignment and the formal verification of these alignment links.

A summary of the similarities and differences between the studied approaches is presented in table 1.1. Most of the presented works deal with graphical requirements and/or HLA modeling. However, we can conclude from the table that most of these works do not formally verify requirements and HLA models to prove their correctness and consistency. The works that allow requirements and HLA modeling at the same time define mechanisms to align requirements and architecture models in order to provide a traceability between them and demonstrate that the architecture corresponds to stakeholders needs. Although, these works, except [Goknil et al., 2014], do not give a formalization of these alignments.

Model refinement and decomposition are two main mechanisms to master the complexity of complex systems. The table shows that only UML-B, [Miyazawa and Cavalcanti, 2014, Lima et al., 2017] and [Wang et al., 2019], presents solutions to support model refinement. Refinement is a process that allows to gradually introduce the different parts that constitute complex systems and allows to define how sub-components interplay can satisfy parent system goals. Model decomposition is shown in the table as a mechanism presented in [Abrial, 2009, Butler, 2009a] and used in SysML/KAOS to decompose Event-B models into finer-grained models. Nevertheless, the presented approaches that support HLA graphical modeling do not provide a solution to decompose HLA into a system/sub-system hierarchy. Adding to that, HLA model decomposition into several sub-components allows a better management of large complex systems and more comfortably than the whole.

The work of [Goknil et al., 2014] is the most complete work in the table evaluation criteria. In fact, it allows requirements modeling and its formal specification, HLA graphical modeling and it assures requirements and architecture alignment and its formal specification. However, in this approach, there is no model decomposition applied on HLA models to give system/sub-system hierarchy. HLA models are not formally verified. This approach defines only two types of alignment links. One of them is specified manually. Its process is generally semi-automatic and iterative since software architect has to decide on the outcome of the supporting tools, so it is not possible to guarantee the consistency of the outcomings.

Based on these remarks and limitations, the work of our thesis aims to propose a complementary approach to existing approaches. Indeed, they do not give a complete support for modeling complex systems from requirements to HLA with a formal verification behind the different steps and a formal traceability which verifies the satisfaction of the stakeholders requirements by the HLA elements. For this, our motivations are to provide a holistic

Related works	Requirements modeling	Requirements formal specification	HLA graphical modeling	Design model refinement	Model decomposition	HLA formal specification	Requirements & Architecture alignment	Alignment formal specification
SysML	✓		✓				✓	
KAOS	✓						✓	
SysML/KAOS	✓	✓			✓			
UML-B			✓	✓		✓		
CHESS toolset	✓		✓					
[Salunkhe et al., 2021]			✓			✓		
[Poorhadi et al., 2022]			✓			✓		
[Lima et al., 2017] & [Miyazawa and Cavalcanti, 2014]			✓	✓				
[Abrial, 2009] & [Butler, 2009b]					✓	✓		
[Goknil et al., 2014]	✓	✓	✓				✓	✓
[Wang et al., 2019]	✓		✓	✓			✓	

Table 1.1 – Table summarizing the evaluation of related works

process which covers the conceptualization process of complex systems. More precisely, it aims to support requirements modeling with its formal verification, HLA modeling with its formal specification and the most principal necessity is to link HLA with requirements in order to guarantee the traceability between these two entities.

Chapter II

Contribution: A methodology for high-level architecture modeling aligned with requirements models

---

Contents

In this chapter, we propose a model-based methodology to prove the alignment of HLA models with stakeholders needs for the purpose of their satisfaction. This methodology allows, on one hand, to model requirements and HLA and formally verify them. On the other hand, it allows to formally verify traceability between these two parts. Results presented in this chapter are published in [Bougacha, 2020, Bougacha et al., 2022b, Bougacha et al., 2022a, Bon et al., 2023, Bougacha et al., 2023]

1 Methodology overview

Complex systems such as railway systems are composed of a large set of sub-systems. They generally are heterogeneous in that they integrate various kinds of components as mechanical, electronic, or software. The design of these systems requires the collaboration of domain experts and the use of a common language to communicate with each other to build a consistent model. Moreover their design depends on solutions that can address the interplay between sub-systems. Therefore, high-level architectures of complex systems should be represented as a layered hierarchy of sub-systems. Such a HLA must enable the specification of the main functional elements of a system, together with its interfaces and interactions while each sub-system can function independently from its participation in the main system. It constitutes a framework common to all the domain experts involved in the design of the system.

The development process of this kind of systems becomes critical since it could not respond to stakeholders needs due to its complexity and the consequence of a failure in such systems may be serious [Leveson, 2016]. The quality of such systems is the main measure of their success, that depends on the degree to which they fulfill their requirements. Requirements modeling is an important activity in the design process. The competitiveness of a system cannot be ensured unless its HLA is aligned with its requirements. Indeed,

alignment provides strong guarantees, and is key to a coherent governance and success of the system. Therefore, it is important to bring closer requirements and HLA modeling activities. In modern system development methods, analysts start the development process with an inception phase where they must acquire a deep knowledge of the system requirements. This phase is crucial since it prepares for requirements modeling, analysis and verification, which gives a global view on stakeholders needs.

Model-Driven Engineering (MDE) [Kent, 2002] is a software engineering approach which defines a generic framework to generate code using successive model transformations and allows to express separately each of the concerns of users, designers, architects, etc. In MDE, the crucial key point is the use of models as primary entities to process them automatically or half automatically. These models are abstract representations of a reality. MDE is increasingly used to develop complex systems. However it is difficult to establish traceability links between requirements and design models in the context of MDE. There is a substantial gap between requirements descriptions and designs, because transformation from requirements to design models is not included in MDE, which starts from an analysis model (or design model) and ends with deployed code.

MDE gives a simple manner to represent systems for a better understanding while it can provide graphical models. However, nowadays, the usefulness of formal verification and validation of system specifications is industrially demonstrated [Bonvoisin, 2016], especially for critical systems which require a high level of safety.

To cope with these issues, we propose a model-based methodology, summarized in Figure 2.1, composed of three phases: (1) SysML/KAOS Modeling, (2) SysML HLA Modeling and (3) Alignment.

A methodology provides a logical and systematic means of proceeding with the design process as well as a set of guidelines for decision-making [Zhu, 2005]. A methodology provides a sequence of activities, methods and often uses a set of notations or diagrams. A methodology is especially important for large complex projects where many designers are involved. Its use establishes a set of common communication channels for translating design to code and a set of common objectives. Methodology refers to the overarching strategy and rationale of a research project. It involves studying methods used in a field and theories or principles behind them, in order to develop an approach that matches required objectives. Methods are specific tools and procedures used to collect and analyze data.

The aim of the proposed methodology is to model stakeholders requirements, to represent HLA in a layered hierarchy of system/sub-systems relationships and finally to align HLA elements with requirements to guarantee traceability between them.

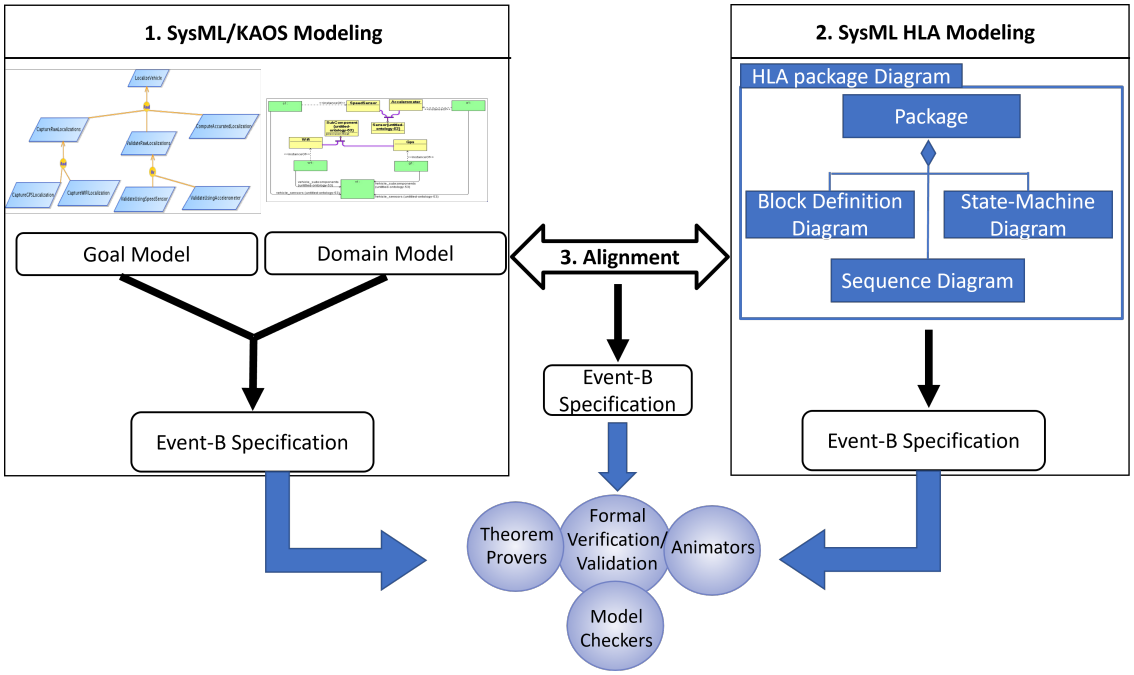

Figure 2.1 – Methodology overview

SysML/KAOS **Modeling [Laleau et al., 2010]:** Consists of a RE phase in which requirements are modeled using SysML/KAOS goal models enriched with domain models. Afterwards, these requirements models are formalized using Event-B in order to be formally verified. We have chosen the SysML/KAOS approach for the following reasons:

- ✓ It provides strong semantic expressiveness.
- ✓ SysML is well recommended by the AFT project partners
- ✓ Event-B is specialized in modeling systems, and has been used in safety-critical system applications such as the Paris Metro lines [Behm et al., 2003, Abrial, 2006]
- ✓ SysML/KAOS refinement and decomposition mechanisms are well suited with Event-B.

This phase is composed of two steps:

- First, a graphical modeling of system requirements as a hierarchy of goals is achieved using the SysML/KAOS modeling framework and refinement patterns. Then, these goal models are enriched with domain models that define the system structure and concepts used to define goals.

- Second, an EVENT-B formalization of these goal and domain models is carried out.

A formal verification step using EVENT-B provers, model checkers and animators is applied on the formalized specification to prove the correctness and consistency of system requirements.

SYSML **HLA Modeling:** Corresponds to the definition of the HLA of critical complex systems that must be validated by domain experts and its translation into EVENT-B formal specification to prove its correctness and consistency.

To conduct this phase, a two step process is performed:

- First, a graphical modeling of HLA in a system/sub-system layered hierarchy is presented using SYSML extension mechanisms that we have defined.
- Second, an automatic translation of HLA models into EVENT-B models is established.

The results presented in this phase have been published in [Bougacha et al., 2022b, Bougacha et al., 2022a].

Alignment: Aims to provide alignment links between SYSML/KAOS elements and HLA elements. These alignment links allow to guarantee traceability relationships between participating entities to assure that HLA elements satisfy the system goals hence they satisfy stakeholders needs.

The alignment process contains two steps:

- First, alignment links are graphically specified to be well understood, documented and validated by all the stakeholders.
- Second, a formalization of these alignment links is achieved to prove their consistency. New proof obligations are generated in addition with existing proof obligations of type invariant preservation, feasibility of non-deterministic actions and well-definedness.

A formal verification step is performed to check the resulting EVENT-B specification. Discharging all proof obligations (existing and alignment proof obligations) allows to prove that system requirements are formally aligned with HLA elements.

This chapter is organized as follows. Section 2 presents an excerpt of a case study that we will use to exemplify our works. Section 3 gives a presentation of the proposed HLA SYSML extensions and their translation to EVENT-B. This is followed by a requirements and HLA alignment approach in Section 4 in which we describe the SYSML/KAOS requirements modeling approach and the process to generate EVENT-B formal specification

in Subsection 4.1. Section 5 presents an illustration of our methodology on use cases. Finally, Section 6 reports our conclusions.

2 ATO over ERTMS case study excerpt

Throughout this section, we use an extract of a case study inspired from ATO over ERTMS system (Automatic Train Operation over European Railway Traffic Management System)[Bon et al., 2022]. A railway system may be controlled using Automatic Train Operation (ATO): this is one of the challenging tasks of the railway industry. In railway, four different grades of automation (GoA) are used. With **GoA2** traction and braking are automatic but the driver ensures the environment monitoring and is able to switch towards manual driving if necessary.

This subsystem is activated and deactivated by the driver, which implies the enabling/disabling of the **Railway System**. The driver is also responsible for switching driving mode between manual or automatic. The **GoA2** is composed of two subsystems : **OnBoard** and **Track**. The **OnBoard** subsystem is responsible for executing the driving mode chosen by the driver and updates the state of the **Track** subsystem with the current driving mode. The **GoA2** subsystem functions in a global framework called ATO over ERTMS. The specification is based on a normative and prenormative documentation. The ATO itself is not specified and the studied system is only the context and interfaces of the ATO. The same phenomenon occurs with relationships with the track system: ERTMS specifies the OnBoard system and interfaces with the track. As a consequence, the track side is not specified as it is linked to national specific implementations.

3 High-level architecture modeling and formal verification

Complex systems are systems composed of many components which may interact with each other, such as air traffic management system, railway systems, smart grid, autonomous automobile systems, medical monitoring, industrial control systems, robotics systems, etc. Their behavior is intrinsically difficult to model due to the dependencies, competitions, relationships, or other types of interactions between their parts or between a given system and its environment. In many cases, it is useful to represent such a system as a network where nodes represent components and links their interactions. Therefore, designing HLA of these systems depends on solutions that can address interplay between their sub-systems.

This HLA should be represented as a layered hierarchy of sub-systems. It must enable the specification of the main functional elements of the system, together with its interfaces and interactions.

For that, we propose to combine SysML and the Event-B formal method. The choice is on SysML rather than UML while it offers a set of concepts more relevant to model systems and as previously presented, is recommended by the AFT project partners. Indeed, the AFT project reuses the RailTopoModel [1] that contains a SysML-based functional ontology of a railway infrastructure. Moreover, the European initiative EULYNX [2] has defined a standard SysML-based model of railways signalling system components. Event-B allows to specify systems rather than just software and it is already used in many safety-critical systems [Lecomte et al., 2017]. Its use is also recommended in the ASTRAIL study [ASTRAIL, 2017]

Figure 2.2 presents an overview of the process of modeling HLA and its formalization into Event-B models.

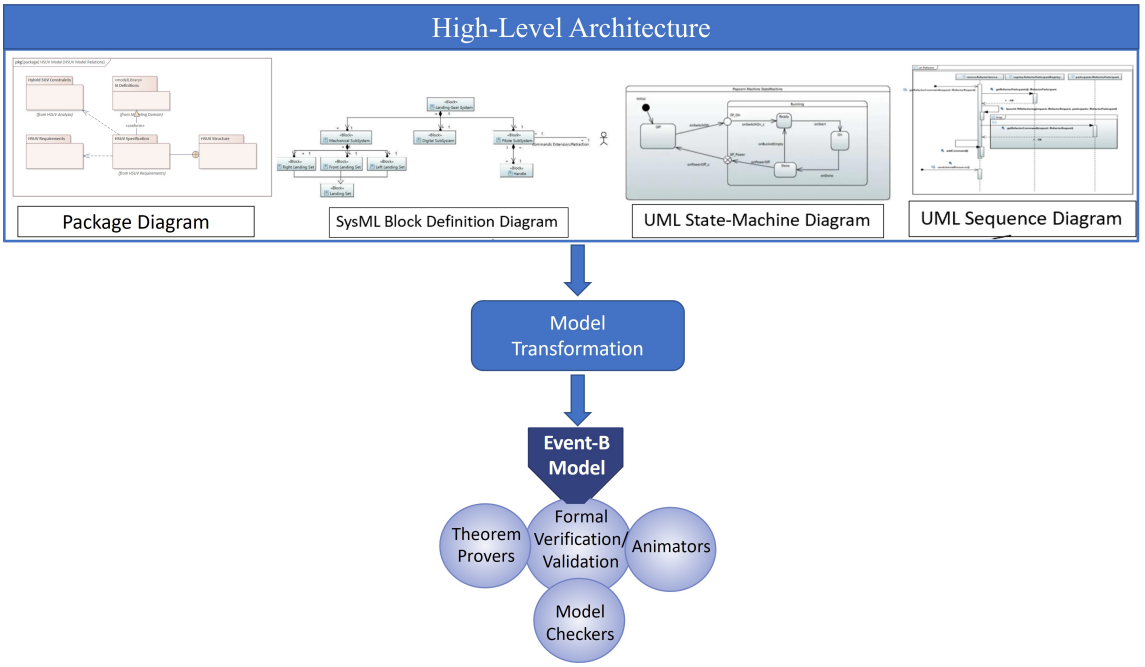

Figure 2.2 – HLA modeling and formalization approach

This process is composed of two steps

— The first step consists in modeling high-level architectures using SysML diagrams.

1. http://www.railtopomodel.org/en/. It is a standard for the representation of railway infrastructure-related data
2. https://www.eulynx.eu/

Four kinds of SysML diagrams have been selected: package, block definition, state-machine and sequence diagrams.

The refinement and decomposition mechanisms are interesting characteristics of Event-B that facilitate a step-by-step design and make proofs easier to discharge. Therefore, we propose to extend SysML with these relevant mechanisms to enable an automatic translation. These extensions are applied on two SysML diagrams, the package and sequence diagrams.

— The second step consists in translating SysML diagrams into Event-B models. This model translation is implemented using three sets of rules: a set for elements related to a package; a set for the SysML refinement extensions; a set for the SysML decomposition extensions. This step is performed in two phases.

1. A model-to-model transformation to implement the above rules. It takes as input the SysML extended meta-model and it produces as output an Event-B model conform to the Event-B meta-model.
2. A model-to-text transformation to generate Event-B textual formal specifications. This textual specification can be introduced into provers such as AtelierB [AtelierB, 1990], model-checkers and animators such as ProB [ProB, 2003] to verify the consistency of the modeled HLA of the system and to be validated by domain experts through models animations.

3.1 High-level architecture modeling

HLA is a layered hierarchy of sub-systems that collaborate together to satisfy parent system goals and to represent interactions with environment entities. Moreover, a sub-system can have its own life and can exist independently besides its contribution in parent system life cycle.

We have chosen to represent HLA by four SysML diagrams: package, block definition, state-machine and sequence diagrams. This choice is based on the description presented in the **SysML for systems engineering** book [Holt and Perry, 2008]:

— Package diagram: Its main use is to show high-level relationships between groups of things in a model. It is used to display the way a model is organized in the form of a package containment hierarchy.

— Block Definition Diagram (BDD): It is the most widely used diagram in SysML for

modeling the static structure of a system. It is also the richest diagram in terms of the amount of syntax available to the modeler. It also allows to adopt design techniques for creating extensible system HLA structures, a practice that masters the complexity and to change the design as stakeholders needs evolve. As with all SysML diagrams, it is not necessary to use every piece of syntax, as experience has shown that 80 % of any modeling task can be achieved by using approximately 20 % of block definition diagram syntax.

— State-machine diagram: it is one of the most widely used diagrams to describe state-dependent behavior of an object. State-machine diagrams are usually applied to any element that has behavior such as: actors, use cases, methods, subsystems, systems etc. They are typically used in conjunction with sequence diagrams and have very strong relationships with BDDs.

— Sequence diagram: it is an interaction diagram that details how operations are carried out. It captures high-level interactions between users and the system, between the system and other systems, or between sub-systems, suitable to represent HLA systems/sub-systems interplay. Adding to that, sequence diagrams are very powerful when used as a consistency check between various interacting objects that already have their internal behavior defined with state-machine diagrams.

3.1.1 SysML and Event-B refinement and decomposition mechanisms extensions

To model a HLA hierarchy, new packages are designed for each system (a parent system or a sub-system). Each package is composed of a set of diagrams:

— a BDD represents systems (parent systems or sub-systems) as *blocks* and *associations* which link sub-systems to their parent system and to environment entities.

— a state-machine diagram, one for each system of the BDD, specifies the behavior of the system by a set of its different *states* and the *transitions* process between these states.

— a sequence diagram represents the life cycle of the current system, the interactions between its sub-systems and how they cooperate to satisfy the objectives of the parent system.

3.1.1.1 Package diagram and its extensions

Package diagram is a static structural diagram that shows the relationships among packages and their contents. It allows to group the structures of a model and defines high level relationships between these groupings. This diagram encompasses two main elements:

— A *package* represents a graphical node. It is made up of a number of *packageable elements*. In SysML, almost any element can be enclosed within a package. In HLA modeling, we are interested in block, state-machine and sequence diagrams as packageable elements in the package. A package itself is also a packageable element and thus can contain other packages.

— A *dependency* represents a graphical path that links different packages and how they depend on each other. Note that the semantics of dependency is informal and can be adapted for particular needs.

Figure 2.3 shows an example of package diagram that encompasses a set of packages related between each other using a dependency relationship.

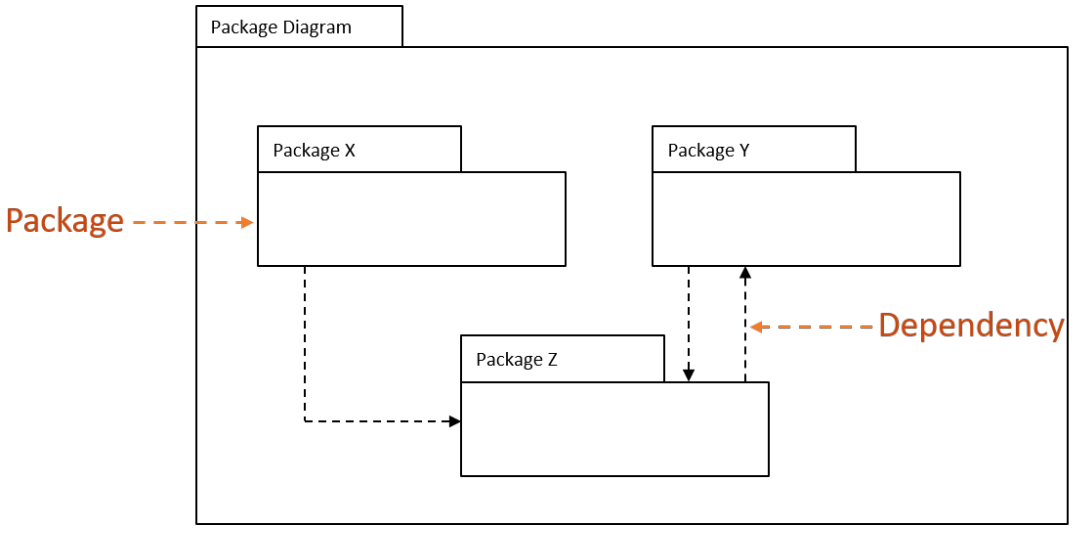

Figure 2.3 – Package diagram example

To model a system HLA in a layered hierarchy of sub-systems, two kinds of relationships have been introduced.

— The first one is inspired from the refinement link of Event-B and is called *HLA_refines*. It is defined between two packages and is used to detail the behavior of the parent package. For this, new blocks and a new sequence diagram are introduced in the child package. This new sequence diagram describes the interactions between blocks

to satisfy the parent behavior.

— The second relationship, called *HLA_ decompose*, comes from the fact that some blocks of a package can be considered as sub-systems because they have their own life and can exist independently of the other blocks. In this case, they become new packages and the link with the parent package is *HLA_ decompose* link. This concept corresponds to the decomposition mechanism of Event-B, more precisely the shared-event decomposition since the systems/sub-systems we consider behave as distributed systems.

To represent these system/sub-systems relationships, the extract of SysML package diagram meta-model used for our HLA modeling encompasses one or more packages related one to the other using a *dependency* relationship. This relationship is extended by introducing new meta-classes, as described by the grey boxes in Figure 2.4 *HLA_ refines* and *HLA_ decompose*, as sub-classes of the meta-class *Dependency*.

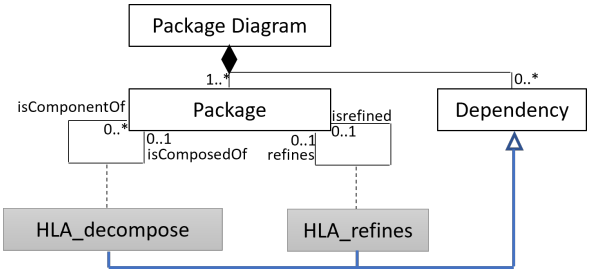

Figure 2.4 – SysML extended package diagram meta-model

Figure 2.5 illustrates the use of the proposed package diagram extensions. *HLA_ refines* extension is used between Packages 1 and 1.1 while Package 1.1 describes the sub-systems interplay to satisfy the parent system behavior represented in the Package 1. Package 1.1 sub-systems can exist independently, therefore, *HLA_ decompose* extension is used to represent the system/sub-systems decomposition into Packages 2, 3 and 4. Each of these packages describes a separate sub-system. Packages 2 and 4 represent sub-systems that could behave as a parent system while it encompasses nested components and sub-systems. Therefore, these two packages are *HLA_ refine* into Packages 2.1 and 4.1 that describe the interplay of their sub-systems.

3.1.1.2 HLA restricted BDD

A BDD is a structural diagram. As HLA is represented by a set of system/sub-systems layered hierarchy we are only interested in basic modeling elements of this diagram. These

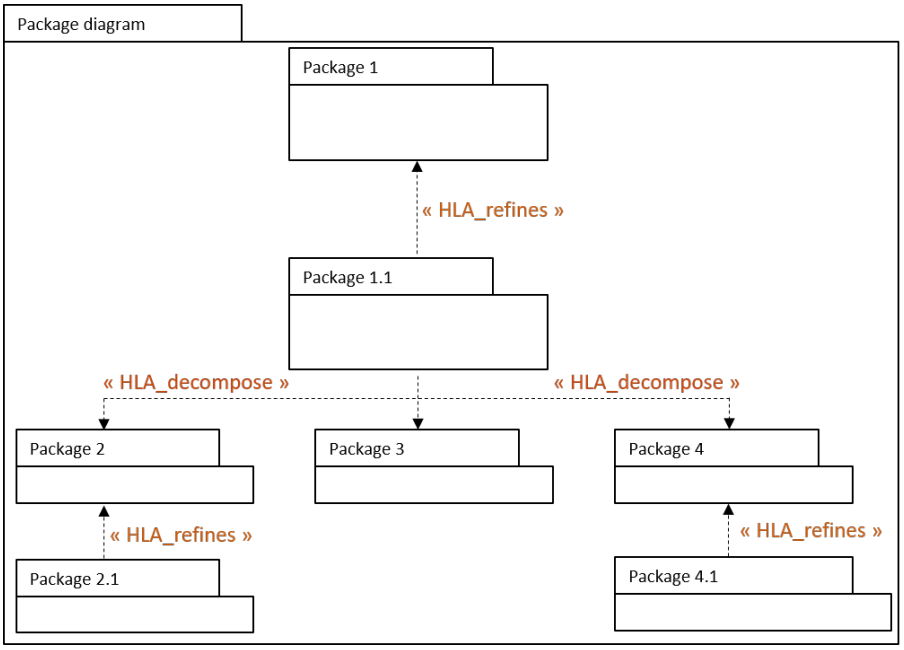

Figure 2.5 – Package diagram extension application example

two basic elements are *Block* and *Relationship*.

— A *Block* defines a collection of features used to describe a system, sub-system, component or other system elements.

— A *Relationship* relates together one or more blocks. It participates in describing the structure of a system, sub-system or component. In SysML, block relationships encompass many types of links. To model HLAs, we are interested in two types of relationships: *Association* and *Composition*.

These concepts come from UML class diagrams, with the same semantics. The extract of BDD meta-model, used for HLA modeling, is presented in Figure 2.6. It can encompass one or more *blocks* related to each other using *associations*. Inheritance is expressed by the reflexive association subBlock/superBlock. An association has two ends, also called roles, represented by the *AssociationEnd* class and linked to a block by the *characBlock* association. The Characteristic class groups the common characteristics of the Association End. The *characMultMin* and *characMaxMult* attributes describe the minimum and maximum multiplicities of a role or an association. An association Block is represented by a link of the association *assocBlock*.

Figure 2.7 presents an example of the BDD diagram of the ATO over ERTMS case study excerpt. Block ATOoETCS_GOA2System is the main system and is composed of two subsystems Track and OnBoard related to the main system with a composition association. A

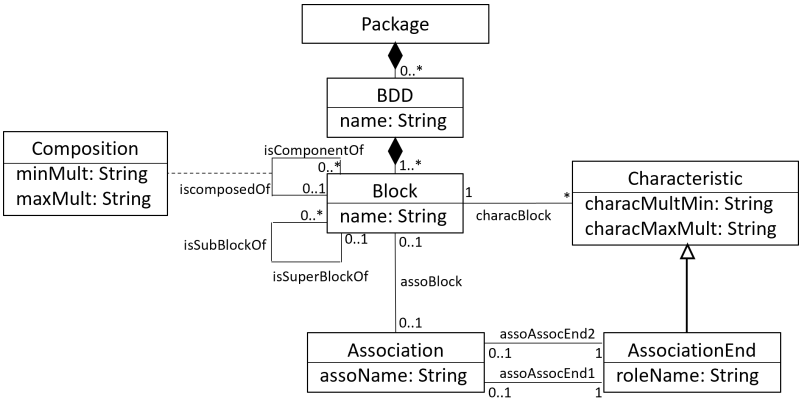

Figure 2.6 – SysML BDD meta-model

simple association interactsWith is presented between the OnBoard system and Pilot which describes the control of the OnBoard system by the Pilot.

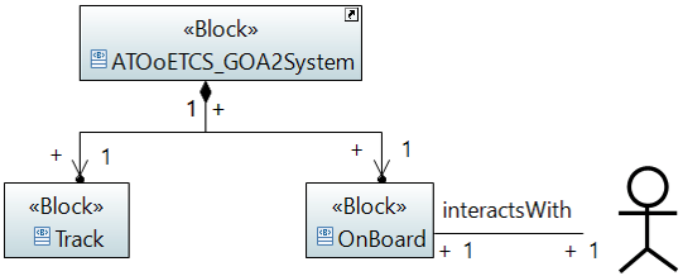

Figure 2.7 – BDD example

3.1.1.3 HLA restricted state-machine diagram

A state-machine diagram is used to model the behavior of a block. Such a diagram is composed of two basic elements: *states* and *transitions*. It describes the state changes of a block instance during its life cycle.

— A *state* is an abstraction of the attribute values and links of an object. Sets of values are grouped together into a state according to properties that affect the gross behavior of the object.

— A *transition* arrow depicts the movement from one state to another. These changes are triggered by events associated to the transitions of the diagram. A system represented as transitions between states is very useful for describing complex behaviors.

We have extracted all the concepts of SysML state-machine diagrams that we need to model HLAs. They are presented in the meta-model of Figure 2.8. It is composed of 0 or

more *states* identified by a name and *transitions* represented by the triggering event name. A state could be a source of an outgoing transition and a target of an incoming transition.

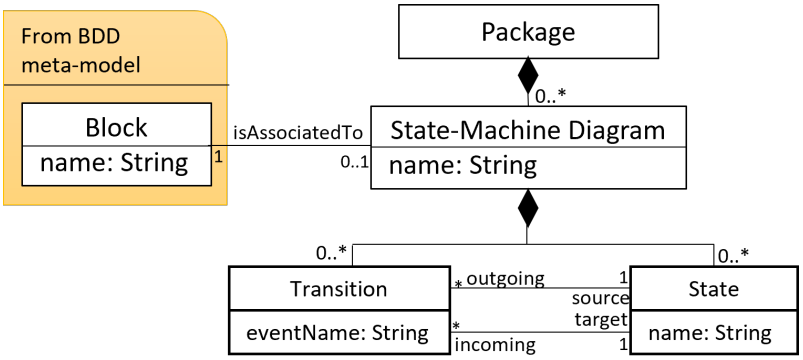

Figure 2.8 – SysML State-machine diagram meta-model

As an example of state-machine diagram, we give an extract of the state-machine diagram that represent the behavior of the OnBoard system. This diagram is shown in Figure 2.9 and it describes the lifecycle of the system with two states BoardForAutomaticDriving and BoardForManualDriving and two transitions SetBoardForManual and SetBoardForAutomatic.

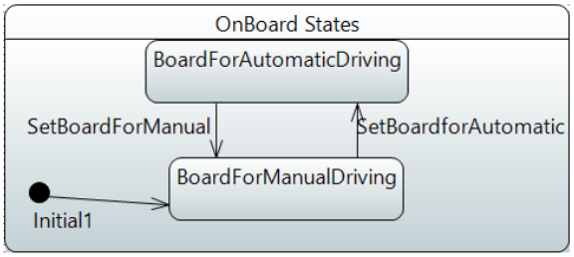

Figure 2.9 – State-machine diagram example

3.1.1.4 Sequence diagram and its extensions

A sequence diagram is used to display the interactions between users, objects, systems and entities within a system. It presents a dynamic view of a use case, a requirement, or a system, a view that expresses sequences of behaviors and event occurrences over time. All the concepts of SysML sequence diagrams that we need to model HLAs are presented in the meta-model of Figure 2.11:

— An *Interaction* is a behavioral specification that comprises a sequence of communications exchanged among a set of instances within a collaboration to accomplish a specific purpose, such as a parent system goal.

— An object *lifeline* represents the existence of an object over some time. Objects that exist throughout an interaction should appear at the top of the object dimension with their lifelines drawn parallel to the time dimension.

— *Messages* specify communication from one object to another, with an expectation that an activity will be performed by the recipient object.

— The intersection of a message arrow and a lifeline is represented by the element *MessageOccurenceSpecification*.

— A *GeneralOrdering* represents a binary relation between two *MessageOccurenceSpecification*, to describe that one *MessageOccurenceSpecification* must occur before the other in a valid trace. This mechanism provides the ability to define partial orders of *MessageOccurenceSpecification* that may otherwise not have a specified order.

— *CombinedFragment* is a logical grouping which contains the conditional structures that affect the flow of messages. A combined fragment contains interaction operands and is defined by the interaction operator. An interaction operator defines the semantics of a combined fragment and determines how to use the interaction operands in the combined fragment. An *interactionOperand* is a container that groups the interaction fragments and messages that run if the guard condition is met. If there is no guard condition, the block always runs. An *interactionConstraint* is a constraint used in interactions to guard an operand in a combined fragment.

Figure 2.10 shows an example of a sequence diagram of the OnBoard system. It shows the elements involved in a execution scenario namely Driver, ATOonBoard etc., as well as the message exchanges between the system and actors, or between parts of the system or subsystems, in a chronological manner.

The following constraints/extensions have been defined:

— Each message corresponds to a transition in the state-machine of the block associated to the target lifeline. This association is established using the *signature* property of the sequence diagram message.

— As we have introduced a refinement link between packages, we need to specify how this refinement is elaborated between the refining package and the refined package. The parent system main goal is produced through the interplay of its sub-systems. Each one executes some behaviors to satisfy some whole system tasks. This sub-systems interplay covers the whole system tasks and satisfies the main system goal. The

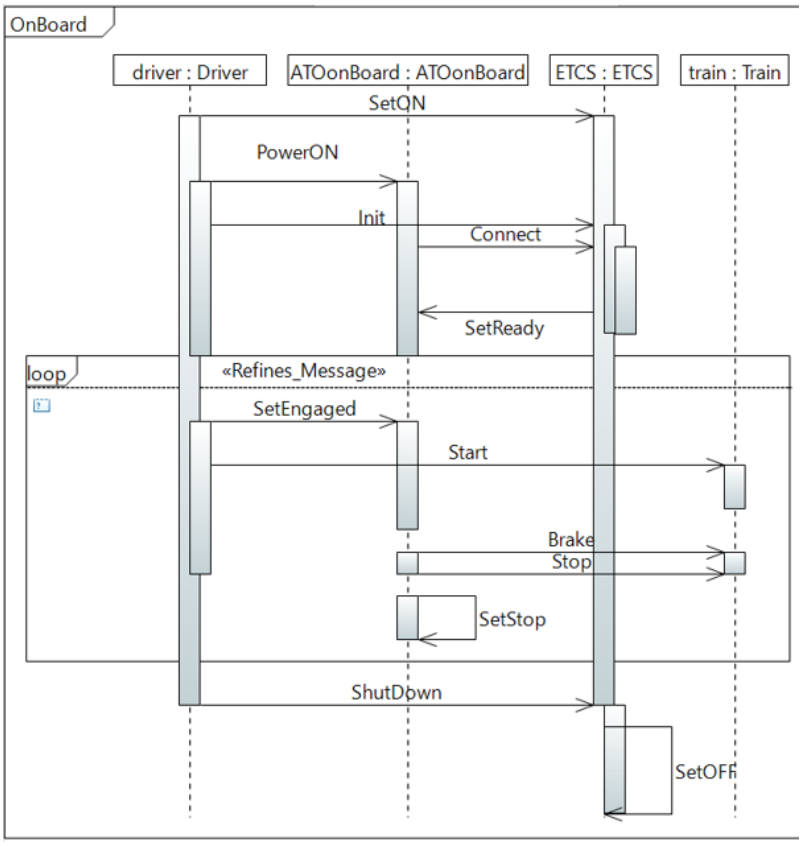

Figure 2.10 – Sequence diagram example

main system life cycle and the sub-systems interplay are represented using sequence diagrams. Therefore, to elaborate the refinement, we opt to refine a message from the parent system sequence diagram with a message resulting from its sub-systems interplay. The choice is on sequence diagram messages while it represents the system life cycle activities. Thus, this is achieved by:

→ Adding a new meta-class in the meta-model, called *Refines_ Message*, sub-class of the meta-class *Message* that contains an attribute of type *Message*, called *Refined_ Message*.

→ *Refines_ Message* used to associate the sequence diagram message resulted from the sub-systems interplay with a message from parent system sequence diagram defined in the *Refined_ Message* attribute.

Figure 2.12 illustrates the use of the proposed sequence diagram extension applied on ATO over ERTMS case study excerpt [Bougacha et al., 2022a]. Excerpt (1) describes the sequence diagram of Component RS of type Railway System with Message EnableSystem. Excerpt (2) shows Message (Activate) exchanged between Components Driver and

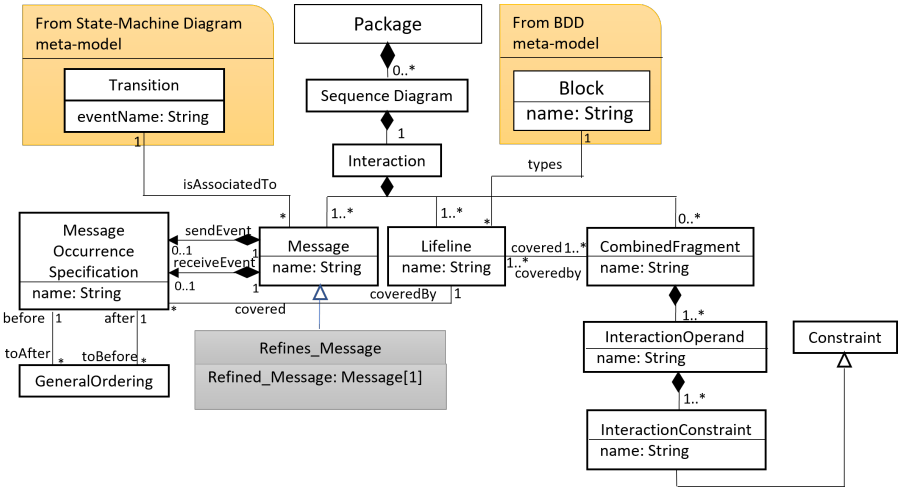

Figure 2.11 – Extract of SysML extended sequence diagram meta-model

ATOoverETCSGoA2, sub-components of Railway System. In this system, the activation of ATOoverETCSGoA2 components implies the enabling of the Railway System. Then, Message Activate refines the parent system message EnableSystem, as precised in Excerpt (3).

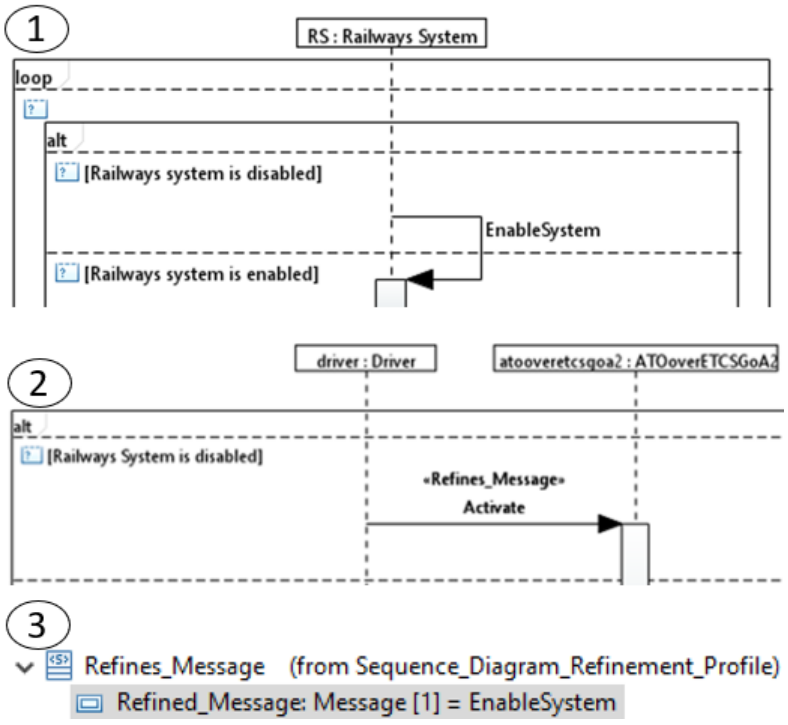

Figure 2.12 – Example of the sequence diagram extensions

3.1.2 SysML HLA modeling process architecture

Figure 2.13 presents the SysML HLA modeling process architecture. As we have already presented, a package diagram is created to design an HLA. This diagram describes the layered hierarchy of system/sub-systems relationships and encompasses a set of packages each of them corresponds to a component of this hierarchy (a system or a sub-system). The packages are presented as follows:

1. The first package Package 1 describes the main system in a BDD and its associated state-machine diagram which defines the system behavior and a sequence diagram that represents its life cycle. Sequence diagram messages are associated with the state-machine diagram transitions.

2. The main system represented by Package 1 is a parent system composed of a set of sub-systems that collaborate to satisfy the parent system main goal. From this end, the second package Package 1.1 is created and it refines Package 1 using the *HLA_ refines* stereotype. The parent system to sub-systems composition relationship is designed in the BDD. A state-machine diagram is found and associated to each sub-system behavior. A sequence diagram is created to define the life cycle of the sub-systems interplay for the purpose to satisfy the parent system goal. The result of this interplay is represented by a sequence diagram message that refines, using the*Refines_ Message* stereotype, a message from the parent system sequence diagram.

3. Sub-systems can be considered as independent systems that can exist by their own besides their participation in the main system life cycle. To this aim, a decomposition mechanism using *HLA_ decompose* stereotype is applied on the parent system and a new package is created for each sub-system (Package 2, Package 3, Package 4). This decomposition is applied on the package Package 1.1 because it is the refinement package of Package 1 in which we have introduced the interplay of the parent system corresponding sub-systems.

4. From step 3, If one of these sub-systems is a parent system that is composed of other sub-systems then these steps should be re-executed from step 1 until we arrive to a package with no encompassed sub-systems such as Package 3.

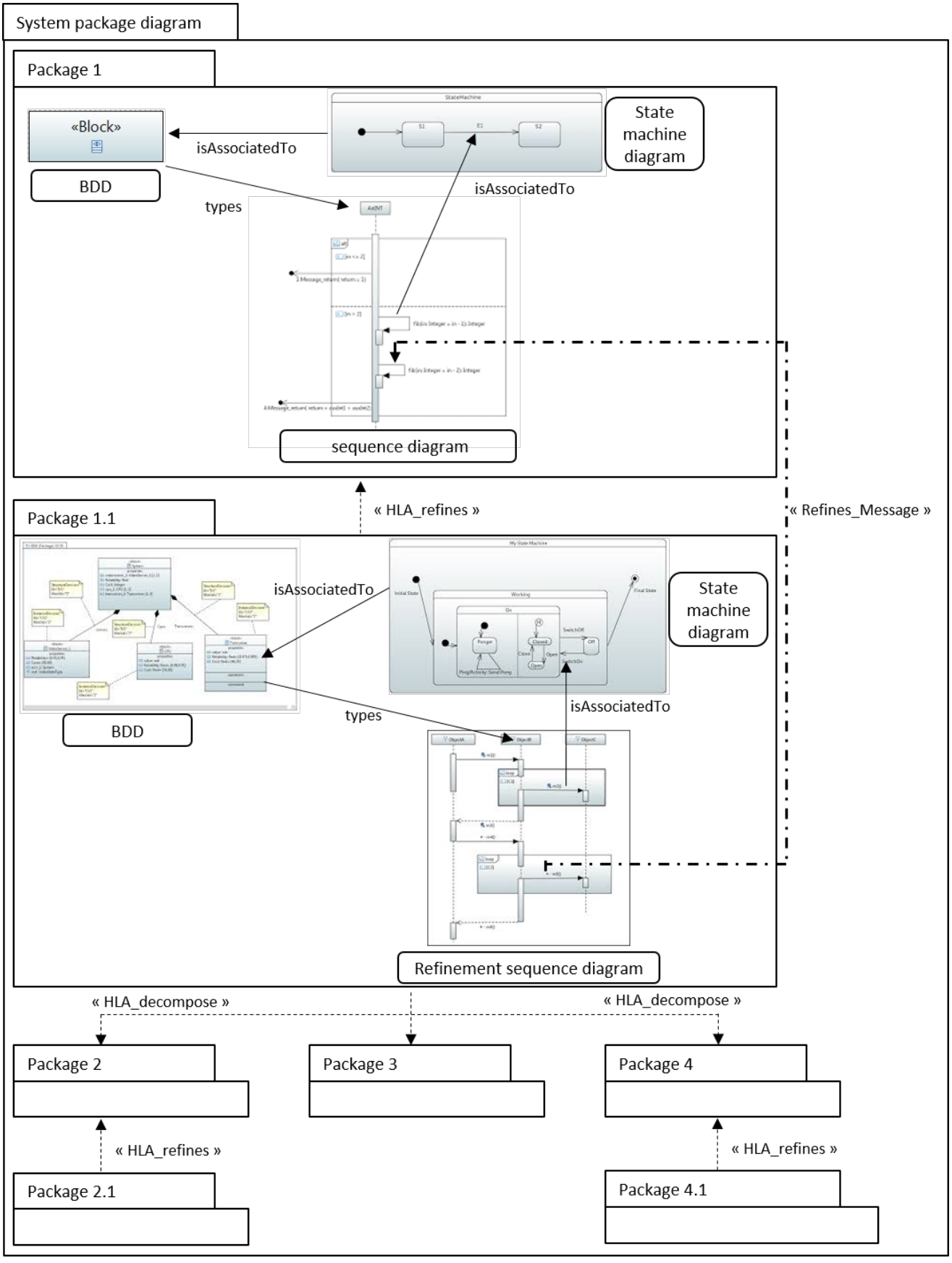

Figure 2.13 – SysML HLA modeling process architecture

3.2 SysML to Event-B Translation

HLA modeling with SysML diagrams is a designer easy practice to define graphical views of critical systems which give a better understanding. However, it may lead to some design errors due to the lack of precise semantics. So, a formal notation is required to rigorously check the correctness of such systems and guarantee the validation and the verification of their specification. In this subsection, we present the translation rules that enable an automatic translation of Event-B specification from HLA models enriched with safety relevant Event-B mechanisms. This step is performed in two phases: model-to-model transformation and model-to-text transformation.

3.2.1 Model-to-Model transformation

Model-to-model transformation is a model-driven process that enables to derive a target model from a source model that are conform to a source and a target meta-models, and create one or more mapping declarations that define relationships between these two meta-models. To this end, this translation requires source models which are, in our work, HLA models that are conform to the extended SysML meta-models presented in the previous subsection and target models that are Event-B models conform to Event-B meta-model. However, a standardized meta-model for Event-B is still not available. The one proposed in the Rodin Platform [Abrial, 2010] is of high complexity and cannot cover our needs. Therefore, to conduct this model-to-model transformation we propose an Event-B meta-model conform to the Event-B notation used in AtelierB and restricted to the concepts that are relevant to our use.

3.2.1.1 Event-B meta-model

Our proposed Event-B meta-model is shown in Figure 2.14. It presents EventB_SPEC as the root meta-class. This meta-class defines an Event-B specification. It is composed of zero or more CONTEXT and zero or more MACHINE.

— A CONTEXT describes the static part of a system, it is composed of:

- Zero or more CONSTANTS and SETS;
- zero or more AXIOMS mandatory to define constant types and properties;
- A context can be extended with zero or more contexts and it can be seen with zero or more machines.

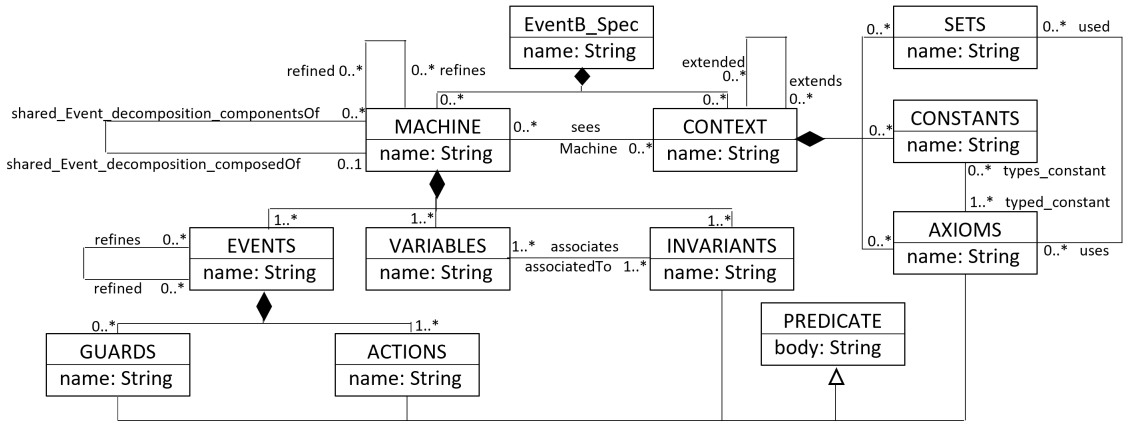

Figure 2.14 – EVENT-B meta-model

— A MACHINE describes the dynamic part of a system, it is composed of:

- One or more VARIABLES which represents the list of state variables of the model;
- One or more INVARIANTS which represents the typing predicates of the various variables and the predicates that the variables should obey;
- One or more EVENTS that represents the list of the various events related to the model. It is composed of a set of GUARDS and ACTIONS. Events can be refined and new ones can be introduced. The refinement of an event has to verify that the guard of the refined event is stronger than the guard of the abstract one and the effect of the refined action is stronger than the effect of the abstract one.
- A machine can refine or be refined by another machine and it can see zero or more contexts.
- A machine can be decomposed into one or more sub-machines following the shared-event decomposition approach. The shared-event decomposition is a set of events that are synchronized and shared by sub-components. This approach defines a partial version of a global event in each sub-machine, when the variables of a global event are distributed between separate sub-machines. This is to simulate the action of the global event on the considered variables. We have adopted the shared-event decomposition approach for our HLA modeling as we can consider that a system composed of sub-systems acts as a distributed system

Some EVENT-B concepts presented in section 2.4 are not supported in this meta-model because they are not relevant to our use. We give as an example the use of THEOREMS, in

our HLA we don't model any type of these theorems. Also VARIANTS, in our translation events are created from sequence diagram message in a specific scenario. Then, these events cannot be convergent. Events parameters are not supported in our meta-model because the proposed translation is an end-to-end translation and all defined elements in the HLA have corresponding element in the Event-B specification.

3.2.1.2 Translation Rules

Three sets of rules have been defined.

— The first one considers elements related to a package.

— The second one deals with the SysML refinement extensions.

— The last one deals with SysML decomposition extensions.

Translation of package elements.

Tables 2.1 and 2.2 summarized the rules that translate all the elements of a package, which includes package diagram, BDD, state-machine diagram, sequence diagram elements. It should be noted that in these two tables, E_X designates the result of the translation of Element X.

Rule 1 of Table 2.1: a package diagram gives an Event-B project containing the HLA Event-B specification.

Rule 2 of Table 2.1: a package, inside a package diagram, that is not a decomposed package gives an Event-B machine and an Event-B context.

		SysML concepts		Event-B concepts	
Rule	Translation of	Element	Constraint	Element	Constraint
1	Package Diagram	PD	PD is a Package Diagram	E_PD	E_PD ∈ EventB_Spec
2	Package that is not a decomposed package	P	P ∈ SysML_Package P ∉ ran(HLA_decompose)	E_P_M E_P_CONT	E_P_M ∈ MACHINE E_P_CONT ∈ CONTEXT E_P_M SEES E_P_CONT

Table 2.1 – Translation rules for a package diagram

The rules of Table 2.2 are applied for the elements of a given package. Note that the translation rules of elements of the BDD meta-model (Figure 2.6), in particular *Block*, *Association*, *AssociationEnd* and *Characteristic* are the same as those used for translating the equivalent concepts in class diagrams [Laleau and Mammar, 2000]. Mainly, an association between two blocks is modeled as a relation between two constants representing

these blocks instances. The relation become a function, an injection,... depending on the multiplicity of the association.

Figure 2.15 shows an example of the translation of an association called interactsWith between the Pilot and the OnBoard subsystem. The EVENT-B formalization of this association shown in Listing 1 is presented as a bijective function between the sets PILOT and OnBoard created from the corresponding elements of HLA.

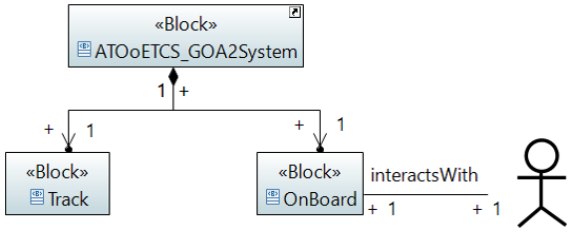

Figure 2.15 – Association translation rule example

```
SYSTEM ATOoETCS_GOA2SystemL1_CONT
SETS
    ...; OnBoard; PILOT
CONSTANTS
    pilot, onboard, interactsWith,
    ...
PROPERTIES
    interactsWith ∈ PILOT ↦
    OnBoard
    ...
END
```

Listing 1 – Example of a translation rule for an association.

The first three rules in Table 2.2 are rather straightforward. Rule 6 expresses that a lifeline L associated to a block B is translated by a constant E_LC, instance of the abstract set E_B_S associated to B, and a variable E_LV that represents the current state of E_LC in the state-machine associated to B.

Rule 7 needs to be more precisely defined. An EVENT-B event E_M obtained from the translation of a message M is of the form:

SELECT G_M **THEN** A_M **END**

where G_M are the guards and A_M are the actions. G_M and A_M are obtained as follows.

— Let SQ be a sequence diagram.

— Let CB be a combined fragment in the sequence diagram that contains an interaction operand IO.

— A message M is defined between two lifelines in SQ, L_1, its origin, and L_2, its destination (L_1 and L_2 are not necessary different).

— L_1 (L_2, resp.) is associated to block B_1 (B_2, resp.).

— Let SM_1 (SM_2, resp.) be the state-machine associated to B_1 (B_2, resp.).

		SysML concepts		Event-B concepts	
Rule	Translation of	Element	Constraint	Element	Constraint
3	Block	B	B ∈ SysML_Block	E_B_S	E_B_S ∈ SETS
4	State-machine of a block	SM, B	SM ∈ SysML_State-machine SM is associated to B	E_SM	E_SM ∈ SETS
5	State-machine states	S_1,S_2, ... S_n SM	S_i ∈ SysML_States SM ∈ SysML_State-machine S_i is a state of SM	E_S_1,E_S_2, ... E_S_n	E_S_i ∈ CONSTANTS E_SM={E_S_1,E_S_2,... E_S_n}
6	Lifeline of a sequence diagram associated to a block	L,B, SM	L ∈ SysML_Lifeline L is an instance of block B, SM is the state diagram associated to B B and SM have already been translated	E_LC, E_LV	E_LC ∈ CONSTANTS E_LC ∈ E_B_S E_LV ∈ VARIABLES (E_LV ∈ {E_LC} → E_SM) ∈ INVARIANTS (E_LV :∈ {E_LC} → E_SM) ∈ INITIALISATION
7	Sequence diagram message that is not a refined message	M	M ∈ SysML_Message	E_M	E_M ∈ EVENTS

Table 2.2 – Translation rules for elements of a package

— M is associated to a transition T of SM_2.

— Let SS_2 be the source state of T and TS_2 be the target state of T.

Note that Rule 5 gives E_SS_i (E_TS_i, resp.) as the Event-B elements associated to SS_i (TS_i, resp.). Rule 6 gives E_L_iC and E_L_iV as the Event-B elements associated to L_i.

- Calculation of A_M

  $A\_M \triangleq E\_L_2V(E\_L_2C) := E\_TS_2$

  The current state of E_L_2C corresponding to the lifeline L_2 is the target state of the transition T.

- Calculation of G_M

  — If M is the first message of SQ then

  $$G\_M \triangleq E\_L_2V(E\_L_2C) = E\_SS_2$$

  — If M is encompassed in IO constrained by the interaction constraint G_IC then

  $$G\_M \triangleq G\_IC \wedge E\_L_2V(E\_L_2C) = E\_SS_2$$

We precise that G_IC is defined using a lifeline L associated to a block from

the parent system and its current state CS_L. Then

$$G\_IC \triangleq E\_LV(E\_LC) = E\_CS_L$$

— Else, let $Prev\_M$ be the message of SQ that precedes M. Its lifeline destination is necessary L_1 and $Prev\_M$ corresponds to a transition T' of SM_1 whose source state is SS_1 and target state TS_1.

$$G\_M \triangleq E\_L_1V(E\_L_1C) = E\_TS_1 \wedge E\_L_2V(E\_L_2C) = E\_SS_2$$

This means that M is triggered after $Prev\_M$ (i.e. the current state of E_L_1C is the target state of the transition T' and the current state of E_L_2C is the source state of T).

It should be noted that this ordering between messages M and $Prev\_M$ is defined by the *GeneralOrdering* meta-class of the sequence diagram meta-model.

Translating SysML refinement extensions to Event-B. Two rules are defined to translate the two SysML package and sequence diagram refinement extensions.

— **Machine Refinement Rule** is defined as follows: Let P_1 and P_2 two SysML packages such that P_2 *HLA_refines* P_1. P_1 and P_2 are translated into Event-B according to Rule 2 and give $E\_P_1\_M$ and $E\_P_2\_M$ machines.

In $E\_P_2\_M$, two clauses are added to express the SysML package refinement:

— $E\_P_2\_M$ **REFINES** $E\_P_1\_M$

— $E\_P_2\_M$ **SEES** $E\_P_1\_CONT$

— $E\_P_1\_M$ variables are copied in $E\_P_2\_M$

— $E\_P_1\_M$ variables initialisation is copied in $E\_P_2\_M$

— **Event Refinement Rule** is defined as follows:

Let P_1 and P_2 two SysML packages such that P_2 *HLA_refines* P_1. P_1 and P_2 are translated into Event-B according to Rule 2 and Machine Refinement Rule. This gives $E\_P_1\_M$ and $E\_P_2\_M$ machines such that $E\_P_2\_M$ refines $E\_P_1\_M$.

Let M_1 (M_2, resp.) a SysML message of the sequence diagram of P_1 ($_2$, resp.) such that M_2 *Refines_Message* M_1. M_1 is translated according to Rule 7:

$$E\_M_1 \triangleq \textbf{SELECT } G\_M_1 \textbf{ THEN } A\_M_1 \textbf{ END}$$

Then M_2 is translated by:

E_M2 ref $E\_M1 \triangleq$ **SELECT** $G\_M_1 \wedge E\_LVM_2(E\_LCM_2) = E\_SSM_2$
THEN $A\_M_1 \parallel E\_LVM_2(E\_LCM_2) := E\_TSM_2$ **END**

Translating SysML decomposition extensions to Event-B. As already stated, we use the shared-event decomposition approach [Butler, 2009b] of Event-B to translate the SysML package decomposition extension.

— First a new Event-B machine called Interface that corresponds to each of the decomposed machines representing sub-systems is created.

— Then all the variables of the machine to be decomposed are assigned to one of these decomposed machines.

— The elements from the machine to be decomposed linked to a variable assigned to an interface are also assigned in this interface.

Following the shared-event decomposition mechanism, all the variables of the machine to be decomposed must be assigned to one of the decomposed machines. However, if the machine to be decomposed is a refinement machine that refines an abstract one then it contains also the redefinition of abstract variables coming from refinement. We recall also that after decomposition each sub-system can function independently from its participation in the main system. Therefore, the interface that represents each sub-system encompasses only elements related to this sub-system with no relationship with the parent system. That is why, we have created a supplementary new Event-B machine called Refinement_Interface that will get from the decomposed machine all abstract variables and their associated elements coming from refinement.

The Machine Decomposition Rule is illustrated in Figure 2.16 and defined as follows.

Let P, P_1 and P_2 three SysML packages such that P is $HLA\_decompose$ into P_1 and P_2. we recall that P_1 and P_2 are sub-systems of P and then corresponds to the blocks B_1 and B_2 in P. P is translated into Event-B according to Rule 2 and give E_P_M machine.

— E_P_M machine is shared-event decomposed into two machines called $E\_P_1\_Interface$ and $E\_P_2\_Interface$ that correspond to P_1 and P_2.

— Each $E\_P_i\_Interface$ contains the elements of E_P_M linked to the B_i block: SEES clause, variables, invariant and the events that read or modify these variables.

— If E_P_M is a refinement machine then a new interface called $E\_P\_Refinement\_Interface$ is created. It contains the elements of E_P_M coming from the refinement (abstract variables and their related elements).

— $E\_P_i\_Interface$ is refined by a machine called $E\_P_i\_M$ that contains the translation of the elements of P_i (blocks, sequence diagram, state-machines).

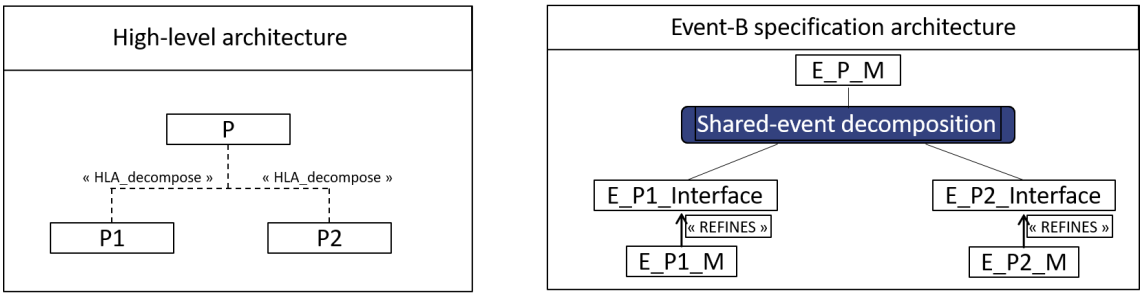

Figure 2.16 – Decomposition extension translation to EVENT-B

3.2.2 Model-to-Text Translation

This step generates automatically an EVENT-B textual specification from the resulting EVENT-B models of the previous step using Acceleo [Musset et al., 2006]. Acceleo is a template based technology allowing to automatically produce any kind of source code from any data source available in EMF format. This textual specification can be introduced into provers such as AtelierB [AtelierB, 1990], model-checkers and animators such as ProB [ProB, 2003] to verify the consistency of the modeled system. In this way, design errors can be detected and invariant violation can be discovered.

3.2.3 Example of application of the translation rules

To give a better understanding of this translation, we present here an application example of these rules on an extract of HLA models. This extract is taken from the ATO over ERTMS case study presented in [Bougacha et al., 2022a]. Figure 2.17 shows the package ATOoverETCSGoA2 that includes a BDD describing the ATOoverETCSGoA2 system and its sub-systems Track and OnBoard. State-machine diagrams are created for each sub-system to specify their behaviors and a sequence diagram describing the sub-systems interplay to satisfy the parent system behavior is designed.

Listings 2 and 3 show the generated EVENT-B specification that represents the structural and dynamic parts from this HLA excerpt after application of the rules described in Section 3.2.1.2. Listing 2 presents the context ATOoverETCSGoA2_CONT which defines the structural part and the machine ATOoverETCSGoA2 shown in Listing 3 which defines the dynamic part.

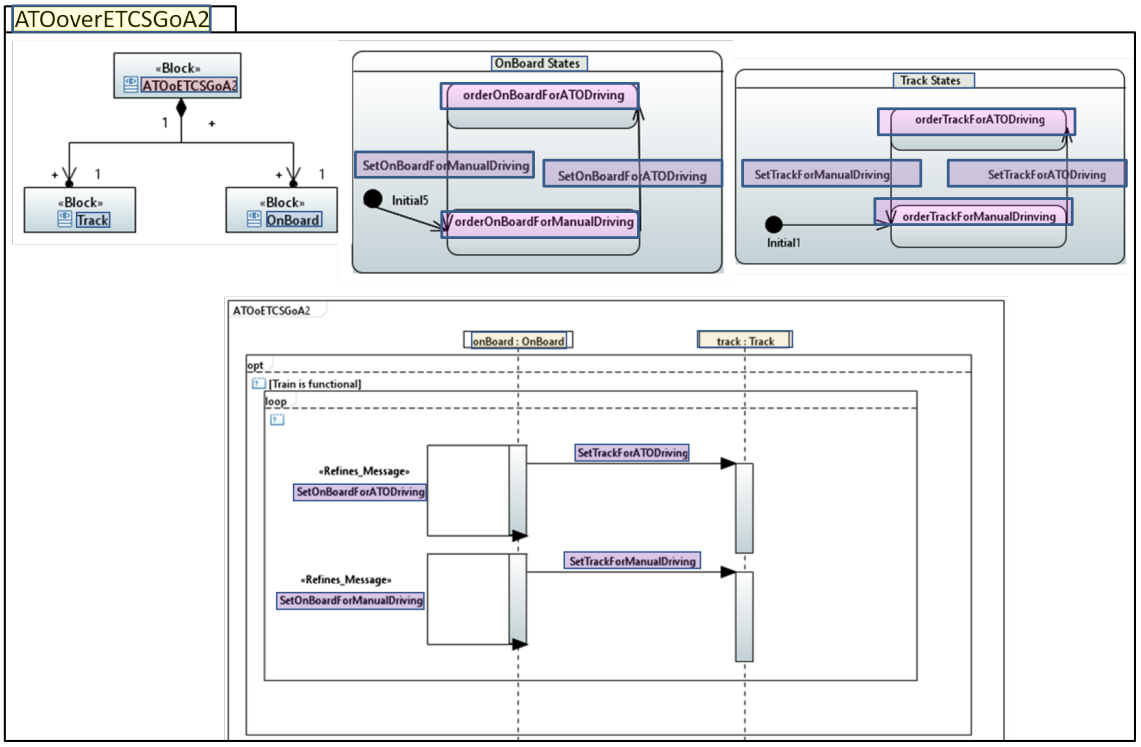

Figure 2.17 – SysML to Event-B translation rules application example

Therefore, the steps of the translation process for the example are:

— First, the package ATOoverETCSGoA2 (framed in yellow) is transformed into a context ATOoverETCSGoA2_CONT and a machine ATOoverETCSGoA2 that sees the created context. (rule 2 in Table 2.3)

— The ATOoverETCSGoA2 system is a sub-system that corresponds to a parent system. The decomposition of the machine created from the parent system package allows to create a set of interfaces each of which corresponds to a sub-system (see Sections 3.1.1.1 and 3.2.1.2). Therefore, the machine ATOoverETCSGoA2 refines ATOoverETCSGoA2_Interface. It gets from this interface all its seen context, variables and their initialisations. (Machine refinement rule in Section 3.2.1.2)

— BDD blocks Track and OnBoard (framed in blue) are transformed into two sets Track and OnBoard. (rule 3 in Table 2.3)

— Sequence diagram lifelines onboard and track are translated into two constants: onboard such that onboard ∈ OnBoard

and track such that track ∈ Track. (rule 6 in Table 2.3)

— Two variables onboardState and trackState are created from the sequence diagram lifelines onboard and track. (rule 6 in Table 2.3)

		SysML concepts		Event-B concepts	
Rule	Translation of	Element	Constraint	Element	Constraint
1	Package Diagram	PD	PD is a Package Diagram	E_PD	E_PD ∈ EventB_Spec
2	Package that is not a decomposed package	P	P ∈ SysML_Package P ∉ ran(HLA_decompose)	E_P_M E_P_CONT	E_P_M ∈ MACHINE E_P_CONT ∈ CONTEXT E_P_M SEES E_P_CONT
3	Block	B	B ∈ SysML_Block	E_B_S	E_B_S ∈ SETS
4	State-machine of a block	SM, B	SM ∈ SysML_State-machine SM is associated to B	E_SM	E_SM ∈ SETS
5	State-machine States	S_1,S_2, $\ldots S_n$ SM	S_i ∈ SysML_States SM ∈ SysML_State-machine S_i is a state of SM	E_S_1,E_S_2, $\ldots$E_S_n	E_S_i ∈ CONSTANTS E_SM={E_S_1,E_S_2,$\ldots$E_S_n}
6	Lifeline of a Sequence Diagram associated to a block	L,B, SM	L ∈ SysML_Lifeline L is an instance of block B, SM is the state diagram associated to B B and SM have already been translated	E_LC, E_LV	E_LC ∈ CONSTANTS E_LC ∈ E_B_S E_LV ∈ VARIABLES (E_LV ∈ {E_LC} → E_SM) ∈ INVARIANTS (E_LV :∈ {E_LC} → E_SM) ∈ INITIALISATION
7	Sequence Diagram Message that is not a refined message	M	M ∈ SysML_Message	E_M	E_M ∈ EVENTS

Table 2.3 – Translation rules application table

— State-machine diagrams OnBoard States and Track States (framed in green) defining sub-systems behaviors are transformed into two sets OnBoardStates and TrackStates. (rule 4 in Table 2.3)

— States of the state-machine diagrams (framed in pink), such as orderOnBoardForATO-Driving from the state-machine diagram OnBoard States, are mapped into constants orderOnBoardForATODriving such that orderOnBoardForATODriving ∈ OnBoardStates. (rule 5 in Table 2.3)

— Once the variables created their typing invariants and initialisation are generated. For example the typing invariant of Variable onboardState is: onboardState ∈ OnBoard → OnBoardStates. (rule 6 in Table 2.3)

— Variable atooveretcsgoa2State comes from the refined machine ATOoETCSGoA2_Interface. (Machine refinement rule in Section 3.2.1.2)

— Sequence diagram messages (framed in purple) related to state-machine diagram transitions such as SetTrackForATODriving are mapped to events such as SetTrackFo-rATODriving in the machine. (rule 7 in Table 2.3)

— The guard and action of the event are related to the source and target states of the transition associated to the sequence diagram message. We give for example the guard:

trackState(track) = orderTrackForManualDrinving

and the action:

trackState(track) := orderTrackForATODriving. (Guards and actions calculation rules in Section 3.2.1.2).

— The guard of the event is related to sequence diagram combined fragment constraints and to the message that precedes the current message (see Section 3.2.1.2). The guard associated to the event SetTrackForATODriving is:

onboardState(onboard)=orderOnBoardForATODriving. (Guards and actions calculation rules in Section 3.2.1.2).

— As shown in the sequence diagram, the message SetOnBoardForATODriving *Refines_Message* the message SetATO from the parent system sequence diagram. Therefore, this is translated in the EVENT-B specification by:

SetOnBoardForATODriving ref SetATO.

The event SetOnBoardForATODriving redefines from event SetATO all its guards and actions. (Event refinement rule in Section 3.2.1.2)

```
SYSTEM ATOoverETCSGoA2_CONT
SETS OnBoard; Track; TrackStates; OnBoardStates
CONSTANTS onboard, track, orderOnBoardForATODriving,
    orderTrackForManualDrinving, orderOnBoardForManualDriving,
    orderTrackForATODriving
PROPERTIES
    onboard ∈ OnBoard ∧ track ∈ Track ∧
    ...
    orderOnBoardForATODriving ∈ OnBoardStates∧
    orderTrackForATODriving ∈ TrackStates∧
END
```

Listing 2 – SysML to EVENT-B translation rules application example context.

```
REFINEMENT ATOoverETCSGoA2
REFINES ATOoverETCSGoA2_Interface
SEES ATOoverETCSGoA2_CONT, RailwaysSystemL1_CONT,
    RailwaysSystemL0_CONT
VARIABLES onboardState, trackState, atooveretcsgoa2State
```

```
INVARIANT
    onboardState ∈ OnBoard → OnBoardStates ∧
    trackState ∈ Track → TrackStates
INITIALISATION
    onboardState :∈ {onboard} → OnBoardStates ||
    trackState :∈ {track} → TrackStates ||
    atooveretcsgoa2State :∈ {atooveretcsgoa2} →
    ATOoverETCSGoA2States
EVENTS
    SetTrackForATODriving=
    SELECT onboardState(onboard)= orderOnBoardForATODriving ∧
    trackState(track)= orderTrackForManualDrinving THEN
    trackState(track):= orderTrackForATODriving
    END ;
    SetOnBoardForATODriving ref SetATO=
    SELECT onboardState(onboard)=orderOnBoardForManualDriving ∧
    atooveretcsgoa2State(atooveretcsgoa2)=manualDriving THEN
    onboardState(onboard):=orderOnBoardForATODriving ||
    atooveretcsgoa2State(atooveretcsgoa2):=ATODriving
    END
END
```

Listing 3 – SysML to Event-B translation rules application example context.

3.2.4 HLA formal verification

This verification generates automatically a set of proof obligations corresponding to the modeled functional properties without any non-functional properties integration, and proof obligations from the application of model decomposition. These proofs are of type invariant preservation, non-deterministic action feasibility and well-definedness. The verification step is of paramount importance. It provides a theorem proving method for process verification to detect probable invariants violations during the verification using model checking. The verification process also goes far beyond the simple verification of the structural properties of the model. Precisely, it enables the verification of the advanced decomposition and refinement aspects and system behavior of different states achieved from the initial

state after execution process. The formal verification step checks the correctness of the designed HLA of railway system specification thanks to ProB [ProB, 2003] model checker and animator which is used to discover some errors and invariant violation during the model animation from different execution scenarios or during the verification using model checking. Therefore, our approach provides a reliable HLA of complex system with a high level of integrity.

3.3 Conclusion

In this section, we have proposed a set of extensions on SysML to be aligned with Event-B refinement and decomposition mechanisms. These extensions allow to automatically translate SysML models of HLA into Event-B specifications. Two SysML parts are extended. The first part is about package diagrams which are customized to represent the decomposition of system/sub-systems hierarchies and the refinement of a system by its sub-systems interplay. The second part consists in customizing sequence diagrams with stereotypes applied on messages to refine the parent system behavior by the collaboration of its sub-systems processes. We have also defined a set of translation rules to translate SysML models into Event-B specifications in order to formally verify them using AtelierB.

Contributions about this section are published in [Bougacha et al., 2022b].

4 Requirements & high-level architecture alignment

The quality of a complex system is the main measure of its success, that depends on the degree to which it fulfills its requirements. To this end, alignment links between requirements models and HLA models need to be established. These semantic links can be the support to prove the compliance of HLA specification with the expression of system requirements. The third part of our work aims to propose a model-based approach to establish alignment links graphically between SysML/KAOS models and HLA models and then the translation of these links into Event-B formal specification and their formal verification (see Figure 2.18).

The process is composed of two steps:

— The first step consists in graphically modeling alignment links between leaf goals and sequence diagram messages. This graphical modeling step is a simple manner to

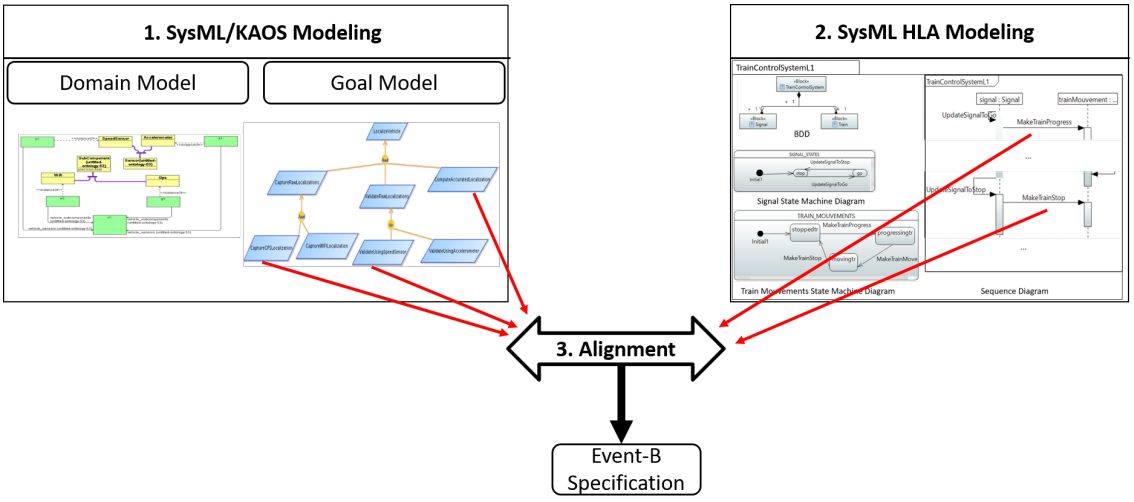

Figure 2.18 – SysML/KAOS models and SysML HLA models alignment approach

establish alignment links and to validate them by system stakeholders.

— The second step aims to formalize these alignment links into Event-B specification in order to be integrated in AtelierB for formal verification purposes. To perform this formalization, a set of translation rules is defined.

4.1 SysML/KAOS modeling and formal verification

As already introduced in 2.2.1, SysML/KAOS [Laleau et al., 2010, Gnaho et al., 2013b] is a RE method that allows the modeling of functional and non-functional requirements and the domain of a system. This approach aims to derive an Event-B specification directly from the SysML/KAOS models. Figure 2.19 presents an overview of the SysML/KOAS approach [Fotso et al., 2018a].

4.1.1 SysML/KAOS Modeling

As shown in this Figure 2.19, the first step of this process is to build goal models enriched with domain models. In our work, we are interested, for now, with functional goals knowing that a functional SysML/KAOS is called an "Achieve" goal. The functional goals hierarchy is built through a succession of refinements using the three types of refinement patterns **AND**, **OR** and **MILESTONE**.

Domain models are described by ontologies expressed using the SysML/KAOS domain modeling language [Tueno et al., 2017c, Fotso et al., 2018b], based on OWL [Sengupta and Hitzler, 2014] and PLIB [Pierra, 2006]. Each refinement level in the functional goal model is

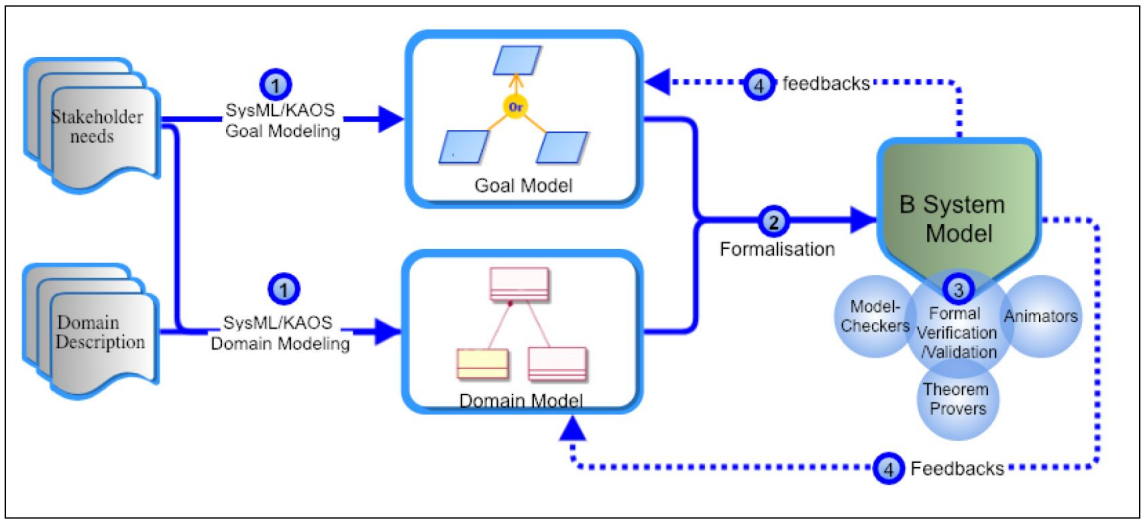

Figure 2.19 – SysML/KAOS specification process

enriched with a domain model. Domain models can be linked together to form a hierarchy. Concepts, as elements of domain model, designate collections of individuals with common properties.

SysML/KAOS **models example**

Figure 2.20 shows an excerpt from the SysML/KAOS goal model that represents the functional goals of the ATO over ERTMS case study. The main purpose is to move the train on GOA2 (goal MoveTrainOnGOA2). To achieve this purpose, the system must first, initiate the OnBoard system to be driven by ATO (goal MakeOnBoardForATODriving) and second, initiate the Track system that the Onboard is initiated to be driven by ATO (goal MakeTrackForATODriving). Two agents are defined: Onboard and Track, each of which is associated to a leaf goal that it is responsible for.

Figure 2.21 represents the SysML/KAOS domain model associated with the root level of the goal diagram (Figure 2.20). The ATOoETCS_GOA2 entity is modeled as a concept named ATOoETCS_GOA2System. The possible states of a ATOoETCS_GOA2 are modeled as an instances of attribute named ATOoETCS_GOA2SystemStates, which contains two instances of DataValue of type STRING: manualDriving and automaticDriving. atooetcs_goa2 is modeled as an instance of an individual named atooetcs_goa2 individual of ATOoETCS_GOA2System.

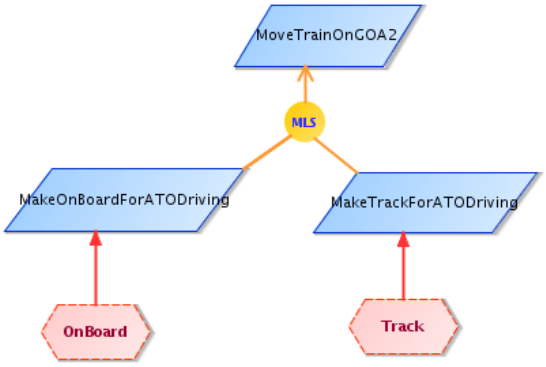

Figure 2.20 – Excerpt from the steam-boiler control system SysML/KAOS goal model

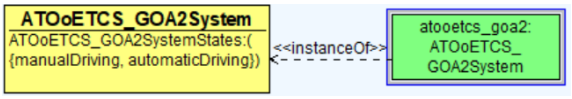

Figure 2.21 – Excerpt from the steam-boiler control system SysML/KAOS domain model

4.1.2 Event-B formalization of SysML/KAOS Models

The second step shown in Figure 2.19 is a B System [3] formalization of SysML/KAOS models. Goal models provide the behavioral part (events) of the specification while domain models provide its structural part (sets, constant and their properties, variables and their invariant) and the initialisation of state variables. It remains to manually specify the body of events and to formally verify and validate the specification with AtelierB tools.

As the semantics of the refinement between goals is different from that of the refinement between Event-B components, proof obligations for goal refinement are generated as presented in 2.2.2. They depend on the goal refinement operator used and complete the Event-B proof obligations for invariant preservation and for event feasibility.

Event-B formalization of SysML/KAOS models example

For the Event-B formalization of SysML/KAOS models, we recall that each goal model level is represented as a machine in the Event-B specification. Listing 4 represents the root level of the Event-B specification of the ATO over ETCS GOA2 system. The concept ATOoETCS_GOA2System, of the domain model, gives a set and its individual atooetcs_goa2 gives a constant typed with a property as an element of set ATOoETCS_GOA2System and a variable atooetcs_goa2State gives a total function from ATOoETCS_GOA2System to ATOoETCS_GOA2SystemStates and initialised in the INITIALISATION clause. At this level, event MoveTrainOnGOA2 allows to change the driving from manual to automatic.

Each refinement level of the Event-B specification is the result of the translation of

3. B System designates a syntactic variant of Event-B offered within the AtelierB tool.

goal model refinement level. Listing 5 represents an excerpt of the EVENT-B formalization of the first refinement level of the goal diagram presented in Figure 2.20 with its associated domain models. This level defines an EVENT-B component containing 3 variables, 3 invariants and 2 events MakeOnBoardForATODriving, MakeTrackForATODriving. These events MILESTONE REFINE the parent goal MoveTrainOnGOA2. Therefore, the EVENT-B formalization of this refinement pattern is produced using the keyword ref_milestone.

```
SYSTEM ATOoETCS_GOA2System
SETS ATOoETCS_GOA2System, ATOoETCS_GOA2SystemStates
CONSTANTS atooetcs_goa2, manualDriving, automaticDriving
PROPERTIES
    ...
    atooetcs_goa2 ∈ ATOoETCS_GOA2System ∧
    ATOoETCS_GOA2SystemStates = {automaticDriving, manualdriving}
VARIABLES atooetcs_goa2State
INVARIANT
    atooetcs_goa2State ∈ ATOoETCS_GOA2System →
    ATOoETCS_GOA2SystemStates
INITIALISATION
    atooetcs_goa2State : ∈ {atooetcs_goa2} →
    ATOoETCS_GOA2SystemStates
EVENTS
    MoveTrainOnGOA2 =
    SELECT atooetcs_goa2State(atooetcs_goa2)= manualdriving THEN
    atooetcs_goa2State(atooetcs_goa2):= automaticDriving
    END
END
```

Listing 4 – EVENT-B formalization of the SYSML/KAOS models example root level presented in Figures 2.20 and 2.21.

```
SYSTEM ATOoETCS_GOA2System2
REFINES ATOoETCS_GOA2System
    ...
VARIABLES atooetcs_goa2State, onboardState,     trackState,
INVARIANT
    ...
```

```
INITIALISATION
    ...
EVENTS
    MakeOnBoardForATODriving ref_milestone  MoveTrainOnGOA2 =
    ...
    END;
    MakeTrackForATODriving ref_milestone  MoveTrainOnGOA2 =
    ...
    END
END
```

Listing 5 – An excerpt of the Event-B formalization of the SysML/KAOS models example first refinement level presented in Figures 2.20.

4.2 Graphical alignment

To process the first step, we note that the requirements side is represented by a SysML/KAOS goal model enriched with a domain model for each goal model level and the architecture side is represented by a SysML HLA model.

To specify graphical alignment links between these two sides, we have chosen to establish links between concepts from each side:

— leaf goals from SysML/KAOS goal model because they are the most concrete goals of a goal model. A leaf goal is assigned to an agent (environment, software or sub-system agent) responsible for the goal satisfaction.

— Sequence diagram messages while they represent running interactions between system components. Each message corresponds to a transition in the state-machine of the block associated to the target component.

We propose three kinds of alignment links to satisfy leaf goals assigned to a sub-system (an agent).

— The first alignment kind is called *Satisfy*. It is defined to represent an alignment link when one message can satisfy one goal.

— The second kind is called *And_Satisfy*. It is defined when a goal is satisfied by a set of messages, i.e. the execution of all of them, in any order, is needed to satisfy the

goal.

— The last alignment kind is called *Milestone_ Satisfy*. It is defined when a sequential execution of a set of messages in a specific order is needed to satisfy a goal.

These alignment links definitions are inspired from two concepts:

— First, the SysML requirement diagram *Satisfy* relationship, a dependency between a requirement and a model element that fulfills the requirement. This relationship direction points from the satisfying (client) model element to the (supplier) requirement that is satisfied.

— Second, The SysML/KAOS refinement operators *AND, OR* and *MILESTONE* which represents the goal model hierarchy where higher-level goals can be refined into lower-level sub-goals and the set of sub-goals satisfy the higher-level goal. We recall that, when a goal is AND-refined into sub-goals, all of them must be satisfied for the parent goal to be satisfied. The MILESTONE refinement consists in identifying milestone states that must be sequentially satisfied for the parent goal to be satisfied. Finally, when a goal is OR-refined, the satisfaction of one its sub-goals is sufficient for the satisfaction of the parent goal.

Therefore, to create alignment links, we have merged these two concepts to adapt the SysML *Satisfy* relationship between requirements and model elements and enrich it with the goal/sub-goals satisfaction semantics.

Figure 2.22 shows the alignment meta-model, in which these three alignment kinds are represented as subclasses of the meta-class *Dependency*. Two binary associations are then defined. The first one is to link a leaf goal to a dependency and the second is to link a dependency to a set of messages of a sequence diagram. Note that a message can be linked to several dependencies. The kind of alignment link is specified by a stereotype on the arrows between messages and leaf goals. Let us note that, for a *Milestone_ Satisfy* alignment link, the order of the messages must be given by numbering the relevant arrows.

Graphical alignment example

To give a simple representation of these graphical alignment links, we have applied the proposed alignment process on example from the ATO over ERTMS case study excerpt. Shown in Figure 2.23, the right hand model presents the SysML/KAOS goal model of the ATO over ETCS GOA2 whereas the left hand model presents its associated HLA package taken from HLA models [Bougacha et al., 2022a]. The SysML/KAOS goal model contains

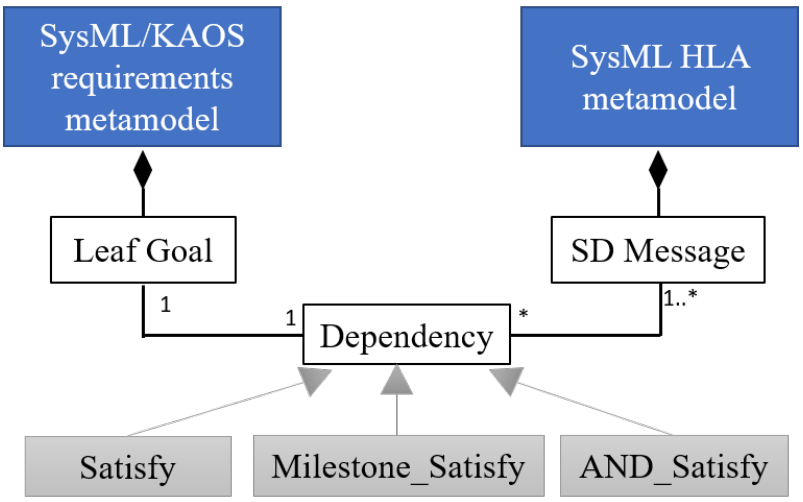

Figure 2.22 – Alignment meta-model

two leaf goals MakeOnBoardForATODriving and MakeTrackForATODriving. These two leaf goals are satisfied respectively by the sequence diagram messages SetBoardforAutomatic and SetTrackforAutomatic. Then, two alignment links are created of kind *Satisfy*, from the sequence diagram messages to the leaf goals to represent that the satisfaction of one leaf goal is established with the related sequence diagram message.

More details and illustrations about the graphical alignment modeling will be presented in Section 5.4.

4.3 Formalization of graphical alignment links

To formalize graphical alignment links using Event-B, a set of translation rules has been defined.

Recall that the Event-B formalization of the SysML/KAOS models is described in [Matoussi et al., 2011a, Tueno Fotso et al., 2018] and the Event-B formalization of the SysML HLA models is presented in 3.2.1.2. Each goal, including leaf goals, is transformed to an event in the Event-B specification of requirements models and each sequence diagram message is transformed to an event in the HLA model Event-B specification.

The main idea is to define an alignment link between a set of messages and a leaf goal as a refinement relationship between the corresponding Event-B events. There are many reasons why it is not possible to use the standard Event-B refinement namely:

— The semantics of the Event-B refinement do not correspond to our alignment semantics. Indeed, Event-B refinement process allows to gradually enrich the differ-

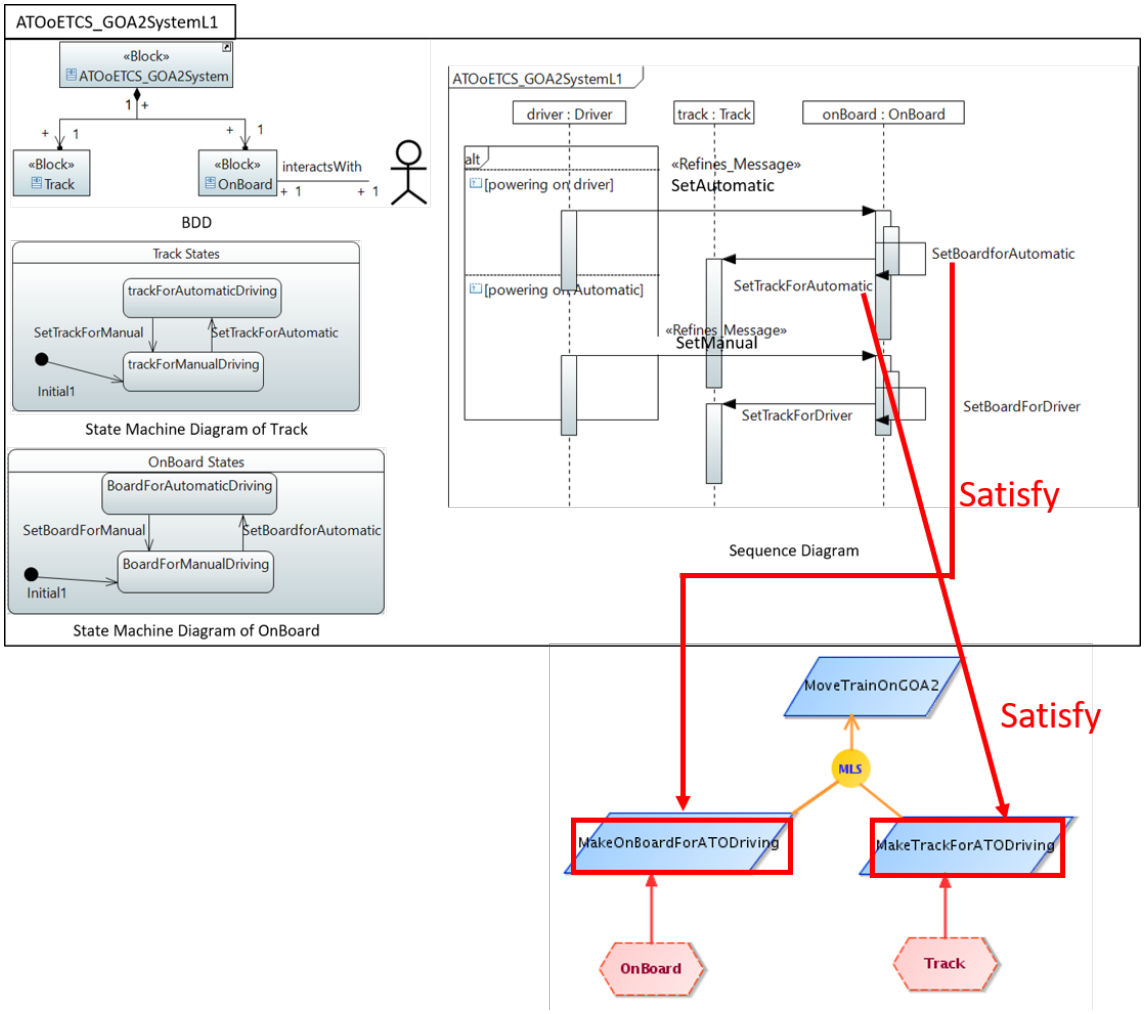

Figure 2.23 – Graphical alignment example

ent parts that constitute the system with more functional, safety, etc details, starting from an abstract model to a more concrete one.

— The messages that satisfy a leaf goal can belong to distinct Event-B machines whereas in Event-B it is not possible to specify that a set of machines refines one machine.

However, as the formalization of SysML/KAOS models and SysML HLA models is carried out in Event-B, we think that it would be more appropriate to formalize alignment with Event-B. The proposed solution is to build a new Event-B machine for each alignment link. New sets of refinement proof obligations are specified, one for each kind of alignment. Discharging these proof obligations allows to formally verify the satisfaction of a leaf goal by a set of HLA messages.

Figure 2.24 illustrates how this new machine is built for a leaf goal *LG* which is

And_Satisfy by the HLA sequence diagram messages M_1 and M_2:

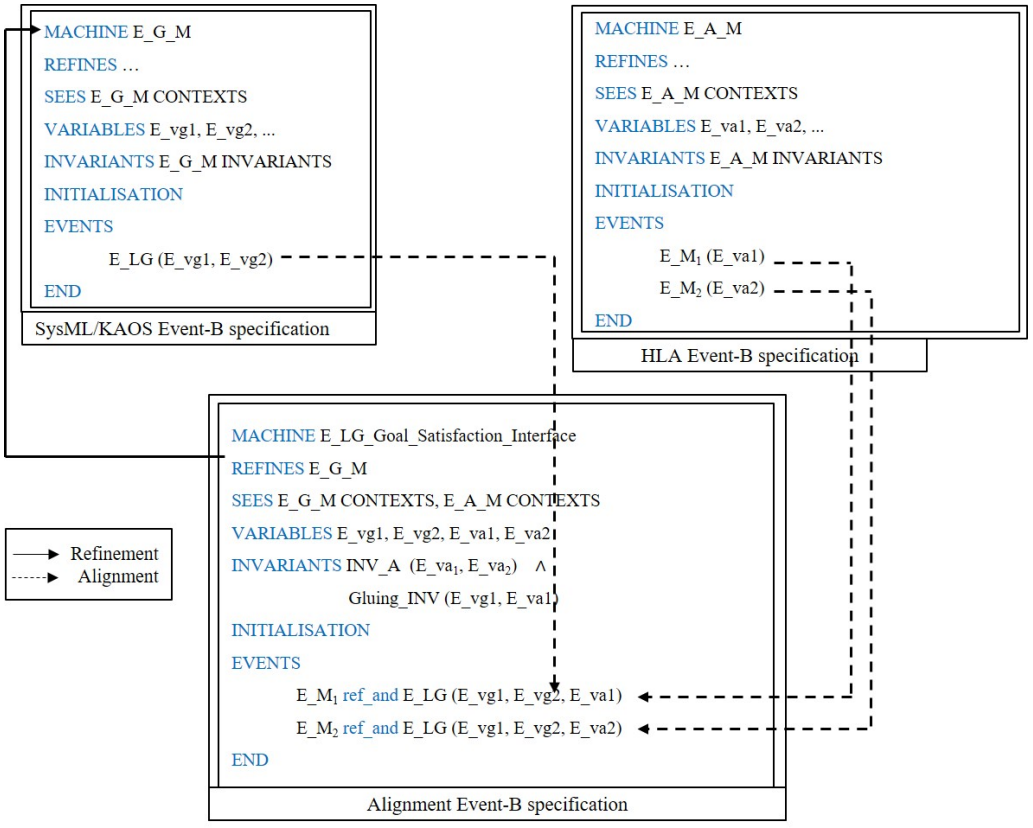

Figure 2.24 – EVENT-B architecture of the proposed alignment

— The top left machine, called *E_G_M*, is the SysML/KAOS EVENT-B machine that contains the event *E_LG* corresponding to the leaf goal *LG*.

— The top right machine, called *E_A_M*, is the HLA EVENT-B machine that contains the events E_M_1 and E_M_2 corresponding to the messages M_1 and M_2. In this example, let us assume that M_1 and M_2 belong to the same machine.

— the new machine, called *E_LG_Goal_Satisfaction_Interface*, is the alignment machine. This machine refines *E_G_M* and imports the events E_M_1 and E_M_2 from the HLA machine *E_A_M*.

Table 2.4 presents the first four translation rules that allows to obtain the elements of the EVENT-B alignment machines. The two columns of *Source concepts* describe the elements related to the SysML/KAOS models and the SysML HLA models needed for defining alignment links. The two columns of *Target concepts* describe the elements of the EVENT-B alignment machine obtained from translating the *Source concepts*.

Rules 2, 3 and 4 need to be more precisely defined.

		Source concepts		Target concepts	
Rule	Translation of	Element	Constraint	Element	Constraint
1	Leaf goal to satisfy	LG E_LG	LG is the leaf goal E_LG is the event related to LG E_G_M is the Event-B machine that contains E_LG	E_LG_Goal_ Satisfac- tion_Interface	E_LG_Goal_Satisfaction_Interface ∈ MACHINE; E_LG_Goal_Satisfaction_Interface REFINES E_G_M; E_LG_Goal_Satisfaction_Interface SEES E_G_M CONTEXTS;
2	Variables involved in E_LG	E_vg_i, i ∈ [1..n]	E_vg_i are the variables of E_G_M involved in E_LG	E_vg_i, i ∈ [1..n]	E_vg_i, ∈ VARIABLES
3	Messages responsible for the satisfaction of the leaf goal LG	M_j E_M_j j ∈ [1..p]	M_1, ... M_p are messages E_M_j is the event related to message M_j; $E\_A\_M_j$ is the Event-B machine that contains E_M_j	E_M_j j ∈ [1..p]	E_M_j ∈ EVENTS E_LG_Goal_Satisfaction_Interface SEES $E\_A\_M_j$ CONTEXTS
4	Variables involved in E_M_j events	$E\_va_{j,k}$ j ∈ [1..p] k ∈ [1..q]	$E\_va_{j,k}$ are the variables involved in E_M_j $INV\_A\_M_j(E\_va_{j,1}$, ...,$E\_va_{j,q})$ is the part of $E\_A\_M_j$ invariant related to the $E\_va_{j,k}$ variables	$E\_va_{j,k}$, j ∈ [1..p] k ∈ [1..q]	$E\_va_{j,k}$ ∈ VARIABLES $INV\_A\_M_j(E\_va_{j,1}$, ..., $E\_va_{j,q})$ ∈ INVARIANTS

Table 2.4 – First four translation rules for alignment links

— **Rule 2**. This rule aims at copying the variables E_vg_i (i ∈ [1..n]), involved in the event E_LG corresponding to the leaf goal LG to satisfy from the machine E_G_M, in the alignment machine of LG called $E\_LG\_Goal\_Satisfaction\_Interface$.

— **Rule 3**. M_j (j ∈ [1..p]) are the messages of the HLA model responsible for the satisfaction of the leaf goal LG. E_M_j (j ∈ [1..p]) are the corresponding Event-B events. These messages can belong to several sequence diagrams, consequently the corresponding events can belong to several Event-B machines $E\_A\_M_j$. Each E_M_j is copied from $E\_A\_M_j$ in the $E\_LG\_Goal\_Satisfaction\_Interface$ and this machine will also SEES $E\_A\_M_j$ CONTEXTS.

— **Rule 4**. Each event E_M_j (j ∈ [1..p]) uses a set of variables $E\_va_{j,k}$ (k ∈ [1..q]) coming from the HLA model. Rule 4 aims at copying these variables in the alignment machine $E\_LG\_Goal\_Satisfaction\_Interface$. $INV\_A\_M_j(E\_va_{j,1}$, ...,$E\_va_{j,q})$, the part of $E\_A\_M_j$ invariant related to the $E\_va_{j,k}$ variables, is also copied in the machine $E\_LG\_Goal\_Satisfaction\_Interface$.

Rule 5 consists in constructing the gluing invariant in the machine $E\_LG\_Goal\_Satisfaction\_Interface$. These invariants link variables of the HLA Event-B machines $E\_A\_M_j$ to variables of the SysML/KAOS Event-B machine E_G_M. Recall that, as presented in [Tueno Fotso et al., 2018], each leaf goal LG is associated to one or more elements of

the domain model. They give the structural part (CONTEXTS, VARIABLES and INVARIANTS) of the EVENT-B machine E_G_M. Likewise, in the HLA model translation presented in 3.2.1.2, the source and the target lifelines of a sequence diagram message are associated to blocks in the BDD. They give the VARIABLES and INVARIANTS part of the EVENT-B machines $E\_A\_M_j$. Thus, the gluing invariant comes to establish links between elements of the domain model and elements of the BDD. This can only be a creative activity from the designer that cannot be automatized.

To formalize the semantics of the proposed alignment links between goals and messages, we propose a set of new refinement proof obligations, different from the proof obligations generated for the EVENT-B standard refinement. They will depend on the satisfaction relationship used. These proof obligations are inspired by the proof obligations defined for the formalization of refinement links between goals [Matoussi et al., 2011a] to establish that a set of sub-goals satisfies a parent goal. We have adapted them to establish that one or more messages can satisfy a goal.

Let LG be a leaf goal and M_1, M_2 two messages. Let E_LG be the event associated to LG and E_M_1, E_M_2 the two events associated to M_1 and M_2. Each event E is of the form: $E =$ **SELECT** E_Guard **THEN** E_Post.

Rule 6 is composed of three sub-rules that depend on the used satisfaction relationship.

— **Rule 6.1**: *Satisfy* relationship. Define that LG is satisfied by M_1, then:

$$E\_M_1 \text{ ref } E\_LG$$

where ref is the standard EVENT-B refinement.

This proof obligation ensures that the satisfaction of E_LG depends on the execution of E_M_1.

— **Rule 6.2**: *And_Satisfy* relationship. Assume that LG is satisfied by M_1 and M_2, then:

$$E\_M_1 \text{ ref\_and } E\_LG$$
$$E\_M_2 \text{ ref\_and } E\_LG$$

New proof obligations are generated:

- $E\_M_1\_Guard \Rightarrow E\_LG\_Guard$
- $E\_M_2\_Guard \Rightarrow E\_LG\_Guard$
- $(E\_M_1\_Post \wedge E\_M_2\_Post) \Rightarrow E\_LG\_Post$

These proof obligations ensure firstly that $E\_M_1\_Guard$ and $E\_M_2\_Guard$ should

never contradict E_LG_Guard and secondly the execution of E_M_1 and E_M_2 without any specific order implies the satisfaction of E_LG.

— **Rule 6.3**: *Milestone_Satisfy* relationship. Assume that LG is satisfied by the sequential execution of M_1 and M_2, then:

$$E\_M_1 \text{ ref\_milestone } E\_LG$$

$$E\_M_2 \text{ ref\_milestone } E\_LG$$

New proof obligations are generated:

— $E\_M_1\_Guard \Rightarrow E\_LG\_Guard$

— $E\_M_1\_Post \Rightarrow E\_M_2\_Guard$

— $E\_M_2\_Post \Rightarrow E\_LG\_Post$

These proof obligations ensure firstly that $E\_M_1\_Guard$ should never contradict E_LG_Guard. Secondly the scheduling constraint should be respected with $E\_M_1\_Post$ implies $E\_M_2\_Guard$ and finally the execution of E_M_1 followed by the execution of E_M_2 implies the satisfaction of E_LG

Note that these new proof obligations can be automatically generated by the proof obligation generator of AtelierB.

An advantage of our approach is that we can use all the support tools of AtelierB to discharge proof obligations, but also its animator to simulate the specification, and then to validate and verify it.

Formalization of graphical alignment example

To formalize the alignment links presented in Figure 2.23, we present an application example of the proposed formalization rule. Figure 2.25 shows the alignment of the events MakeOnBoardForATODriving and MakeTrackForATODriving respectively by the sequence diagram messages SetBoardForAutomatic and SetTrackForAutomatic. In this example, we are interested to formalise the alignment of the leaf goal MakeOnBoardForATODriving into Event-B. To aim, this formalisation, we give the Listing 6 which gives the Event-B machine that contains the leaf goal OnBoard_Interface to satisfy and Listing 7 which gives the Event-B machine that contains the message from HLA that satisfies the leaf goal called ATOoETCS_GOA2System2. Listing 8 shows the generated Event-B specification of the alignment through the application of rules described in Section 4.3 and presented in Table 2.5.

Therefore, we give the formalisation process of this example as follows:

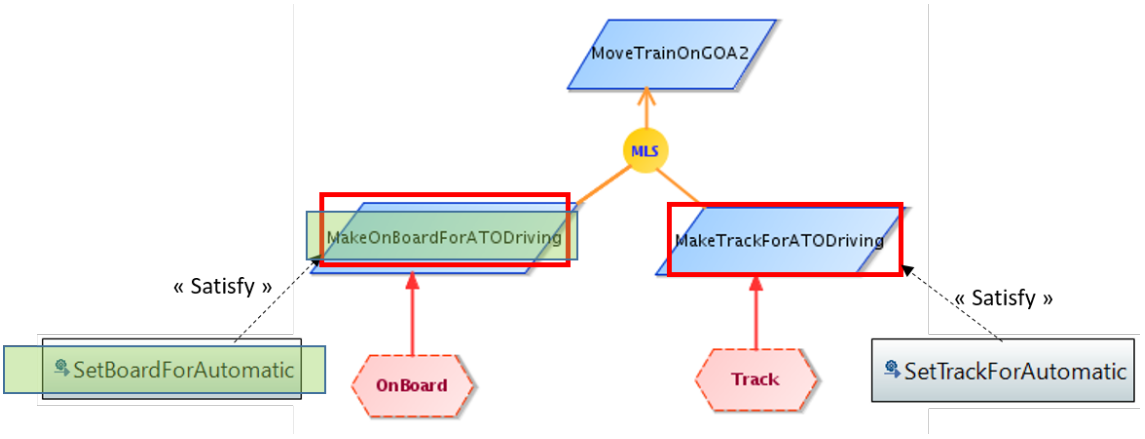

Figure 2.25 – EVENT-B architecture of the proposed alignment

		Source concepts		Target concepts	
Rule	Translation of	Element	Constraint	Element	Constraint
1	Leaf goal to satisfy	LG E_LG	LG is the leaf goal E_LG is the event related to LG E_G_M is the EVENT-B machine that contains E_LG	E_LG_Goal_Satisfac-tion_Interface	E_LG_Goal_Satisfaction_Interface ∈ MACHINE; E_LG_Goal_Satisfaction_Interface REFINES E_G_M; E_LG_Goal_Satisfaction_Interface SEES E_G_M CONTEXTS;
2	Variables involved in E_LG	E_vg_i, $i \in [1..n]$	E_vg_i are the variables of E_G_M involved in E_LG	E_vg_i, $i \in [1..n]$	$E\_vg_i \in$ VARIABLES
3	Messages responsible for the satisfaction of the leaf goal LG	M_j E_M_j $j \in [1..p]$	$M_1, \ldots M_p$ are messages E_M_j is the event related to message M_j; $E\_A\_M_j$ is the EVENT-B machine that contains E_M_j	E_M_j $j \in [1..p]$	$E\_M_j \in$ EVENTS E_LG_Goal_Satisfaction_Interface SEES $E\_A\_M_j$ CONTEXTS
4	Variables involved in E_M_j events	$E\_va_{j,k}$ $j \in [1..p]$ $k \in [1..q]$	$E\_va_{j,k}$ are the variables involved in E_M_j $INV\_A\_M_j(E\_va_{j,1}, \ldots, E\_va_{j,q})$ is the part of $E\_A\_M_j$ invariant related to the $E\_va_{j,k}$ variables	$E\_va_{j,k}$, $j \in [1..p]$ $k \in [1..q]$	$E\_va_{j,k} \in$ VARIABLES $INV\_A\_M_j(E\_va_{j,1}, \ldots, E\_va_{j,q}) \in$ INVARIANTS

Table 2.5 – Alignment Formalisation rules application table

— First, a new Event-B machine called MakeOnBoardForATODriving_Goal_Satisfaction_Interface from the leaf goal MakeOnBoardForATODriving (framed in green in Figure 2.25) is created. It refines OnBoard_Interface and gets all its seen context. (rule 1 in Table 2.5).

— Variables involved in the event MakeOnBoardForATODriving to satisfy which are onboardState are copied in MakeOnBoardForATODriving_Goal_Satisfaction_Interface. (rule 2 in Table 2.5).

— The goal MakeOnBoardForATODriving is *Satisfy* with the message SetBoardForAutomatic (framed in green in Figure 2.25), then the corresponding event is copied from the HLA EVENT-B specification machine ATOoETCS_GOA2System2 with the involved variable onboardHLAState and its associated typing invariant (rule 3 and 4 in Table 2.5).

— The gluing invariant must be manually defined where the variables onboard, from

the goal machine Onboard_Interface, and onboardHLAState, from the HLA machine ATOoETCS_GOA2System2, are linked since they represent the same entities but with a different viewpoint (goal or architecture). (rule 5 presented in Section 4.3).

— Event SetBoardForAutomatic refines with a ref keyword the event MakeOnBoardForATODriving (Rule 6.1 presented in Section 4.3).

```
SYSTEM OnBoard_Interface
SEES ATOoETCS_GOA2SystemL1_CONT, ATOoETCS_GOA2SystemL0_CONT
ABSTRACT_VARIABLES onboardState,
...
EVENTS
    MakeOnBoardForATODriving ref_milestone MoveTrainOnGOA2 =
    ...
    END;
    ...
END
```

Listing 6 – Extract from SysML/KAOS OnBoard Interface.

```
REFINEMENT ATOoETCS_GOA2System2
REFINES ATOoETCS_GOA2System
SEES ATOoETCS_GOA2SystemL1HLA_CONT, ATOoETCS_GOA2SystemL0HLA_CONT
VARIABLES atooetcs_goa2State, onboardHLAState, trackState,
...
EVENTS
    SetBoardForAutomatic =
    SELECT onboardState(onboard)=BoardForManualDriving THEN
        onboardState(onboard):=BoardForAutomaticDriving
    END;
...
END
```

Listing 7 – Extract from HLA ATOoETCS_GOA2System2

```
REFINEMENT MakeOnBoardForATODriving_Goal_Satisfaction_Interface
REFINES OnBoard_Interface
```

```
SEES ATOoETCS_GOA2SystemL1_CONT, ATOoETCS_GOA2SystemL0_CONT,
    ATOoETCS_GOA2SystemL1HLA_CONT, ATOoETCS_GOA2SystemL0HLA_CONT
ABSTRACT_VARIABLES onboardState, onboardHLAState
INVARIANT
    onboardState ∈ OnBoard → OnBoardStates
    /****From goal specification****/ ∧
    OnBoardHLAStates ∈ OnBoardStates → OnBoardStates
    /****From HLA specification****/ ∧
    OnBoardHLAStates[OnBoard]=BoardForAutomaticDriving ⇒
    onboardState [OnBoard]=BoardForAutomaticDriving
    //gluing Invariant. ∧
    OnBoardHLAStates[OnBoard]=BoardForManualDriving ⇒
    onboardState [OnBoard]=BoardForManualDriving//gluing Invariant.
INITIALISATION
    OnBoardHLAStates, OnBoardStates:(
    onboardState ∈ OnBoard → OnBoardStates
    /****From goal specification****/ ∧
    OnBoardHLAStates ∈ OnBoardStates → OnBoardStates
    /****From HLA specification****/ ∧
    OnBoardHLAStates[OnBoard]=BoardForAutomaticDriving ⇒
    onboardState [OnBoard]=BoardForAutomaticDriving
    //gluing Invariant. ∧
    OnBoardHLAStates[OnBoard]=BoardForManualDriving ⇒
    onboardState [OnBoard]=BoardForManualDriving//gluing Invariant.
EVENTS
    SetBoardForAutomatic ref MakeOnBoardForATODriving =
    SELECT onboardState(onboard)=BoardForManualDriving THEN
        onboardState(onboard):=BoardForAutomaticDriving
    END
END
```

Listing 8 – EVENT-B formalization of the goal MakeOnBoardForATODriving alignment link.

More details and illustrations about the graphical alignment modeling and formalization will be presented in Section 5.4.

4.4 Conclusion

In this section, we have proposed a model-based approach to align complex systems HLA models with SysML/KAOS requirements models. This approach is twofold. First, graphical which allows to specify the alignment of a leaf goal with HLA elements responsible for its satisfaction. For this purpose, three kinds of alignment links are defined and a new alignment meta-model is proposed. Second, formal which consists in formalizing the alignment links in Event-B in order to be verified. To produce this Event-B specification, we have proposed a set of translation rules. The semantics of the alignments links are given by these rules and new proof obligations were defined that can be discharged using AtelierB.

The contributions behind this alignment work were published in [Bougacha et al., 2023].

5 Illustration of the methodology on a case study

In this section, we present an illustration of the proposed methodology on a case study. This illustration will give a better understanding and explanation of the methodology based on examples and allows to evaluate its strengths and weaknesses with regard to existing methods. It also allows to test the proposed alignment between the requirements models and the HLA models To this end, we illustrate this methodology on an excerpt of the landing gear system case study [Boniol and Wiels, 2014].

5.1 Landing Gear System case study

The landing gear system case study [Boniol and Wiels, 2014] was proposed in the 4th edition of the ABZ conference (ASM, Alloy, B, TLA, VDM, Z). The goal of this case study is to specify the system responsible for extending and retracting the landing gear of an aircraft. The landing system is in charge of maneuvering landing gears and associated doors. This system is composed of 3 landing sets: front, left and right. Each landing set contains a door, a landing-gear and associated hydraulic cylinders.

The landing sequence proceeds as follows: open the doors of the landing gear boxes, extend the landing gears and close the doors. After taking off, the retraction sequence to be performed is: open the doors, retract the landing gears and close the doors.

The landing gear system architecture is composed of three parts:

— The pilot interface. It is used to command the retraction and outgoing of gears. An Up/Down handle is provided to the pilot. When the handle is switched to Up the retracting landing gear sequence is executed, when the handle is switched to Down the extending landing gear sequence is executed.

— The mechanical part. It is composed of three landing sets: front, left and right sets. Each set is composed of a landing gear uplock box, and a door with two latching boxes in the closed position. The landing gears and doors motion is performed by a set of actuating cylinders. The cylinder position corresponds to the door or landing gear position (when a door is open, the corresponding cylinder is extended). These cylinders perform using hydraulic power provided by a set of electro-valves.

— The digital part. It is composed of two identical computing modules. Each one executes in parallel the same control software. This software is in charge of controlling gears and doors.

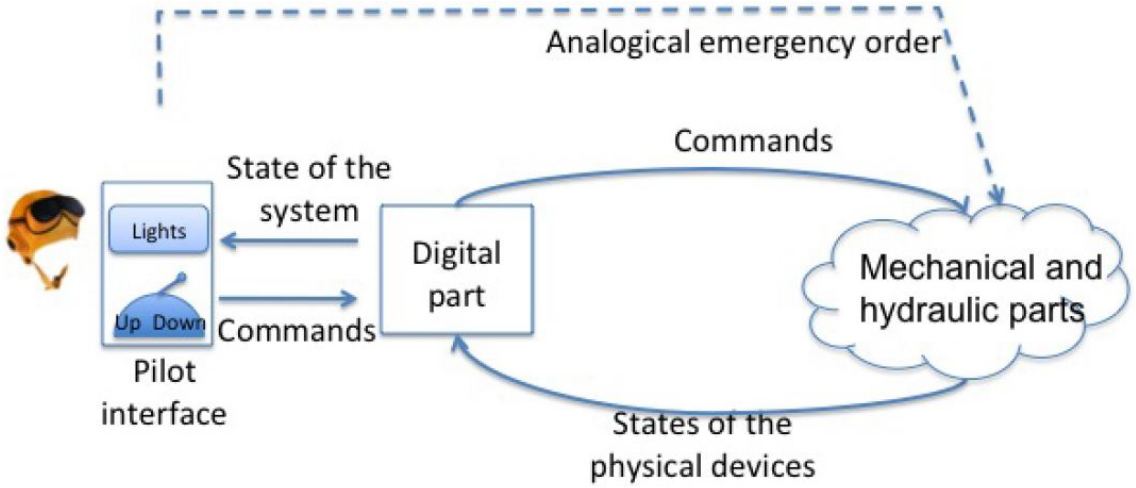

Figure 2.26 – Landing gear system architecture

Figure 2.26 shows the architecture of the landing gear system and the constituting parts which collaborate together to extend or retract the landing gears.

5.2 SysML/KAOS modeling and formalizing of the landing gear system case study

The SysML/KAOS method allows the progressive construction of system requirements from the refinement of stakeholder needs. We have chosen to consider that the general objective that the system must fulfil is to control the movements of the landing gear. The

most abstract level of the formal specification that has been built is a translation of this general objective. Its concrete refinement levels are representations of the choices allowing the achievement of the objective named maneuverLandingGear. The specification includes four refinement levels explicitly related to stakeholder needs through SysML/KAOS models. The SysML/KAOS method allows to trace the source and justify the need for each formal component and its contents. The formal specification is devoted to the formalisation of system functional goals and in their verification with regard to domain properties and safety invariants. The environment behavior is left nondeterministic with respect to domain constraints modeled in SysML/KAOS domain models.

5.2.1 SysML/KAOS Goal modeling

Figure 2.27 presents the SysML/KAOS functional goal diagram focused on the main system purpose which is to control the movements of the landing gear (maneuverLandingGear). To achieve it:

— As already presented the system ensures the extending (maneuverLGforLanding) and retracting (maneuverLGforTakingOff) of the landing gear.

— The second refinement level of the SysML/KAOS goal diagram focuses on the extension and retraction processes. The extension process starts when the decision to extend is stemming (decideToExtend). Afterthat, the landing gears are extended (makeLGExtended). Same for the retraction process which starts by making the decision (decideToTakeOff) and retracts the landing gears (makeLGRetracted).

— The decision to extend (respectively to retract) is triggered when the pilot switch the handle to Down (respectively to Up) (putHandleDown (respectively putHandleUp)). Then, an extension (respectively retraction) order is communicated to the digital entity (communicateExtensionOrder (respectively communicateRetractionOrder)).

— The landing and retraction sequences proceed as follows: stimulate the general electro-valve (StimulateGeneral_Ev), open the doors of the landing gear boxes (makeDoorsOpen, extend/retract the landing gears (makeGearsExtend/ makeGearsRetract) and close the doors (makeDoorsClose) and finally stop the stimulation of the general electro-valve (StopGeneral_EvStimulation).

— To open the doors, first the door opening electro-valve should be stimulated (stimulateDoorOpeningEv) then doors will be opened (OpenDoors).

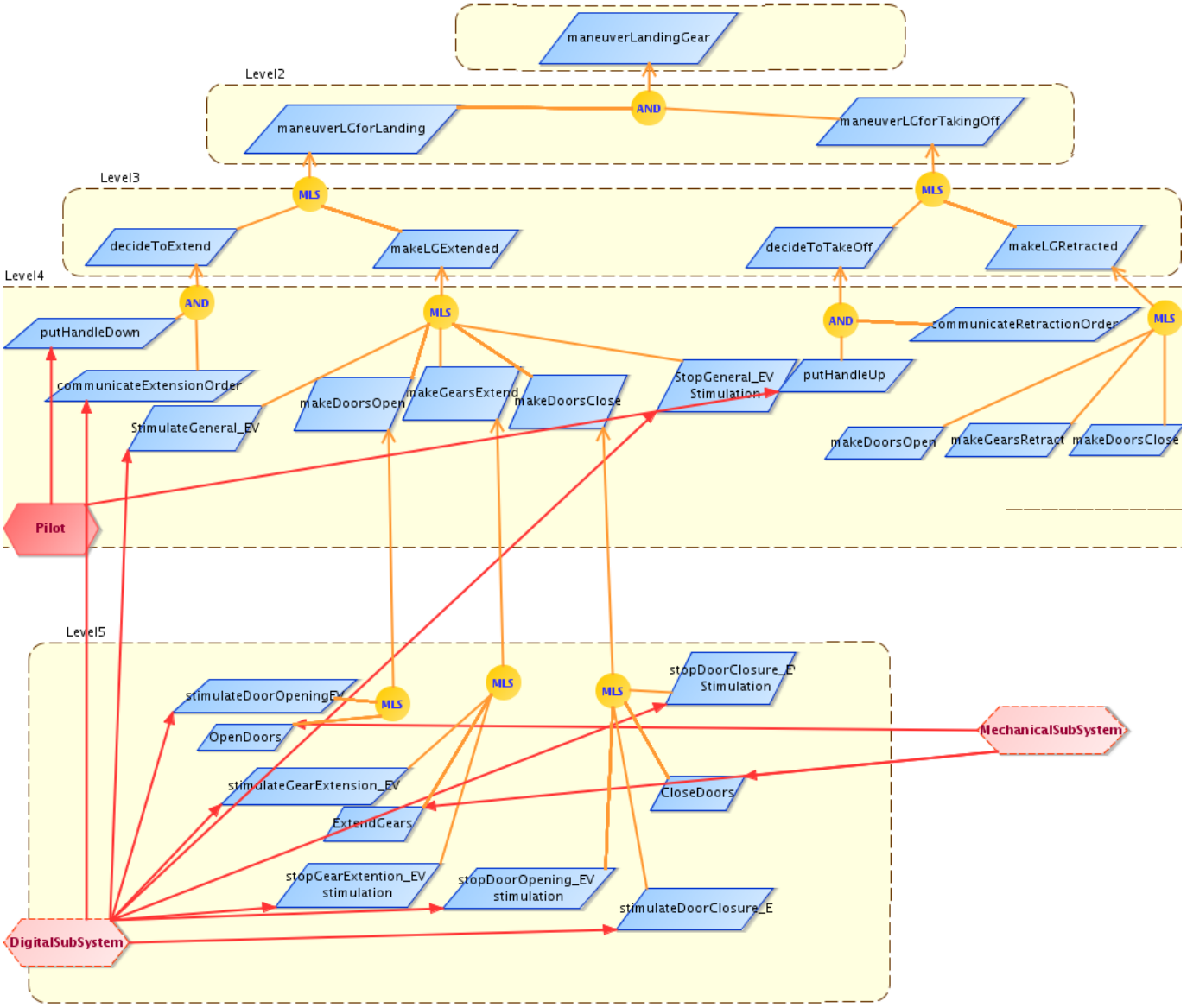

Figure 2.27 – The landing gear system SysML/KAOS goal diagram

— Afterthat, to extend/retract the gears, gear extension/retraction electro-valve should be stimulated (stimulateGearExtension_EV) and then the gears will be extended/retracted and finally gear extension/retraction electro-valve stimulation should be stopped (stopGearExtension_ EVstimulation.

— To close the doors, door opening electro-valve should be stopped (stopDoorOpening _EVstimulation), stimulate the door closure electo-valve (stimulateDoorClosure_EV), then doors should be closed (CloseDoors) and finally door closure electro-valve stimulation is stopped (stopDoorClosure_EVStimulation).

5.2.2 SysML/KAOS Goal model formalization

From the goal model, we distinguish five levels which are translated into five Event-B. The hierarchy of this Event-B specification goes from the abstract goal to the concrete ones called leaf goals.

5.2.2.1 The Root Level

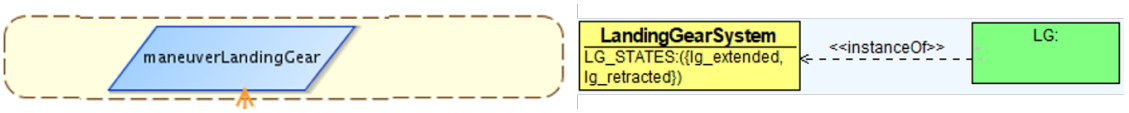

Figure 2.28 – SysML/KAOS root level goal model and corresponding domain model

```
SYSTEM LandingGearSystem_CONTEXT
SETS LandingGearSystem ;
    LG_STATES
CONSTANTS LG, lg_extended ,
    lg_retracted
PROPERTIES
    LG ∈ LandingGearSystem
    ∧ LandingGearSystem = { LG }
    ∧ lg_extended ∈ LG_STATES
    ∧ lg_retracted ∈ LG_STATES
    ∧ LG_STATES={lg_extended ,
    lg_retracted}
    ∧ lg_extended ≠ lg_retracted
END
```

Listing 9 – The root level of Figure 2.28 Event-B specification context.

```
SYSTEM Landing_Gear_System
SEES LandingGearSystem_CONTEXT
ABSTRACT_VARIABLES lgState
INVARIANT
    lgState ∈ LandingGearSystem
    → LG_STATES
INITIALISATION
    lgState:∈ LandingGearSystem
    → LG_STATES
EVENTS
    maneuverLandingGear =
    ANY state  WHERE
    state ∈ LG_STATES THEN
        lgState ( LG ) := state
    END
END
```

Listing 10 – The root level of Figure 2.28 Event-B specification machine.

Figure 2.28 represents the root level of the landing gear system goal model and its associated domain model. The landing gear system entity is modeled as an instance of concept

named LandingGearSystem. The possible states of a landing gear system are modeled as an instances of attribute named LG_STATES, which contains two instances of DataValue of type STRING: lg_extended for the extended state and lg_retracted for the retracted state. LG is modeled as an instance of an individual named LG individual of LandingGearSystem.

The EVENT-B translation of the root level of the goal diagram and the associated domain model presented in Figure 2.28 is shown in Listings 9 and 10. The domain model gives rise to sets, constants, properties, variables and invariants of the formal specification. The root goal is translated into an event for which the body has been manually specified: the control of the landing gear movement states between lg_extended and lg_retracted.

5.2.2.2 The First Refinement Level

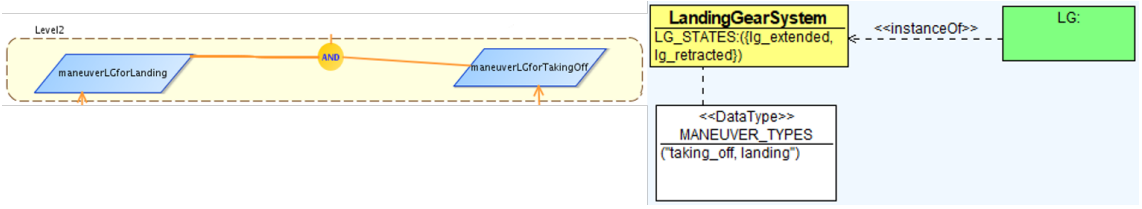

Figure 2.29 – SYSML/KAOS first refinement level goal model and corresponding domain model

```
SYSTEM
    LandingGearSystem_CONTEXT_L1
SETS MANEUVER_TYPES
CONSTANTS taking_off , landing
PROPERTIES
    taking_off ∈ MANEUVER_TYPES
    ∧ landing ∈ MANEUVER_TYPES
    ∧ MANEUVER_TYPES={taking_off , landing}
    ∧ taking_off ≠ landing
END
```

Listing 11 – The first refinement level in Figure 2.29 EVENT-B specification context.

The first refinement level of the goal model and its associated domain model is shown in Figure 2.29. This level refines the root level by the possible maneuvers the landing gear could perform to achieve the root goal. More precisely, goals maneuverLGforLanding and

maneuverLGforTakingOff refine maneuverLandingGear using *AND* refinement operator. The domain model associated to this level refines the one with the root level by adding a new DataType called MANEUVER_TYPES to precise the possible maneuvers of the landing gear. It instantiates two STRING DataValues taking_off and landing.

```
REFINEMENT Landing_Gear_System_L1
REFINES Landing_Gear_System
SEES LandingGearSystem_CONTEXT, LandingGearSystem_CONTEXT_L1
ABSTRACT_VARIABLES currentManeuver, lgState
INVARIANT
    currentManeuver ∈ LandingGearSystem → MANEUVER_TYPES
INITIALISATION
    lgState :∈ LandingGearSystem → LG_STATES ||
    currentManeuver :∈ LandingGearSystem → MANEUVER_TYPES
EVENTS
    maneuverLGforLanding ref_and maneuverLandingGear =
    BEGIN
        lgState(LG):= lg_extended || currentManeuver(LG):= landing
    END ;
    maneuverLGforTakingOff ref_and  maneuverLandingGear =
    BEGIN
        lgState(LG):= lg_retracted || currentManeuver(LG):= taking_off
    END
END
```

Listing 12 – The first refinement level of Figure 2.29 EVENT-B specification machine.

Listings 11 and 12 shows the EVENT-B specification obtained from the translation of the first refinement level presented in Figure 2.29. Each goal in this refinement level is translated into an event for which the body has been manually specified: if the actual state of the landing gear system is lg_extended then the current produced maneuver is landing (event maneuverLGforLanding) else the current produced maneuver is taking_off (event maneuverLGforTakingOff). The *AND* refinement operator between the root level and the first refinement level is formalized with a set of proof obligations defined with the keyword ref_and to be proved. These proof obligations will express that maneuverLGforLanding and maneuverLGforTakingOff are both responsible for maneuverLandingGear.

5.2.2.3 The fourth refinement level

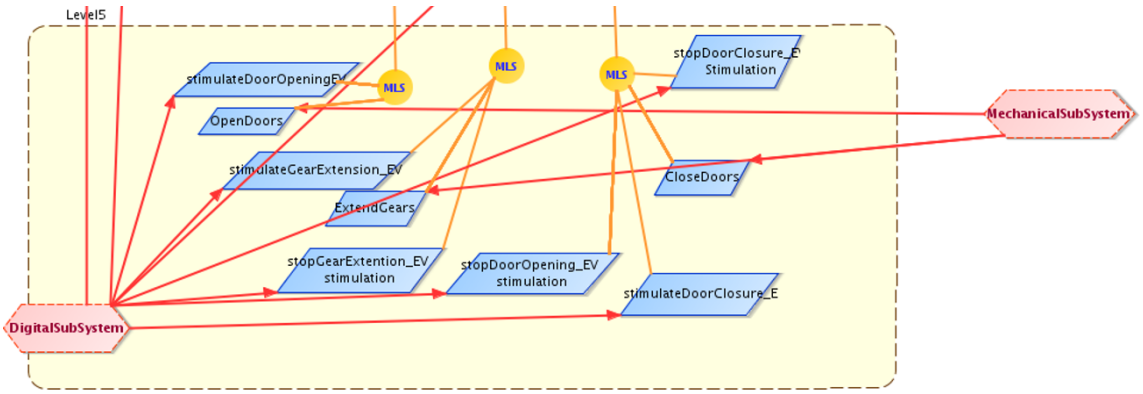

Figure 2.30 – SysML/KAOS root level goal model and corresponding domain model

```
SYSTEM LandingGearSystem_CONTEXT_L4
SEES LandingGearSystem_CONTEXT_L3
CONSTANTS DO_EV, DRSofDO_EV, GE_EV, GRSofGE_EV, DC_EV, DRSofDC_EV
PROPERTIES
    DO_EV ∈ ELECTRO_VALVES ∧
    DRSofDO_EV ∈ ELECTRO_VALVES ⇸ Doors ∧
    DRSofDO_EV = { DO_EV ↦ DRS } ∧
    GE_EV ∈ ELECTRO_VALVES   ∧
    GRSofGE_EV ∈ ELECTRO_VALVES ⇸ Gears ∧
    GRSofGE_EV = { GE_EV ↦ GRS } ∧
    DC_EV ∈ ELECTRO_VALVES ∧
    DRSofDC_EV ∈ ELECTRO_VALVES ⇸ Doors ∧
    DRSofDC_EV = { DC_EV ↦ DRS } ∧
    G_EV ≠ DO_EV ∧ DO_EV ≠ GE_EV ∧ G_EV ≠ GE_EV ∧
    GE_EV ≠ DC_EV ∧ DO_EV ≠ DC_EV ∧ DC_EV ≠ G_EV ∧
    ELECTRO_VALVES = { G_EV , DO_EV , GE_EV , DC_EV }
    END
```

Listing 13 – The fourth refinement level in Figure 2.30 Event-B specification context.

```
REFINEMENT Landing_Gear_System_L4
REFINES Landing_Gear_System_L3
SEES LandingGearSystem_CONTEXT, LandingGearSystem_CONTEXT_L1,
    LandingGearSystem_CONTEXT_L2, LandingGearSystem_CONTEXT_L3,
```

```
    LandingGearSystem_CONTEXT_L4
ABSTRACT_VARIABLES currentManeuver, lgState, decision, handleState,
    extensionOrder, GEVstate, doorsState, gearsState, DO_EVstate,
    GE_EVstate, DC_EVstate
INVARIANT
    DO_EVstate ∈ ELECTRO_VALVES → EV_States ∧
    GE_EVstate ∈ ELECTRO_VALVES → EV_States ∧
    DC_EVstate ∈ ELECTRO_VALVES → EV_States
INITIALISATION
    lgState :∈ LandingGearSystem → LG_STATES ||
    currentManeuver :∈ LandingGearSystem → MANEUVER_TYPES ||
    decision :∈ LandingGearSystem → Actions ||
    handleState :∈ Handle → HANDLE_STATES ||
    doorsState :∈ Doors → Doors_States ||
    gearsState :∈ Gears → Gears_States ||
    extensionOrder :∈ LandingGearSystem → BOOL ||
    GEVstate :∈ ELECTRO_VALVES → EV_States ||
    DO_EVstate :∈ ELECTRO_VALVES → EV_States ||
    GE_EVstate :∈ ELECTRO_VALVES → EV_States ||
    DC_EVstate :∈ ELECTRO_VALVES → EV_States
EVENTS
    stimulateDoorOpeningEV ref_milestone makeDoorsOpen =
    SELECT GEVstate ( lgOfG_EV ~ ( LG ) ) = stimulated THEN
         DO_EVstate ( DRSofDO_EV ~ ( DRS ) ) := stimulated
    END ;
    OpenDoors  ref_milestone makeDoorsOpen =
    SELECT DO_EVstate ( DRSofDO_EV ~ ( DRS ) ) = stimulated THEN
         doorsState ( lgOfDRS ~ ( LG ) ) := open
    END ;
    stimulateGearExtension_EV ref_milestone makeGearsExtend =
        SELECT doorsState ( lgOfDRS ~ ( LG ) ) = open THEN
        GE_EVstate ( GRSofGE_EV ~ ( GRS ) ) := stimulated
    END ;
    ExtendGears ref_milestone makeGearsExtend =
    SELECT GE_EVstate ( GRSofGE_EV ~ ( GRS ) ) = stimulated THEN
        gearsState ( lgOfGRS ~ ( LG ) ) := extended
    END ;
```

```
    stopGearExtension_EV ref_milestone makeGearsExtend =
    SELECT gearsState ( lgOfGRS ~ ( LG ) ) = extended THEN
        GE_EVstate ( GRSofGE_EV ~ ( GRS ) ) := stopped
    END ;
    stopDoorOpeningEV ref_milestone makeDoorsClose =
    SELECT  gearsState ( lgOfGRS ~ ( LG ) ) = extended THEN
         DO_EVstate ( DRSofDO_EV ~ ( DRS ) ) := stopped
    END ;
    stimulateDoorClosure_EV ref_milestone makeDoorsClose  =
    SELECT DO_EVstate ( DRSofDO_EV ~ ( DRS ) ) = stopped THEN
         DC_EVstate ( DRSofDC_EV ~ ( DRS ) ) := stimulated
    END ;
    closeDoors ref_milestone makeDoorsClose  =
    SELECT DC_EVstate ( DRSofDC_EV ~ ( DRS ) ) = stimulated THEN
         doorsState ( lgOfDRS ~ ( LG ) ) := close
    END ;
    stopDoorClosure_EV ref_milestone makeDoorsClose  =
    SELECT doorsState ( lgOfDRS ~ ( LG ) ) = close THEN
         DC_EVstate ( DRSofDC_EV ~ ( DRS ) ) := stopped
    END
END
```

Listing 14 – The fourth refinement level of Figure 2.30 EVENT-B specification machine.

For the fourth refinement level presented in Figure 2.30, the translation to EVENT-B introduces sets, constants, properties, variables and invariants raised from the domain model associated with this level which is a refinement of the third refinement level domain model. Also, 9 events are obtained from the translation of goals and the milestone refinement operators (*MLS* operator) presented between this refinement level goals and their corresponding parent goals are formalized with a set of proof obligations defined with the keyword ref_milestone to be proved. The EVENT-B specification associated to this level is presented in Listings 13 and 14.

5.2.3 SysML/KAOS Goal model decomposition

SysML/KAOS goal models allow the capture of assignments of requirements to agents responsible for their achievement. Each agent is associated with a sub-system. To ensure the distribution of work between several agents and a better maintenability, reusability and scalability of the system, SysML/KAOS allows its partitioning into sub-systems (presented in 3.1.3.3): a goal diagram models the main system and further goal diagrams are built for sub-systems. Actually, each sub-system is associated with an agent that is responsible for achieving its requirements.

In Figure 2.27, leaf goals are assigned to three sub-systems PiloteSubSystem, MechanicalSubSystem and DigitalSubSystem.

— The PiloteSubSystem is responsible for goals putHandleDown and putHandleUp, shown in Figure 2.31.

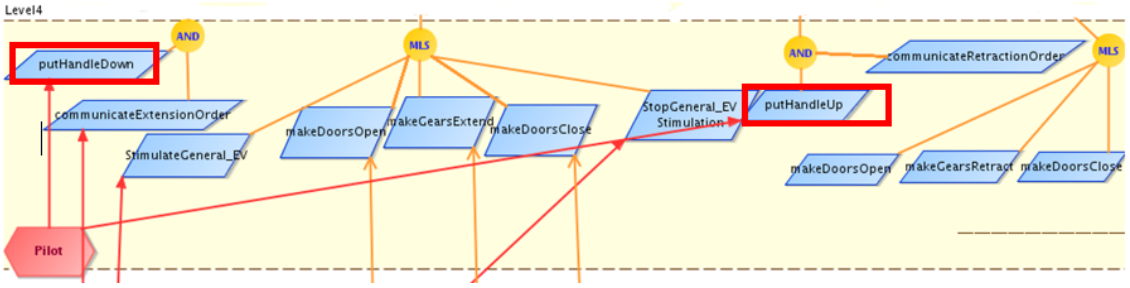

Figure 2.31 – SysML/KAOS PiloteSubSystem assigned goals

— The DigitalSubSystem is responsible for goals communicateExtensionOrder, StimulateGeneral_EV, StimulateDoorOpeningEV, StimulateGearExtension_EV, StopGearExtension_EVStimuation, StopDoorOpening_EVStimulation, StimulateDoorClosure_EV, StopDoorClosure_EV Stimulation and StopGeneral_EVStimulation, shown in Figure 2.32 (Framed in red).

— The MechanicalSubSystem is responsible for goals OpenDoors, ExtendGears and CloseDoors, shown in Figure 2.32 (Framed in green).

After decomposition we give as example the goal model associated to the mechanical subsystem shown in Figure 2.33.

5.2.4 SysML/KAOS model decomposition formalization using Event-B

For the landing gear system, the decomposition must be introduced in the third refinement level because it contains leaf goals associated to PiloteSubSystem and DigitalSubSystem

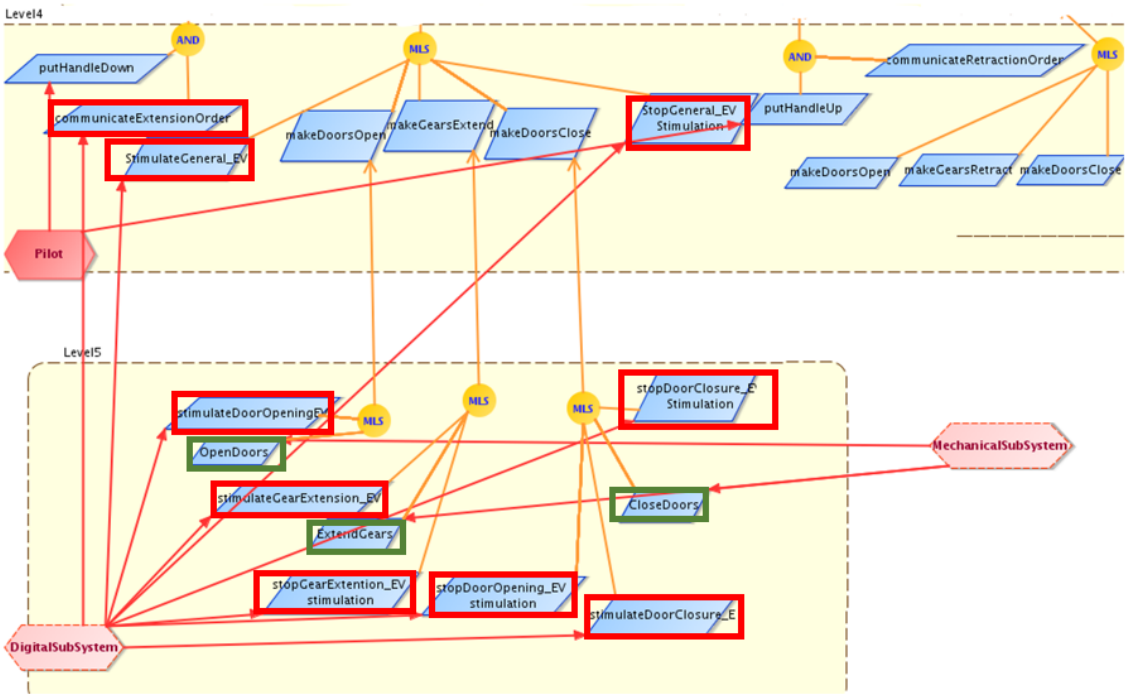

Figure 2.32 – SysML/KAOS DigitalSubSystem and MechanicalSubSystem assigned goals

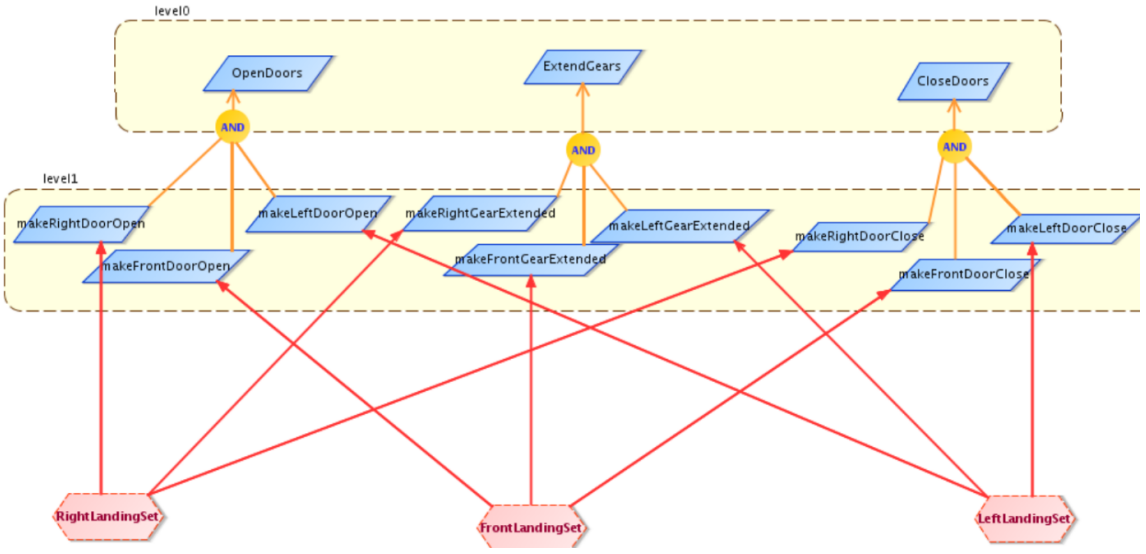

Figure 2.33 – SysML/KAOS MechanicalSubSystem goal model

and in the fourth refinement level because it contains leaf goals associated to DigitalSubSystem and mechanicalSubSystem. For instance, an interface is defined for each subsystem. Precisely, each SysML/KAOS agent gives an interface.

5.2.4.1 SysML/KAOS **Pilote SubSystem**

Here, we give as example the PiloteSubSystem interface presented in Listing 15. This interface is obtained from the decomposition of Landing_Gear_System_L3 along with variables and events corresponding to the PiloteSubSystem. Variable handleState is defined as a *shared variable* while it is updated by the PiloteSubSystem interface and read by DigitalSubSystem. Variables extensionOrder and decision are also considered as *shared variable* while they are updated in DigitalSS_Interface and read in PiloteSS_Interface. Events putHandleDown and putHandleUp are internal events in PiloteSS_Interface, whereas event communicateExtensionOrder is an external event. This event simulates the behavior of internal event communicateExtensionOrder, defined in interface DigitalSS_Interface because it uses the variable handleState.

```
SYSTEM PiloteSS_Interface
SEES LandingGearSystem_CONTEXT, LandingGearSystem_CONTEXT_L3
ABSTRACT_VARIABLES
    /****Shared variable used in DigitalSS_Interface****/
    handleState,
    //updated in PiloteSS_Interface and read in DigitalSS_Interface.
    extensionOrder,
    //updated in DigitalSS_Interface and read in PiloteSS_Interface.
    decision,
    //updated in DigitalSS_Interface and read in PiloteSS_Interface.
INVARIANT
    handleState ∈ Handle → HANDLE_STATES ∧
    decision ∈ LandingGearSystem → Actions ∧
    extensionOrder ∈ LandingGearSystem → BOOL
INITIALISATION
    handleState :∈ Handle → HANDLE_STATES ||
    decision :∈ LandingGearSystem → Actions ||
    extensionOrder :∈ LandingGearSystem → BOOL
EVENTS
    putHandleDown =
    BEGIN
    handleState(LgOfHd~(LG)):=down
    END;
    putHandleUp =
```

```
    BEGIN
    handleState(LgOfHd~(LG)):=up
    END
/*************Shared Events of DigitalSS_Interface****************/
    communicateExtensionOrder =
    SELECT handleState(LgOfHd~(LG))=down THEN
    extensionOrder(LG) :=TRUE || decision(LG):=Extension
    END;
/*****************************************************************/
END
```

Listing 15 – SysML/KAOS Pilote SubSystem Interface.

5.2.4.2 SysML/KAOS **Mechanical SubSystem**

Listings 16 and 17 present an overview of the first refinement level Event-B of the sub-system associated to the agent MechanicalSubSystem. It is a refinement of the interface MechanicalSS_Interface. This specification corresponds to the goal diagram presented in Figure 2.33. As introduced in this goal diagram, to satisfy the goal OpenDoors, the left, right and front doors should be opened. This is represented by the *AND* refinement operator. To formalize this refinement, events makeRightDoorOpen, makeFrontDoorOpen and makeLeftDoorOpen associated to the level 1 of the MechanicalSubSystem goal diagram ref_and the interface event OpenDoors.

```
SYSTEM LandingGearSystem_MechanicalSS_CONTEXT
SEES LandingGearSystem_CONTEXT_L3
SETS SIDES_DOORS; SIDES_GEARS
CONSTANTS
    leftD, DRSofLD,     rightD, DRSofRD,        frontD, DRSofFD,
    leftG, GRSofLG,     rightG, GRSofRG,        frontG, GRSofFG
PROPERTIES
    leftD ∈ SIDES_DOORS ∧ DRSofLD ∈ SIDES_DOORS ⇸
    Doors ∧ DRSofLD = {leftD ↦ DRS} ∧

    rightD ∈ SIDES_DOORS ∧ DRSofRD ∈ SIDES_DOORS ⇸
    Doors ∧ DRSofRD = {rightD ↦ DRS} ∧
```

```
    frontD ∈ SIDES_DOORS ∧ DRSofFD ∈ SIDES_DOORS ⇸
    Doors ∧ DRSofFD = {frontD ↦ DRS} ∧

    leftD ≠ rightD ∧ rightD ≠ frontD ∧ leftD ≠ frontD ∧
    SIDES_DOORS={leftD,rightD,frontD} ∧

    leftG ∈ SIDES_GEARS ∧ GRSofLG ∈ SIDES_GEARS ⇸
    Gears ∧ GRSofLG = {leftG ↦ GRS} ∧

    rightG ∈ SIDES_GEARS ∧ GRSofRG∈ SIDES_GEARS ⇸
    Gears ∧ GRSofRG = {rightG ↦ GRS} ∧

    frontG ∈ SIDES_GEARS ∧ GRSofFG ∈ SIDES_GEARS ⇸
    Gears ∧ GRSofFG = {frontG ↦ GRS} ∧

    leftG ≠ rightG ∧ rightG ≠ frontG ∧ leftG ≠ frontG ∧
    SIDES_GEARS={leftG,rightG,frontG}
END
```

Listing 16 – SysML/KAOS Mechanical SubSystem Level 1 context.

```
REFINEMENT MechanicalSS_L1
REFINES MechanicalSS_Interface
SEES LandingGearSystem_CONTEXT, LandingGearSystem_CONTEXT_L3,
    LandingGearSystem_CONTEXT_L4,
    LandingGearSystem_MechanicalSS_CONTEXT
ABSTRACT_VARIABLES DO_EVstate, GE_EVstate, DC_EVstate, doorsState,
    gearsState, sides_doorState, sides_gearState
INVARIANT
    sides_doorState ∈ SIDES_DOORS → Doors_States ∧
    sides_gearState ∈ SIDES_GEARS → Gears_States ∧
    (sides_doorState[SIDES_DOORS]={open} ⇒
    doorsState[Doors]={open}) ∧
    (sides_doorState[SIDES_DOORS]={close} ⇒
    doorsState[Doors]={close}) ∧
    (sides_gearState[SIDES_GEARS]={extended} ⇒
```

```
    gearsState[Gears]={extended})
INITIALISATION
    DO_EVstate :∈ ELECTRO_VALVES → EV_States ||
    GE_EVstate :∈ ELECTRO_VALVES → EV_States ||
    DC_EVstate :∈ ELECTRO_VALVES → EV_States ||

    doorsState, sides_doorState ∈ (
    (sides_doorState[SIDES_DOORS]={open} ⇒
    doorsState[Doors]={open}) ∧
    (sides_doorState[SIDES_DOORS]={close} ⇒
     doorsState[Doors]={close}) ∧
    (doorsState ∈ Doors →
    Doors_States) ∧
    (sides_doorState ∈ SIDES_DOORS → Doors_States)) ||

    gearsState, sides_gearState ∈ (
    (sides_gearState[SIDES_GEARS]={extended} ⇒
    gearsState[Gears]={extended}) ∧
    (gearsState ∈ Gears → Gears_States) ∧
    (sides_gearState ∈ SIDES_GEARS → Gears_States))
EVENTS
    makeRightDoorOpen ref_and OpenDoors =
    SELECT DO_EVstate(DRSofDO_EV~(DRS)) = stimulated THEN
    sides_doorState(DRSofRD~(DRS)):=open
    END;
    makeLeftDoorOpen ref_and OpenDoors =
    SELECT DO_EVstate(DRSofDO_EV~(DRS)) = stimulated THEN
    sides_doorState(DRSofLD~(DRS)):=open
    END;
    makeFrontDoorOpen ref_and OpenDoors =
    SELECT DO_EVstate(DRSofDO_EV~(DRS)) = stimulated THEN
    sides_doorState(DRSofFD~(DRS)):=open
    END;
    makeRightGearExtended ref_and ExtendGears =
    SELECT GE_EVstate(GRSofGE_EV~(GRS))=stimulated THEN
    sides_gearState(GRSofRG~(GRS)):=extended
    END;
```

```
    makeLeftGearExtended ref_and ExtendGears =
    SELECT GE_EVstate(GRSofGE_EV~(GRS))=stimulated THEN
    sides_gearState(GRSofLG~(GRS)):=extended
    END;
    makeFrontGearExtended ref_and ExtendGears =
    SELECT GE_EVstate(GRSofGE_EV~(GRS))=stimulated THEN
    sides_gearState(GRSofFG~(GRS)):=extended
    END;
    makeRightDoorClose ref_and closeDoors =
    SELECT DC_EVstate(DRSofDC_EV~(DRS)) = stimulated THEN
    sides_doorState(DRSofRD~(DRS)):=close
    END;
    makeLeftDoorClose ref_and closeDoors =
    SELECT DC_EVstate(DRSofDC_EV~(DRS)) = stimulated THEN
    sides_doorState(DRSofLD~(DRS)):=close
    END;
    makeFrontDoorClose ref_and closeDoors =
    SELECT DC_EVstate(DRSofDC_EV~(DRS)) = stimulated THEN
    sides_doorState(DRSofFD~(DRS)):=close
    END
END
```

Listing 17 – SysML/KAOS Mechanical SubSystem Level 1 machine.

Figure 2.34 shows the Event-B specification architecture created from the formalization of the SysML/KAOS models. In this architecture, blue arrows show the SEES relationship between a machine and a context and black arrows present the refinement between two machines. Finally, green arrows present the SysML/KAOS model decomposition Event-B formalization.

5.2.5 Conclusion

In this section, we have presented the use of the SysML/KAOS method for the modeling of system requirements and associated domain models related to the landing gear system [Boniol and Wiels, 2014]. The Event-B formalization of the SysML/KAOS models presented in [Matoussi et al., 2011a, Tueno et al., 2017a] has then been applied to obtain a

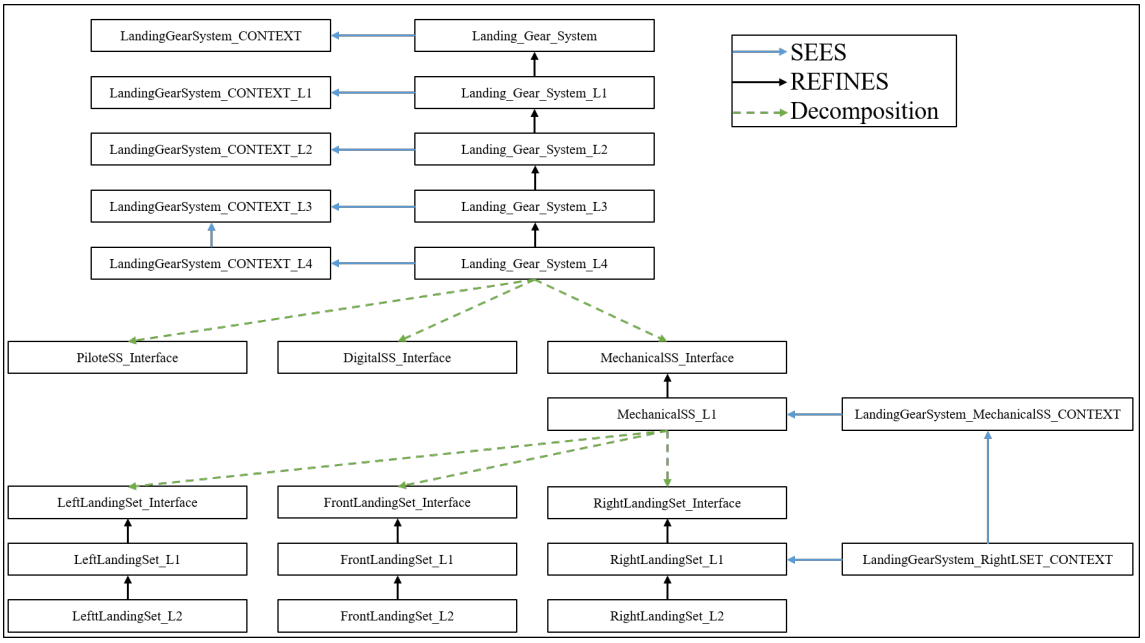

Figure 2.34 – SysML/KAOS Event-B specification Architecture

formal specification containing the system structure and the skeleton of events. AtelierB [AtelierB, 1990] has been used to verify and validate the formal specification, especially to prove the consistency of system requirements and the refinement logic, after the completion of the body of events. The formal verification step for the Event-B specification of the landing gear system SysML/KAOS required to discharge 114 proof obligations. Figure 2.35 summarises the status of the AtelierB project corresponding to the generated Event-B specification. These proofs are of type invariant preservation, non-deterministic action feasibility and well-definedness enriched with proof obligations that represent the goal model refinement patterns. They were all automatically and interactively proved (100%). The columns of the table are: **TC**, for “type checking", to indicate that formal model components are well defined, **GOP**, for “generation of proof obligations", to indicate that proof obligations are well generated. **PO**, for “proof obligations", presents the number of generated proof obligations for every formal model component. **UN** for “unproved" presents the number of unproved proof obligations of each element. Finally, the column **PR** presents the percentage of discharged proof obligations

The full specification of this landing gear system case study can be found in [Bougacha, 2022b].

Project Status for Landing_Gear_System_SysMLKAOS

COMPONENT	TC	GOP	PO	UN	PR	CC	LINES	RULES
DigitalSS_Interface	OK	OK	12	0	100 %		135	0
LandingGearSystem_CONTEXT	OK	OK	0	0	100 %		21	0
LandingGearSystem_CONTEXT_L1	OK	OK	0	0	100 %		17	0
LandingGearSystem_CONTEXT_L2	OK	OK	0	0	100 %		18	0
LandingGearSystem_CONTEXT_L3	OK	OK	0	0	100 %		56	0
LandingGearSystem_CONTEXT_L4	OK	OK	0	0	100 %		31	0
LandingGearSystem_MechanicalSS_CONTEXT	OK	OK	0	0	100 %		54	0
LandingGearSystem_RightLSET_CONTEXT	OK	OK	0	0	100 %		82	0
Landing_Gear_System	OK	OK	1	0	100 %		25	0
Landing_Gear_System_L1	OK	OK	2	0	100 %		32	0
Landing_Gear_System_L2	OK	OK	4	0	100 %		35	0
Landing_Gear_System_L3	OK	OK	15	0	100 %		75	0
Landing_Gear_System_L4	OK	OK	18	0	100 %		94	0
MechanicalSS_Interface	OK	OK	3	0	100 %		140	0
MechanicalSS_L1	OK	OK	17	0	100 %		86	0
PiloteSS_Interface	OK	OK	6	0	100 %		39	0
RightLandingSet_Interface	OK	OK	3	0	100 %		57	0
RightLandingSet_L1	OK	OK	22	0	100 %		91	0
RightLandingSet_L2	OK	OK	11	0	100 %		65	0
TOTAL	OK	OK	114	0	100 %	OK	1153	0

Figure 2.35 – Project Status for Landing Gear System SysML/KAOS Event-B specification

5.3 High-level architecture modeling and formalizing of the landing gear system case study

In this section, we give an illustration of the HLA modeling approach presented in Section 3 applied on the landing gear system case study.

5.3.1 The main system (Level 0)

To model the HLA hierarchy, we start the modeling process by creating a new package diagram that represents the HLA hierarchy levels as a set of packages. Each package contains a BDD, a state-machine diagram and a sequence diagram.

The first package to be modeled in this hierarchy is LandingGearSystemL0 that describes the main system. This package shown in Figure 2.36 is composed of:

— A unique block called Landing Gear System which describes the main system.

— A state-machine diagram called Landing Gear System States which describes the behavior of the main system.

— A sequence diagram that shows the main functionalities and the life cycle of a landing gear system, that is to extend and retract a landing gear.

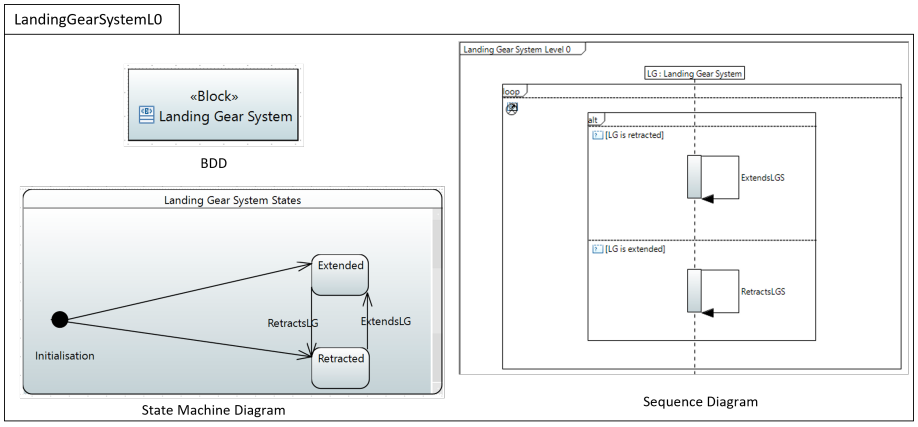

Figure 2.36 – Landing gear system HLA main system package

```
SYSTEM LandingGearSystemL0_CONT
SETS LandingGearSystem;
    LandingGearSystemStates
CONSTANTS lg, Extended,
    Retracted
PROPERTIES
    lg ∈ LandingGearSystem ∧
    LandingGearSystem ={lg} ∧
    Retracted ∈
    LandingGearSystemStates ∧
    Extended ∈
    LandingGearSystemStates ∧
    Retracted ≠ Extended ∧
    LandingGearSystemStates =
    {Extended, Retracted}
END
```

Listing 18 – Landing gear system HLA main system Event-B specification context.

```
SYSTEM LandingGearSystemL0
SEES LandingGearSystemL0_CONT
VARIABLES lgState
INVARIANT
    lgState ∈ LandingGearSystem
    → LandingGearSystemStates
INITIALISATION
    lgState :∈ {lg} →
    LandingGearSystemStates
EVENTS
    RetractsLGS =
    SELECT lgState(lg)=Extended
    THEN
        lgState(lg):=Retracted
    END;
    ExtendsLGS =
    SELECT lgState(lg)=Retracted
    THEN
        lgState(lg):=Extended
    END
END
```

Listing 19 – Landing gear system HLA main system Event-B specification machine.

The EVENT-B formalization of the landing gear system HLA main system package presented in Figure 2.36 is shown in Listings 18 and 19. The BDD and state-machine diagrams give rise to sets, constants and properties. Variables and typing invariants are generated from sequence diagram lifelines associated to BDD blocks. The sequence diagram messages ExtendsLGS and RetractsLGS are translated into EVENT-B events representing the life cycle of system states from extended to retracted. Recall that the sequence diagrams messages are associated to transitions of the state-machine diagram. Thus, the guard of the event comes from the source state of the transition and its action is the target state.

5.3.2 Landing gear system HLA level 1

The landing gear system is composed of three sub-systems (Pilote SubSystem, Digital SubSystem and Mechanical SubSystem) and a Pilot to command the extension and retraction of the gear. Thus, the behavior of the main system (landing gear system) is satisfied by the result of these entities interplay.

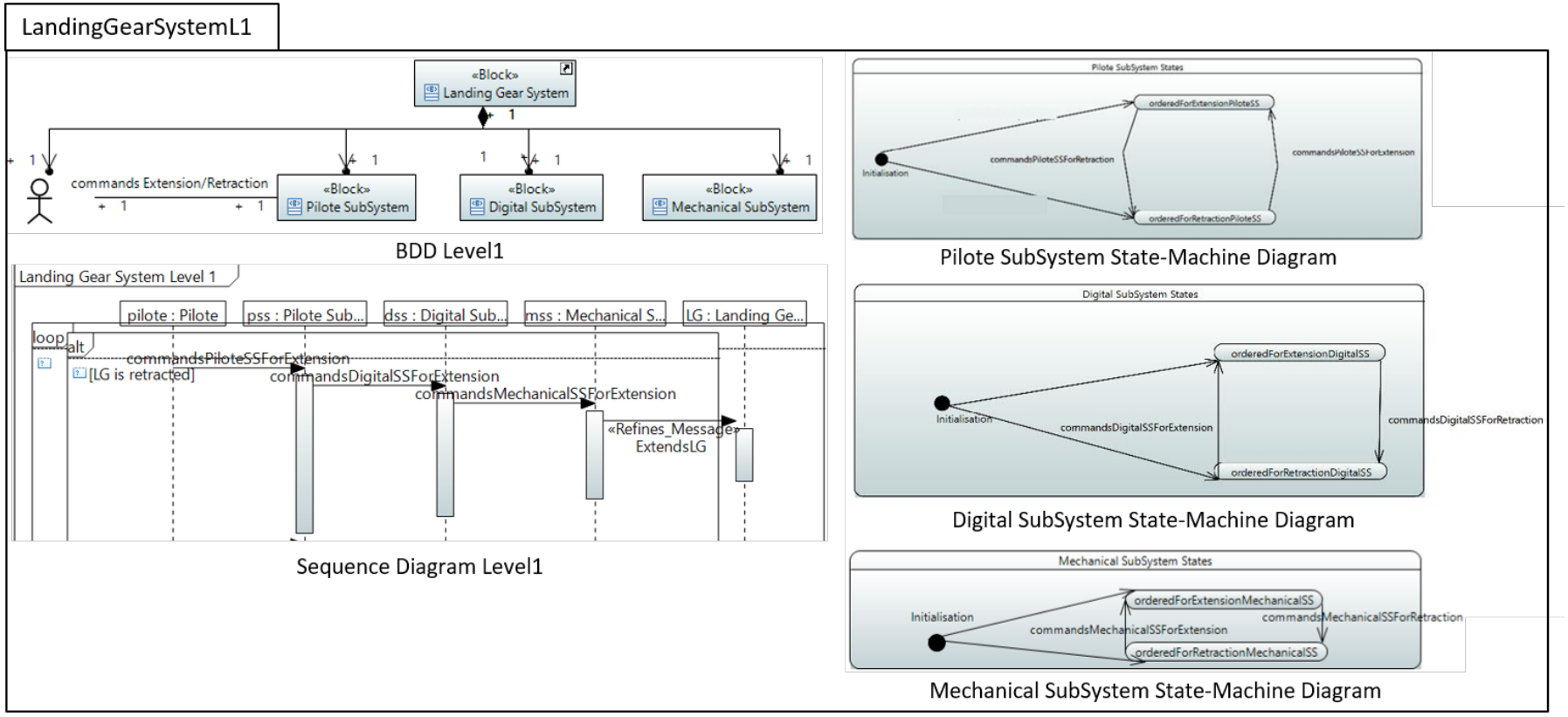

Figure 2.37 – Landing gear system HLA Level 1 package

To process in the HLA hierarchy, we create a second package called LandingGearSystemL1 which encompasses the different entities and the composition relationship with their parent system.

The package LandingGearSystemL1 shown in Figure 2.37 contains the BDD describing the sub-systems, the state-machine diagrams for all the sub-systems (Pilote, Digital and Mechanical SubSystems) and the sequence diagram, an extract of it is shown (the full version can be found in [Bougacha, 2022b]) which defines the Pilot/sub-systems process that refines the main system process.

To represent the main system behavior satisfaction by the sub-systems behaviors interplay, we use the *HLA_refines* link, presented in 3.1.1.1, between LandingGearSystemL0 and LandingGearSystemL1 to express that LandingGearSystemL1 refines LandingGearSystemL0 (see Figure 2.38). More precisely, the LandingGearSystemL1 sequence diagram message ExtendsLG allows the extension of the landing gear system after the sub-systems processes interplay. Therefore, it is a refinement of the message ExtendsLGS of the sequence diagram of the LandingGearSystemL0, specified by the *Refines_Message* stereotype (see Figure 2.37).

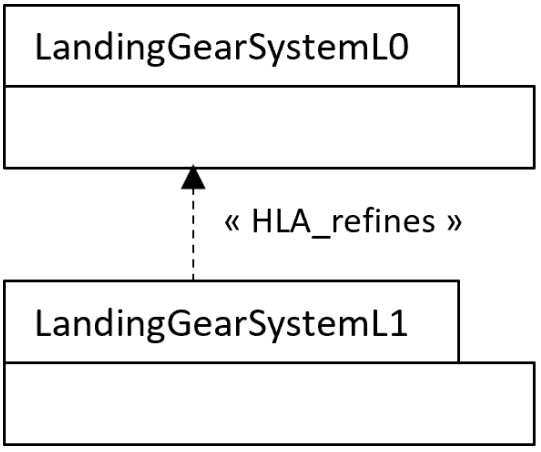

Figure 2.38 – Landing gear system HLA level 0 & 1 package refinement

Listings 20 and 21 shows the EVENT-B specification obtained from the translation of LandingGearSystemL1 package presented in Figure 2.37. Each sequence diagram message is translated into an event. The created set of events represents the collaboration scenario between participating sub-systems in order to satisfy the parent system behavior. As already mentioned, LandingGearSystemL0 *HLA_refines* LandingGearSystemL1 and the sub-systems behaviors interplay allow the extending of the landing gear represented by the message ExtendsLG. This message refines the message ExtendsLGS from LandingGearSystemL0 using the stereotype *Refines_Message*. To formalize this refinement in EVENT-B, the machine

created from the first package LandingGearSystemL0 is refined by the machine created from LandingGearSystemL1 (the clause REFINES in Listing 21) and the event ExtendsLG from LandingGearSystemL1 refines the event ExtendsLGS from LandingGearSystemL0 (the use of ref between the two events). Same for the retraction scenario where the event RetractsLG LandingGearSystemL1 refines the event RetractsLGS from LandingGearSystemL0.

```
SYSTEM LandingGearSystemL1_CONT
SETS DigitalSubSystem; MechanicalSubSystem; PiloteSubSystem;
    DigitalSubSystemStates; PiloteSubSystemStates;
    MechanicalSubSystemStates; PILOT
CONSTANTS  mss, dss, pss, pilot,
    orderedForExtensionPiloteSS, orderedForRetractionPiloteSS,
    orderedForExtensionDigitalSS, orderedForRetractionDigitalSS,
    orderedForExtensionMechanicalSS, orderedForRetractionMechanicalSS,
    commandsExtension, commandsRetraction
PROPERTIES
    mss ∈ MechanicalSubSystem ∧ dss ∈ DigitalSubSystem ∧
    pss ∈ PiloteSubSystem ∧ pilot ∈ PILOT ∧ PILOT={pilot} ∧
    PiloteSubSystem ={pss} ∧ MechanicalSubSystem ={mss} ∧
    DigitalSubSystem ={dss} ∧

    orderedForRetractionPiloteSS ∈ PiloteSubSystemStates ∧
    orderedForExtensionPiloteSS ∈ PiloteSubSystemStates ∧
    orderedForExtensionPiloteSS ≠ orderedForRetractionPiloteSS ∧

    orderedForRetractionDigitalSS ∈ DigitalSubSystemStates ∧
    orderedForExtensionDigitalSS ∈ DigitalSubSystemStates ∧
    orderedForExtensionDigitalSS ≠ orderedForRetractionDigitalSS ∧

    orderedForExtensionMechanicalSS ∈ MechanicalSubSystemStates ∧
    orderedForRetractionMechanicalSS ∈ MechanicalSubSystemStates ∧
    orderedForRetractionMechanicalSS ≠
    orderedForExtensionMechanicalSS ∧

    DigitalSubSystemStates ={orderedForRetractionDigitalSS,
    orderedForExtensionDigitalSS} ∧
    PiloteSubSystemStates ={orderedForRetractionPiloteSS,
```

```
    orderedForExtensionPiloteSS} ∧
    MechanicalSubSystemStates ={orderedForExtensionMechanicalSS,
    orderedForRetractionMechanicalSS} ∧

    commandsExtension ∈ {pilot} ↦ {pss} ∧
    commandsRetraction ∈ {pilot} ↦ {pss}
    END
```

Listing 20 – Landing gear system HLA Level 1 EVENT-B specification context.

```
REFINEMENT LandingGearSystemL1
REFINES LandingGearSystemL0
SEES LandingGearSystemL1_CONT, LandingGearSystemL0_CONT
VARIABLES dssState, mssState, pssState, lgState
INVARIANT
    dssState ∈ DigitalSubSystem → DigitalSubSystemStates ∧
    mssState ∈ MechanicalSubSystem → MechanicalSubSystemStates ∧
    pssState ∈ PiloteSubSystem → PiloteSubSystemStates
INITIALISATION
    dssState :∈ {dss} → DigitalSubSystemStates ||
    mssState :∈ {mss} → MechanicalSubSystemStates ||
    pssState :∈ {pss} → PiloteSubSystemStates ||
    lgState :∈ {lg} → LandingGearSystemStates
EVENTS
    commandsMechanicalSSForRetraction =
    SELECT dssState(dss)=orderedForRetractionDigitalSS ∧
    mssState(mss)=orderedForExtensionMechanicalSS THEN
        mssState(mss):=orderedForRetractionMechanicalSS
    END;
    ExtendsLG ref ExtendsLGS=
    SELECT lgState(lg)=Retracted ∧
    mssState(mss)=orderedForExtensionMechanicalSS THEN
        lgState(lg):=Extended
    END;
    commandsMechanicalSSForExtension =
    SELECT dssState(dss)=orderedForExtensionDigitalSS ∧
    mssState(mss)=orderedForRetractionMechanicalSS THEN
```

```
        mssState(mss):=orderedForExtensionMechanicalSS
    END;
    RetractsLG ref RetractsLGS=
    SELECT lgState(lg)=Extended ∧
    mssState(mss)=orderedForRetractionMechanicalSS THEN
        lgState(lg):=Retracted
    END;
    commandsDigitalSSForRetraction =
    SELECT pssState(pss)=orderedForRetractionPiloteSS ∧
    dssState(dss)=orderedForExtensionDigitalSS THEN
        dssState(dss):=orderedForRetractionDigitalSS
    END;
    commandsPiloteSSForExtension =
    SELECT lgState(lg)=Retracted ∧
    pssState(pss)=orderedForRetractionPiloteSS THEN
        pssState(pss):=orderedForExtensionPiloteSS
    END;
    commandsDigitalSSForExtension =
    SELECT pssState(pss)=orderedForExtensionPiloteSS ∧
    dssState(dss)=orderedForRetractionDigitalSS THEN
        dssState(dss):=orderedForExtensionDigitalSS
    END;
    commandsPiloteSSForRetraction =
    SELECT lgState(lg)=Extended ∧
    pssState(pss)=orderedForExtensionPiloteSS THEN
        pssState(pss):=orderedForRetractionPiloteSS
    END
END
```

Listing 21 – landing gear system HLA Level 1 EVENT-B specification machine.

5.3.3 Landing gear system HLA decomposition

The landing gear system is composed of three sub-systems PiloteSubSystem, DigitalSubSystem and MechanicalSubSystem as presented in the BDD of the package LandingGearSystemL1 shown in Figure 2.37. Each of these sub-systems has its own life and can exist

independently of the other blocks. As denoted in Section 3.1.1.1, LandingGearSystemL1 is decomposed and a new package is created for each sub-system and a *HLA_decompose* link is established between these new packages and the parent system package. Therefore, LandingGearSystemL1 is *HLA_decompose* into PiloteSubSystem, DigitalSubSystem and MechanicalSubSystem. This hierarchy is represented in Figure 2.39. Each of the packages can then be described by a BDD, representing its structure and in particular its possible sub-systems, the state-machines associated to some blocks and a sequence diagram.

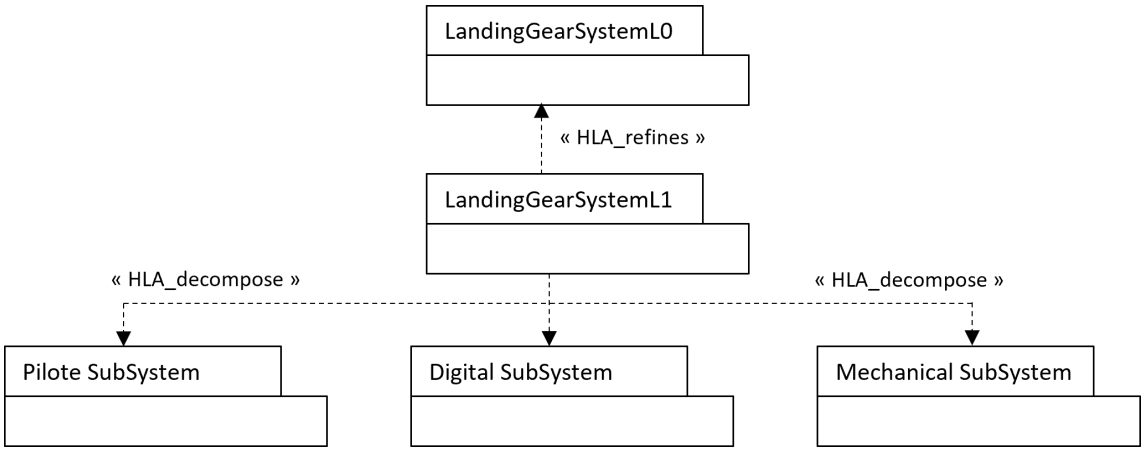

Figure 2.39 – Landing gear system decomposition

The Event-B translation of the SysML decomposition extension applied on the machine LandingGearSystemL1 give rise to four interfaces following the translation rules presented in Section 3.2.1.2. These interfaces are:

— The PiloteSubSystem_Interface related to the Pilote SubSystem.

— The DigitalSubSystem_Interface related to the Digital SubSystem.

— The MechanicalSubSystem_Interface related to the Mechanical SubSystem.

— LandingGearSystemL1 is a refinement machine then LandingGearSystemL1_Refinement_Interface is created. This interface includes abstract variables coming from refining LandingGearSystemL0 which is lgState and its associated elements (INVARIANTS, INITIALISATION, EVENTS, etc.).

Listing 22 shows the PiloteSubSystem_Interface which contains the variable pssState coming from LandingGearSystemL1 and representing the Pilote SubSystem and its associated elements. The associated elements to pssState from LandingGearSystemL1 are the typing INVARIANTS, its INITIALISATION and EVENTS: commandsPiloteSSForExtension and commandsPiloteSSForRetraction. EVENTS commandsDigitalSSForExtension and commands-

DigitalSSForRetraction are also associated to the variable pssState while they use its different states as guards to execute their actions.

```
SYSTEM PiloteSubSystem_Interface
SEES LandingGearSystemL1_CONT, LandingGearSystemL0_CONT
VARIABLES pssState
INVARIANT
    pssState ∈ PiloteSubSystem → PiloteSubSystemStates
INITIALISATION
    pssState :∈ {pss} → PiloteSubSystemStates
EVENTS
    commandsDigitalSSForRetraction =
    SELECT pssState(pss)=orderedForRetractionPiloteSS THEN
        skip
    END;
    commandsPiloteSSForExtension =
    SELECT pssState(pss)=orderedForRetractionPiloteSS THEN
        pssState(pss):=orderedForExtensionPiloteSS
    END;
    commandsDigitalSSForExtension =
    SELECT pssState(pss)=orderedForExtensionPiloteSS THEN
        skip
    END;
    commandsPiloteSSForRetraction =
    SELECT pssState(pss)=orderedForExtensionPiloteSS THEN
        pssState(pss):=orderedForRetractionPiloteSS
    END
END
```

Listing 22 – PiloteSubSystem Interface.

5.3.4 The Pilote SubSystem HLA

After decomposition, The Pilote SubSystem behaves independently from its participation in the landing gear system. It is represented with a package and can then be described by a BDD, representing its structure and in particular its sub-components, a Handle for this sub-system, the state-machine diagram associated to the Handle and a sequence diagram.

The PiloteSubSystemL1 package is represented in Figure 2.40 and it refines the PiloteSubSystem package created from LandingGearSystemL1 decomposition. This refinement is established between messages of sequence diagram of PiloteSubSystemL1 and messages of sequence diagram of PiloteSubSystem.

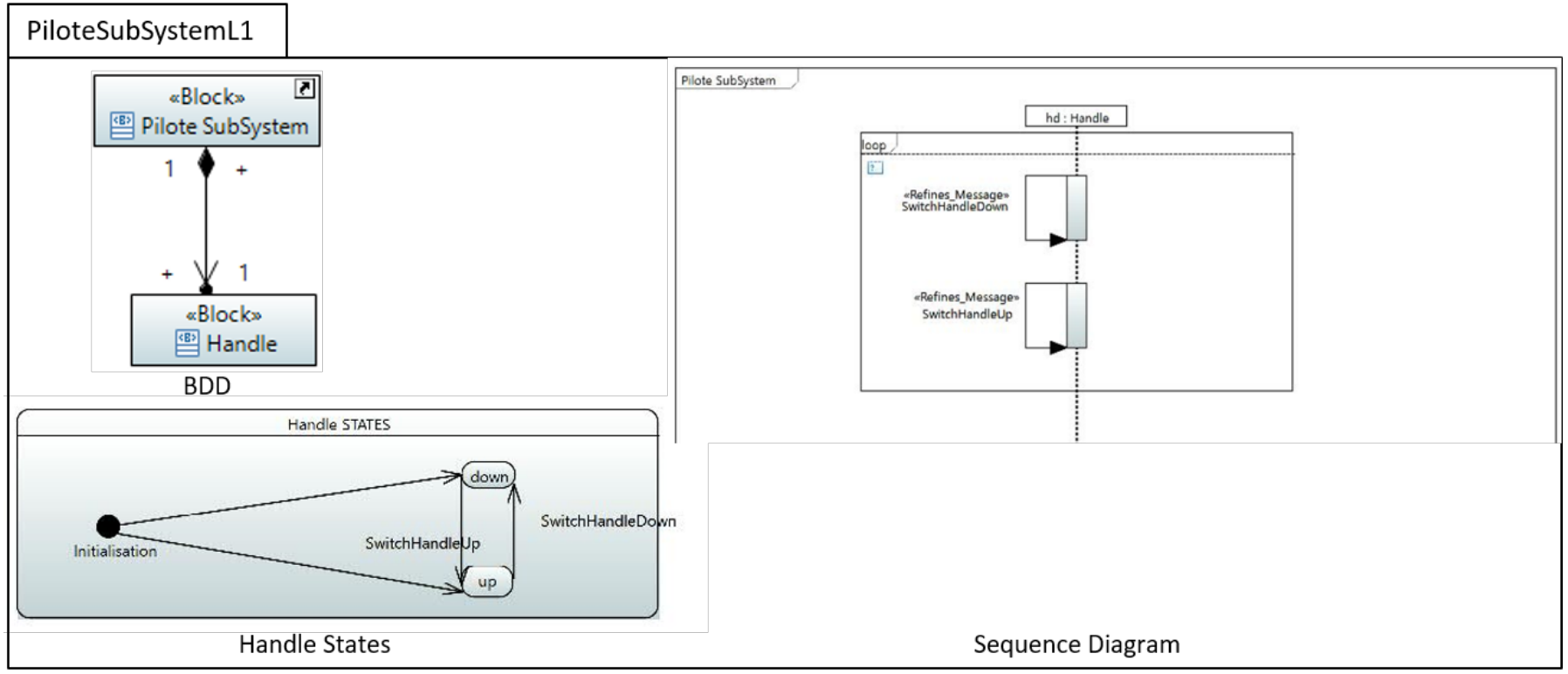

Figure 2.40 – HLA Pilote SubSystem Level 1

The EVENT-B translation of PiloteSubSystemL1 package is shown in Listing 23. A new machine called PiloteSubSystemL1 is created that refines PiloteSubSystem_Interface. Sequence diagram message SwitchHandleUp (resp. SwitchHandleDown) is translated into an event that refines the event commandsPiloteSSForRetraction (resp. commandsPiloteSS-ForExtension) from PiloteSubSystem_Interface.

```
REFINEMENT PiloteSubSystemL1
REFINES PiloteSubSystem_Interface
SEES PiloteSubSystem_CONT, LandingGearSystemL1_CONT,
    LandingGearSystemL0_CONT
VARIABLES hdState, pssState
INVARIANT
    hdState ∈ Handle → HandleSTATES
INITIALISATION
    hdState :∈ {hd} → HandleSTATES ||
    pssState :∈ {pss} → PiloteSubSystemStates
EVENTS
    SwitchHandleUp ref commandsPiloteSSForRetraction=
    SELECT      hdState(hd)=down ∧
    pssState(pss)=orderedForExtensionPiloteSS THEN
        hdState(hd):=up ||
        pssState(pss):=orderedForRetractionPiloteSS
    END;
    SwitchHandleDown ref commandsPiloteSSForExtension=
    SELECT      hdState(hd)=up ∧
    pssState(pss)=orderedForRetractionPiloteSS THEN
        hdState(hd):=down ||
        pssState(pss):=orderedForExtensionPiloteSS
    END
END
```

Listing 23 – HLA PiloteSubSystem Level 1

5.3.5 Formal verification of the landing gear system HLA Event-B specification

The formal verification step for the Event-B specification of the landing gear system HLA required to discharge 136 proof obligations. Figure 2.41 summarises the status of the AtelierB project corresponding to the generated Event-B specification. These proofs are of type invariant preservation, non-deterministic action feasibility and well-definedness. They were all automatically proved (100%).

Project Status for LandingGearSystem2

COMPONENT	TC	GOP	PO	UN	PR	CC	LINES	RULES
DigitalSubSystem_Interface	OK	OK	3	0	100 %		38	0
FrontLandingSetL1	OK	OK	20	0	100 %		90	0
FrontLandingSet_CONT	OK	OK	0	0	100 %		67	0
FrontLandingSet_Interface	OK	OK	3	0	100 %		41	0
LandingGearSystemL0	OK	OK	3	0	100 %		27	0
LandingGearSystemL0_CONT	OK	OK	0	0	100 %		17	0
LandingGearSystemL1	OK	OK	17	0	100 %		89	0
LandingGearSystemL1_CONT	OK	OK	0	0	100 %		47	0
LandingGearSystemL1_Refinement_Interface	OK	OK	3	0	100 %		42	0
LeftLandingSetL1	OK	OK	20	0	100 %		90	0
LeftLandingSet_CONT	OK	OK	0	0	100 %		67	0
LeftLandingSet_Interface	OK	OK	3	0	100 %		40	0
MechanicalSubSystemL1	OK	OK	19	0	100 %		92	0
MechanicalSubSystem_CONT	OK	OK	0	0	100 %		39	0
MechanicalSubSystem_Interface	OK	OK	3	0	100 %		40	0
MechanicalSubSystem_Refinement_Interface	OK	OK	3	0	100 %		63	0
PiloteSubSystemL1	OK	OK	13	0	100 %		38	0
PiloteSubSystem_CONT	OK	OK	0	0	100 %		17	0
PiloteSubSystem_Interface	OK	OK	3	0	100 %		40	0
RightLandingSetL1	OK	OK	20	0	100 %		90	0
RightLandingSet_CONT	OK	OK	0	0	100 %		67	0
RightLandingSet_Interface	OK	OK	3	0	100 %		41	0
TOTAL	OK	OK	136	0	100 %	OK	1182	0

Figure 2.41 – Project Status for landing gear system HLA Event-B specification

5.3.6 Conclusion

In this section, we have proposed the use of SysML extensions to be aligned with Event-B refinement and decomposition mechanisms on the landing gear system case study in order to automatically translate SysML models of HLA to Event-B specifications.

The full specification of this landing gear system case study can be found in [Bougacha, 2022b].

5.4 Requirements & high-level architecture alignment examples

In this section, we give an example of illustration of requirements & HLA alignment approach presented in Section 4 applied on the landing gear system case study. However, requirement & HLA alignment approach applied on the landing gear system case study is quite simple since each leaf goal is satisfied by only one message from HLA which does not give a sufficient explanation about how to use alignment kinds when a leaf goal is satisfied by more than one message from HLA. That is why, we enhance the landing gear example with a second example taken from the train control system case study presented in [Lamsweerde, 2008].

5.4.1 Example of alignment between landing gear system SysML/KAOS models and SysML HLA models

In this example we are interested to align the leaf goals putHandleDown and putHandleUp with messages of the HLA model. Figure 2.42 describes the two alignment links. As presented in Section 4, we have the HLA sequence diagram message SwitchHandleDown which satisfies the goal putHandleDown and the HLA sequence diagram message SwitchHandleUp which satisfies the goal putHandleUp. The arrows from the messages to the goals are labelled by the stereotype *Satisfy*. We have used this kind of alignment because each one of these leaf goals requires only one HLA message to be satisfied.

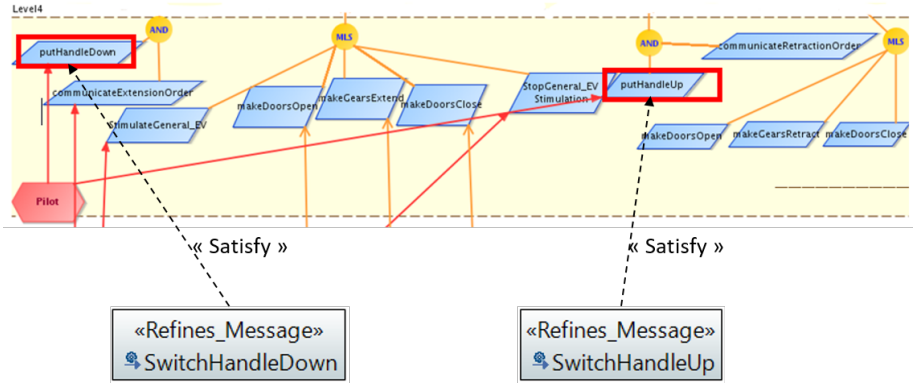

Figure 2.42 – Example of graphical alignment links of the landing gear system case study

The Event-B formalization of these alignment links presented in Section 4.3 is processed after generating the Event-B specifications of the SysML/KAOS models and of the SysML HLA models.

The formalization of the goal putHandleDown alignment with the message SwitchHan-

dleDown is established as follows:

- First, a new EVENT-B machine called putHandleDown_Goal_Satisfaction_Interface is created.
- putHandleDown_Goal_Satisfaction_Interface refines PiloteSS_Interface machine presented in Listing 15 that contains the event putHandleDown.
- putHandleDown_Goal_Satisfaction_Interface gets from PiloteSS_Interface machine all variables involved in event putHandleDown (Rule 2 in Table 2.4).
- The goal putHandleDown is *Satisfy* with the message SwitchHandleDown, then the corresponding event are copied from the HLA EVENT-B specification machine PiloteSubSystemL1 presented in Listing 23 with the involved variable (hdState) and its associated typing invariants (Rules 3 and 4 in Table 2.4).
- The gluing invariant must be manually defined (Rule 5). The variables handleState, from the goal machine, and hdState, from the HLA machine, are linked since they represent the same entities but with a different viewpoint (goal or architecture).
- Event SwitchHandleDown refine with a ref keyword the event putHandleDown (Rule 6.1).

Listing 24 presents the EVENT-B specification of the putHandleDown_Goal_Satisfaction_Interface machine responsible for the satisfaction of the leaf goal putHandleDown.

```
REFINEMENT putHandleDown_Goal_Satisfaction_Interface
REFINES PiloteSS_Interface
SEES LandingGearSystem_CONTEXT , LandingGearSystem_CONTEXT_L3 ,
    PiloteSubSystem_CONT , LandingGearSystemL1_CONT ,
    LandingGearSystemL0_CONT
ABSTRACT_VARIABLES handleState , hdState , pssState
INVARIANT
    handleState ∈ Handle → HANDLE_STATES
    /****From goal specification****/ ∧
    hdState ∈ Handle → HandleSTATES
    /****From HLA specification****/ ∧
    pssState ∈ {pss} → PiloteSubSystemStates
    /****From goal specification****/ ∧
    hdState[HANDLE]=down ⇒ handleState [HANDLE]=down
    //gluing Invariant. ∧
    hdState[HANDLE]=up ⇒ handleState [HANDLE]=up
```

```
    //gluing Invariant.
INITIALISATION
    hdState, handleState:(
    handleState ∈ Handle → HANDLE_STATES
    /****From goal specification****/ ∧
    hdState ∈ { hd } → HandleSTATES
    /****From HLA specification****/ ∧
    hdState[HANDLE]=down ⇒ handleState [HANDLE]=down
    //gluing Invariant. ∧
    hdState[HANDLE]=up ⇒ handleState [HANDLE]=up
    //gluing Invariant.) ∧
    pssState :∈ {pss} → PiloteSubSystemStates
    /****From goal specification****/
EVENTS
    SwitchHandleDown ref putHandleDown =
    SELECT hdState ( hd )= up ∧
    pssState ( pss )= orderedForRetractionPiloteSS THEN
        hdState ( hd ):= down ||
        pssState ( pss ):= orderedForExtensionPiloteSS ||
        handleState ( LgOfHd ~( LG )):= down
    END
END
```

Listing 24 – EVENT-B formalization of the goal putHandleDown alignment link.

For the goal putHandleUp aligned with the message SwitchHandleUp, a second EVENT-B machine called putHandleUp_Goal_Satisfaction_Interface is created (Rule 1 in Table 2.4) as shown in Listing 25. The machine is defined as follows:

— putHandleUp_Goal_Satisfaction_Interface refines PiloteSS_Interface machine presented in Listing 15 that contains the event putHandleUp.

— putHandleUp_Goal_Satisfaction_Interface gets from PiloteSS_Interface machine all variables involved in event putHandleUp (Rule 2 in Table 2.4).

— putHandleUp is *Satisfy* with the message SwitchHandleUp. Therefore, we get from the HLA EVENT-B this event with all involved variables (hdState) and their associated typing invariant. (Rules 3 and 4 in Table 2.1)

— The gluing invariant is the same as in the previous case.

— Event SwitchHandleUp refines with a ref keyword the event putHandleUp (Rule 6.1).

```
REFINEMENT putHandleUp_Goal_Satisfaction_Interface
REFINES PiloteSS_Interface
SEES LandingGearSystem_CONTEXT , LandingGearSystem_CONTEXT_L3 ,
    PiloteSubSystem_CONT , LandingGearSystemL1_CONT ,
    LandingGearSystemL0_CONT
ABSTRACT_VARIABLES handleState , hdState , pssState
INVARIANT
    handleState ∈ Handle → HANDLE_STATES
    /****From goal specification****/ ∧
    hdState ∈ Handle → HandleSTATES
    /****From HLA specification****/ ∧
    pssState ∈ {pss} → PiloteSubSystemStates
    /****From goal specification****/ ∧
    hdState[HANDLE]=down ⇒ handleState [HANDLE]=down
    //gluing Invariant. ∧
    hdState[HANDLE]=up ⇒ handleState [HANDLE]=up
    //gluing Invariant.
INITIALISATION
    hdState , handleState :(
    handleState ∈ Handle → HANDLE_STATES
    /****From goal specification****/ ∧
    hdState ∈ { hd } → HandleSTATES
    /****From HLA specification****/ ∧
    hdState[HANDLE]=down ⇒ handleState [HANDLE]=down
    //gluing Invariant. ∧
    hdState[HANDLE]=up ⇒ handleState [HANDLE]=up
    //gluing Invariant.) ∧
    pssState :∈ {pss} → PiloteSubSystemStates
    /****From goal specification****/
EVENTS
    SwitchHandleUp ref putHandleUp =
    SELECT hdState ( hd )= down ∧
    pssState ( pss )= orderedForExtensionPiloteSS THEN
        hdState ( hd ):= up ||
        pssState ( pss ):= orderedForRetractionPiloteSS ||
```

```
        handleState ( LgOfHd ~( LG )):= up
    END
END
```

Listing 25 – EVENT-B formalization of the goal putHandleUp alignment link.

5.4.2 Example of alignment between train control system SysML/KAOS models and SysML HLA models

5.4.2.1 Train control system SysML/KAOS requirements model

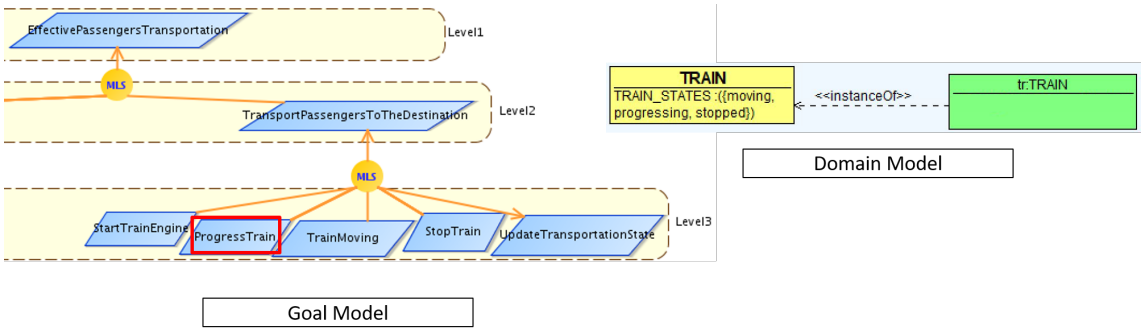

Figure 2.43 – Train Control system goal and domain models

In this example, we present an extract of the goal model of the case study with the domain model corresponding to Level 3 shown in Figure 2.43. The main goal is EffectivePassengersTransportation which is the high level system objective. This goal is refined with a milestone into TransportPassengersToTheDestination and other goals. Goal StartTrainEngine corresponds to starting the train engines. The goal ProgressTrain aims to set the train mode to moving. The goal TrainMoving is responsible for moving the train and StopTrain aims to stop the train. The goal UpdateTransportationState allows to update information corresponding to the transportation mission. These goals refine with a milestone TransportPassengersToTheDestination. In this example we are just dealing with the leaf goal: ProgressTrain.

```
SYSTEM TrainControllerL3_CONT
SETS DESTINATIONS; TRAIN_ENGINE_STATES;  TRAIN_STATES;
CONSTANTS progressing, moving, stopped, ...
PROPERTIES
    ...
```

```
    TRAIN_STATES={progressing, moving, stopped}
END
```

Listing 26 – Extract from the Event-B specification context of the Train Control system SYSML/KAOS Level 3 requirements model.

```
REFINEMENT TrainControllerL3
REFINES TrainControllerL2
SEES TrainControllerL1_CONT, TrainControllerL2_CONT,
    TrainControllerL3_CONT
ABSTRACT_VARIABLES trainState, ...
INVARIANT
    trainState ∈ TRAIN → TRAIN_STATES ∧
    ...
INITIALISATION
    trainState :∈ TRAIN → {stopped} || ...
EVENTS
    ...
    ProgressTrain ref_milestone TransportPassengersToTheDestination =
    SELECT trainState(tr) = stopped THEN
    trainState(tr) := progressing
    END;
    ...
    StopTrain ref_milestone TransportPassengersToTheDestination =
    SELECT trainState(tr) = moving THEN
    trainState(tr) := stopped
    END
END
```

Listing 27 – Extract from the Event-B specification machine of the Train Control system SYSML/KAOS Level 3 requirements model.

An extract of the Event-B formalization of the SysML/KAOS models of Figure 2.43 is shown in Listings 26 and 27. Following the formalization approaches presented in [Matoussi et al., 2011a, Tueno Fotso et al., 2018], the goal model is composed of many

levels. Each level is transformed into a machine in the Event-B specification of the SysML/KAOS models. Listings 26 and 27 show an extract of the translation of level 3 of the goal model into an Event-B specification. CONTEXTs, VARIABLES and INVARIANTS representing the structural part of this specification are obtained from the SysML/KAOS domain model, TRAIN, TRAIN_STATES, etc. Each goal of this level (StartTrainEngine, ProgressTrain, TrainMoving, StopTrain and UpdateTransportationState) is formalized into an Event-B event which defines the dynamic part of the system. The milestone refinement link between the previously presented goals and their parent goal TransportPassengersToTheDestination is represented in the Event-B specification by the ref_milestone keyword that replaces the Event-B ref keyword.

5.4.2.2 Train control system SysML HLA models

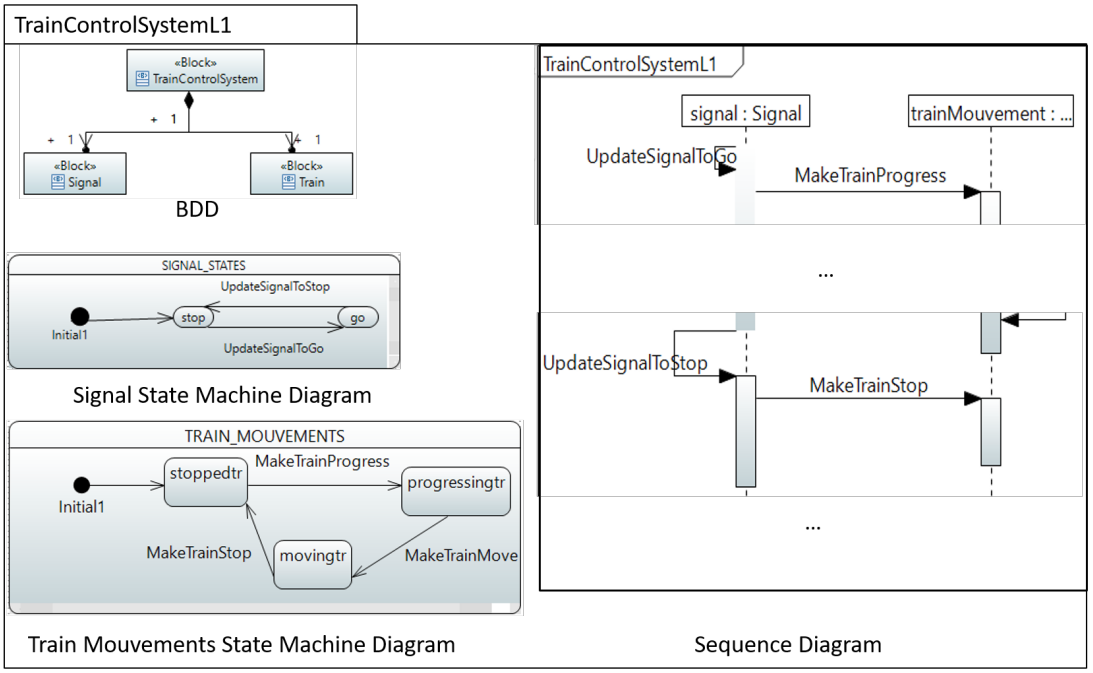

Figure 2.44 – Extract from the Train Control system HLA model

An extract of the HLA model level 1 of the train control system case study is presented in Figure 2.44. The main system TrainControlSystem is composed of two components Signal and Train described in the BDD. The behavior of each component is described by a state-machine diagram. The sequence diagram describes the interplay between the components. In this sequence diagram, the process begins when the signal passes from stop to go. Then the train starts progressing. After progressing, the train moves by proceeding in a cycle of

acceleration and deceleration movements (not shown in the figure). When the train arrives at destination, the signal passes from go to stop and the train should be stopped.

The EVENT-B formalization of this extract of the train control system HLA model is represented in Listings 28 and 29. As already presented in Section 3.2, each package is translated into an EVENT-B machine and a context seen by the machine. In our example, the package TrainControlSystemL1 is translated into the TrainControlSystemL1 machine and the TrainControlSystemL1_CONT context that contains sets (TRAINS, SIGNAL_STATES, etc.) created from the BDD blocks and their associated state-machine diagrams. The TrainControlSystemL1 machine refines TrainControlSystemL0 machine and sees all its contexts. States of the state diagrams are represented as EVENT-B constants in the context. Sequence diagram lifelines are translated into EVENT-B constants, variables and invariants. Finally, sequence diagrams messages are translated into events in the EVENT-B machine.

```
SYSTEM  TrainControlSystemL1_CONT
SETS TRAINS; SIGNAL_STATES; TRAIN_MOUVEMENTS
CONSTANTS stop, go, ..., movingtr, progressingtr, stoppedtr
PROPERTIES
    ...
    SIGNAL_STATES = {stop, go} ∧
    TRAIN_MOUVEMENTS = { movingtr , progressingtr , stoppedtr}
END
```

Listing 28 – Extract from the EVENT-B specification context of the Train Control system HLA level 1.

```
REFINEMENT TrainControlSystemL1
REFINES TrainControlSystemL0
SEES TrainControlSystemL1_CONT, TrainControlSystemL0_CONT
ABSTRACT_VARIABLES signalState, trainMouvementState
INVARIANT
    signalState ∈ SIGNAL_STATES ∧
    trainMouvementState ∈ TRAINS → TRAIN_MOUVEMENTS
INITIALISATION
    signalState :∈ {stop} ||
    trainMouvementState :∈ TRAINS → {stoppedtr}
EVENTS
```

```
UpdateSignalToGo =
SELECT signalState = stop  ∧
trainMouvementState(train)=stoppedtr THEN
signalState := go
END;
MakeTrainProgress =
SELECT signalState = go  ∧
trainMouvementState(train)=stoppedtr THEN
trainMouvementState(train):=progressingtr
END;
...
MakeTrainStop =
SELECT signalState = stop ∧
trainMouvementState(train)=movingtr THEN
trainMouvementState(train):=stoppedtr
END;
...
END
```

Listing 29 – Extract from the EVENT-B specification machine of the Train Control system HLA level 1.

5.4.2.3 Train control system SysML/KAOS models and SysML HLA models alignment

Here, we are interested to align the goal ProgressTrain from goal diagram presented in Figure 2.43 with messages of the HLA model presented in Figure 2.44. Figure 2.45 describes the two alignment links. To satisfy the goal ProgressTrain, the sequential execution of the two sequence diagram messages UpdateSignalToGo and MakeTrainProgress is required. The arrows from the messages to the goal are labelled by the stereotype *Milestone_Satisfy*. Recall that the execution order is determined by the numbers associated to the arrow.

The Event-B formalization of these alignment links follows the defined rules in Section 4.3. To align the goal ProgressTrain with the HLA sequence diagram messages UpdateSignalToGo and MakeTrainProgress, we proceed as follows:

— First, a new Event-B machine called ProgressTrain_Goal_Satisfaction_Interface is

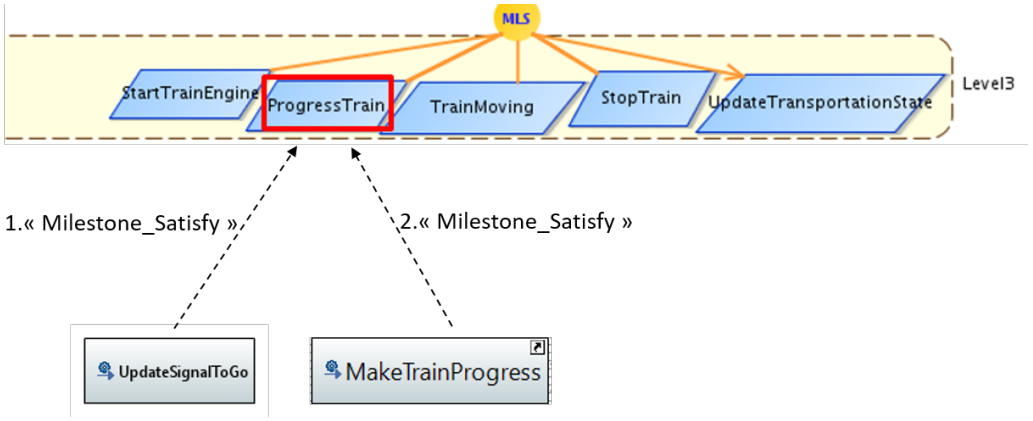

Figure 2.45 – Graphical alignment links of ProgressTrain Goal

created. (Rule 1 in Table 2.4).

— Then, ProgressTrain_Goal_Satisfaction_Interface refines TrainControllerL3 machine that contains the event ProgressTrain.

— ProgressTrain_Goal_Satisfaction_Interface gets from TrainControllerL3 machine (see Listing 27) all variables involved in event ProgressTrain (Rule 2 in Table 2.4).

— As the goal ProgressTrain is *Milestone_Satisfy* with the messages UpdateSignalToGo and MakeTrainProgress, the corresponding events are copied from the HLA Event-B specification (see Listing 29) with all the involved variables (signalState, TrainMouvementState) and their associated typing invariants (Rules 3 and 4 in Table 2.1).

— The gluing invariant must be manually defined (Rule 5). It is used to link the variables trainState, from the goal machine, and trainMouvementState, from the HLA machine, since they represent the same entities but with a different viewpoint (goal or architecture).

— Events UpdateSignalToGo and MakeTrainProgress refine with a ref_milestone keyword the event ProgressTrain (Rule 6.3).

Listing 30 presents the Event-B specification of the ProgressTrain_Goal_Satisfaction_Interface machine responsible for the satisfaction of the leaf goal ProgressTrain.

```
REFINEMENT ProgressTrain_Goal_Satisfaction_Interface
REFINES TrainControllerL3
SEES TrainControllerL1_CONT, TrainControllerL2_CONT,
TrainControllerL3_CONT, TrainControlSystemL1_CONT
ABSTRACT_VARIABLES ..., trainState, signalState, trainMouvementState
INVARIANT
```

```
    signalState ∈ SIGNAL_STATES ∧
    trainMouvementState ∈ TRAINS → TRAIN_MOUVEMENTS ∧
    (trainMouvementState[TRAINS] ={stoppedtr} ⇒
    trainState[TRAIN]={stopped}) /****gluing invariant****/ ∧

    (trainMouvementState[TRAINS] ={progressingtr} ⇒
    trainState[TRAIN]={progressing})/****gluing invariant****/ ∧

    (trainMouvementState[TRAINS] ={movingtr} ⇒
    trainState[TRAIN]={moving}) /****gluing invariant****/
INITIALISATION
    ...
    signalState :∈ {stop} ||
    trainState, trainMouvementState : (
    (trainState ∈ TRAIN → {stopped}) ∧
    (trainMouvementState :∈ TRAINS → {stoppedtr}) ∧
    (trainMouvementState[TRAINS] ={stoppedtr} ⇒
    trainState[TRAIN]={stopped})/****gluing invariant****/  ∧
    (trainMouvementState[TRAINS] ={progressingtr} ⇒
    trainState[TRAIN]={progressing})/****gluing invariant****/  ∧
    (trainMouvementState[TRAINS] ={movingtr} ⇒
    trainState[TRAIN]={moving})/****gluing invariant****/
    )
EVENTS
    UpdateSignalToGo ref_milestone ProgressTrain =
    SELECT signalState = stop ∧
    trainMouvementState(train)=stoppedtr ∧
    trainState(tr)=stopped THEN
    signalState := go
    END;
    MakeTrainProgress ref_milestone ProgressTrain =
    SELECT signalState = go ∧
    trainMouvementState(train)=stoppedtr THEN
    trainMouvementState(train):=progressingtr ||
    trainState(tr):=progressing
    END
END
```

Listing 30 – EVENT-B formalization of the goal ProgressTrain alignment link.

The ProgressTrain_Goal_Satisfaction_Interface machine was introduced in AtelierB to be formally verified and in ProB to animate the execution scenarios. All the proof obligations have been automatically discharged, including the refinement, invariant preservation, non-deterministic action feasibility and well-definedness proof obligations. ProgressTrain_Goal_Satisfaction_Interface required to discharge 11 proof obligations. Figure 2.46 summarizes the proof activity, where **PO** means generated proof obligations, **UN** is for unproved proofs and **PR** for the proved ones.

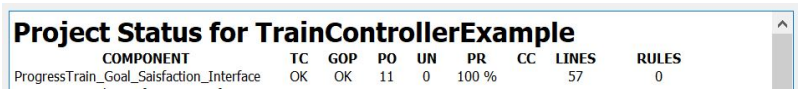

Figure 2.46 – Proof obligations table

5.4.3 Conclusion

In this section, we have presented the use of the proposed model-based approach to align complex systems HLA models with SysML/KAOS requirements models on two extracts of case studies. The first example is taken from the landing gear system case study in which the use of alignment kind *Satisfy* is described between two leaf goals from the SysML/KAOS goal model each of which is aligned with only one HLA sequence diagram message. The second example is about an extract from the train control system case study where a presented leaf goal requires the sequential execution in a specific order of more than one HLA sequence diagram message. This implies the use of *Milestone_Satisfy* alignment kind. The Event-B formalization of these alignment links is produced following the defined rules. The semantics of the alignments links are given by these rules and the definition of new proof obligations can be discharged using AtelierB.

6 Conclusion

This chapter describes the methodology we propose that consists of a holistic process to define system HLA aligned with system requirements. First, we have presented the SysML/KAOS approach dealing with requirements modeling and their formal represen-

tation to give a consistent stakeholders goals definition. Second, a HLA modeling approach is proposed based on SysML extensions with safety relevant Event-B mechanisms which are refinement and decomposition mechanisms. These extensions enable an automatic translation from SysML HLA models to Event-B specifications for the purpose to formally verify the consistency of the modeled HLA. Third, we have proposed a model-based approach that defines alignment links between SysML/KAOS models and SysML HLA models and their formalization into Event-B specification. Finally, an illustration of these works is presented based on two case studies, the landing gear system and the train control system

The next chapter gives more details on the implementation of the proposed methodology.

Chapter III

Implementation

Contents

The previous chapter describes the high-level architecture modeling aligned with system requirements methodology, which is a three-steps methodology including the modeling and

formal verification of system requirements, the process of modeling HLA using SysML extensions and its formalization into Event-B specification and finally the alignment method between HLA elements with system requirements. This chapter elaborates on the implementation of this methodology which is based essentially model-driven engineering and specifically model transformation technologies.

The remainder of this chapter is organized as follows: Section 1 describes some preliminaries required to the implementation of our methodology. Section 2 gives an overview about the implementation of the methodology. Finally, Section 3 presents our conclusion.

1 Preliminaries

1.1 Model Transformation

Model transformations are at the heart of the Model-Driven Engineering approach. However there is still no consensus on how to define and implement a transformation. In the literature, multiple approaches are proposed. To perform model transformations, it should be expressed in a certain modeling language or meta-model. Starting from the source and target meta-models involved in the transformation, two types of transformations can be distinguished: endogenous and exogenous transformations. A transformation is called endogenous if the models involved are conformed to the same meta-model. When the source and target models are of different meta-models, the transformation is said exogenous or even translation.

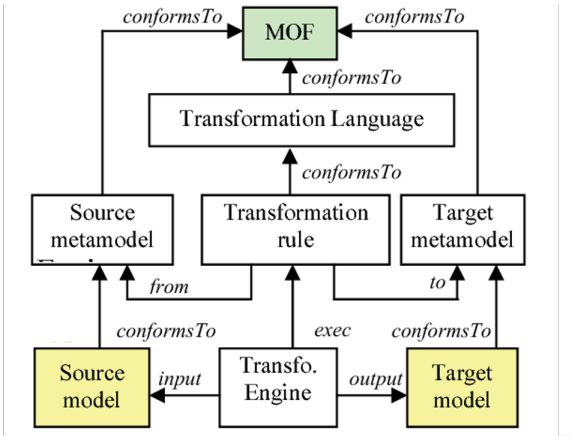

Figure 3.1 – Transformation process [Hammoudi et al., 2008]

Figure 3.1 shows the model transformation process. As shown a model transformation

is defined by a transformation engine that executes a set of transformation rules conforms to an transformation language which allows to transform a source model (conforming to its source meta-model) into a target model (conforming to its target meta-model).

Several transformation languages have been proposed for performing model transformations using the meta-modeling approach, including the MOF2.0 QVT 4 standard, the Kermeta language [Muller et al., 2005] and the ATL language [Bézivin et al., 2003, Bézivin et al., 2005]. In this thesis, we chose the QVT language. Indeed, Query View Transformation (QVT) is a defacto standard specification for model transformation standard published by OMG [Barendrecht, 2010]. Particularly, we focus on QVT-Operational (QVTo), introduced as part of the MOF (Meta Object Facility) standard (OMG, 2011).

1.2 Query View Transformation (QVT)

This transformation language, presented in [Barendrecht, 2010], is used to transform models. The name of the language suggests a three-part structure. The first part is named *Query* because queries can be applied to a source model, an instance of the source meta-model. The second is the *View*, which is a description of how the target model should look like. The third part is the *Transformation*, which is the part where the results of the queries are projected on the view, and thereby creating the target model.

The QVT specification is based on MOF [OMG., 2006] and OCL (Object Constraint Language) [OMG, 2014]. OCL is a formal language originally used to describe constraints in UML models. Nowadays, it can be used with any MOF meta-model.

Transformation techniques can be of three types: declarative, imperative and hybrid (both declarative and imperative). In this thesis, we are interested in the imperative approach, since it is close to programming languages. It consists of going through the source model in a certain order and generating the target model during this course. QVTo - Operational Mappings Language is specified as a standard way to provide imperative implementations. It provides OCL extensions with side effects that allow a more procedural style, and a concrete syntax that looks familiar to imperative programmers. Mappings Operations can be used to implement one or more relations when it is difficult to provide a purely declarative specification of how a relation is to be populated. Mappings Operations can also invokes other Mappings Operations to create the correspondence between source and target model elements. A transformation entirely written using Mapping Operations is called an operational transformation.

2 Overview of the methodology implementation

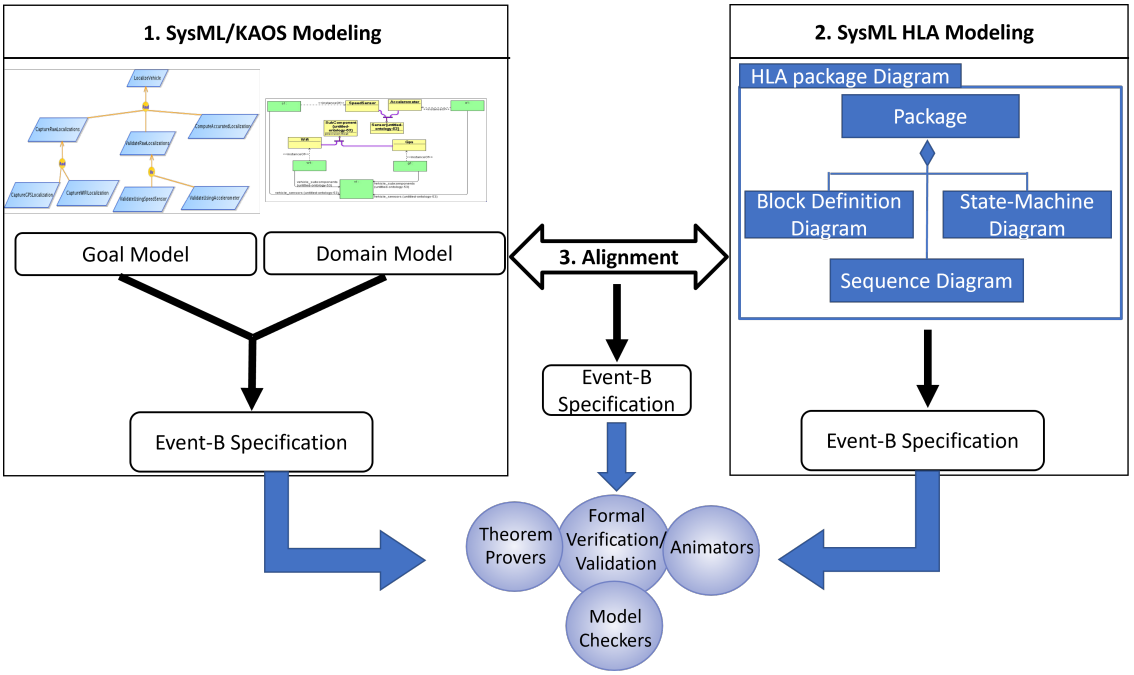

Figure 3.2 – Methodology overview

Our methodology shown in Figure 3.2 is implemented following a set of steps that allow the formal alignment between system requirements and HLA models. As we already presented, this methodology is model-based then we present its implementation as a collaboration of a set of tools and Eclipse plugins that allows the representation of requirements and its formalization, HLA models and their translation into formal specification and traceability establishing between these two parts. In fact our methodology specification process is composed of three parts: requirements, HLA and alignment parts.

2.1 Requirements specification and formalization part

The first step that a user of our methodology should start with, is to specify requirements by defining functional goals and corresponding domain models. After that, an Event-B formalization is produced from requirements models. To this end, we use the Formose tool [Openflexo, 2019] which is an Openflexo open-source platform that federates all the models involved in the SysML/KAOS requirements engineering method. OWLGrEd [Bārzdiņš et al., 2010] is a visual editor for OWL 2.0 [Sengupta and Hitzler, 2014] ontologies that combines UML class diagram notation and textual OWL expressions. They are used to define domain models associated to goal models. Finally, an Event-B formal-

ization of these models is established using AtelierB [AtelierB, 1990] which is an industrial tool that allows for the operational use of the B Method to develop defect-free proven software (formal software).

We limit our use of FORMOD tool only on SysML/KAOS goal model construction because it gives a good representation of goals and their hierarchy. However, this tool is not very stable yet and more complicated to use for modeling domain models and the translation into B system. We use OWLGrEd editor to define domain models associated with the goal models in order to give a better understanding about entities and concepts that participate in elaborating goals.

2.1.1 Usage scenario

The use of the methodology is applied on the case study that we presented in section 5.1. The user starts by designing the SysML/KAOS functional goal model, using the SysML/KAOS FORMOD tool. he should proceed as follows:

— First, the user should create the system goal model according to the SysML/KAOS goal modeling language.

Figure 3.3 shows the goal model defined with two levels of refinement: the root level Level 0 which introduces the main goal maneuverLandingGear and the Level 1 level which describes the refinement of the main goal in two sub-goals (maneuverLGforLanding and maneuverLGforTakingOff). More details about the use of FORMOD can be found in [Tueno Fotso, 2019].

— Second, after modeling the SysML/KAOS goal model, the user can use OWLGrEd to define the domain models associated with each goal model level.

Figure 3.4 shows the domain model associated to the root level of the goal diagram shown in Figure 3.3. This domain model represents the landing gear system entity modeled as an instance of the concept named LandingGearSystem. The possible states of a landing gear system are modeled as an instances of the attribute named LG_STATES, which contains two instances of DataValue of type STRING: lg_extended for the extended state and lg_retracted for the retracted state. LG is modeled as an instance of an individual named LG individual of LandingGearSystem.

— Finally, once the goal model and its corresponding domain models are designed, the user can formalize these models into Event-B using the AtelierB tool.

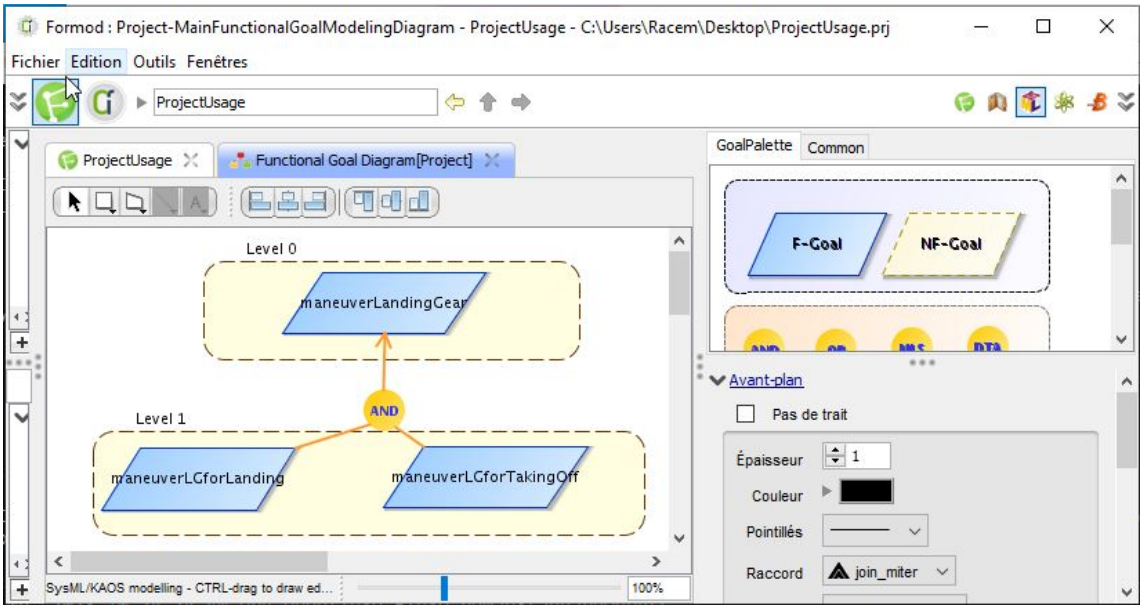

Figure 3.3 – SysML/KAOS Goal modeling

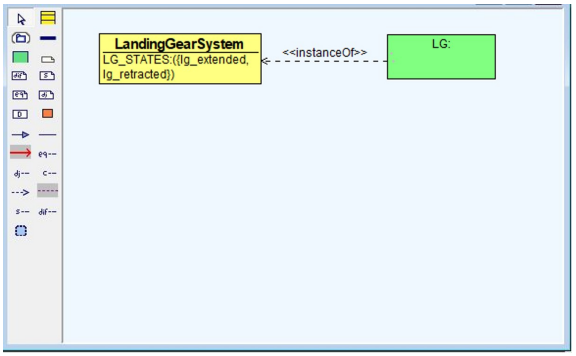

Figure 3.4 – Domain modeling

Figure 3.5 presents the Event-B specification machine corresponding to the root level of the goal model and its seen Event-B specification context corresponding to domain model (shown in Figure 3.4). Once the Event-B specification is completed, a set of proof obligations is generated and the user can discharge them. The discharging of these proof obligations is established using the interactive proof tool of AtelierB shown in Figure 3.6. The user can also check the proof obligations status and their percentages as shown in Figure 3.7.

2.2 HLA modeling and formalization

The second step of this methodology is about modeling HLA and its formalization into Event-B. First a HLA modeling of the corresponding system should be produced and then

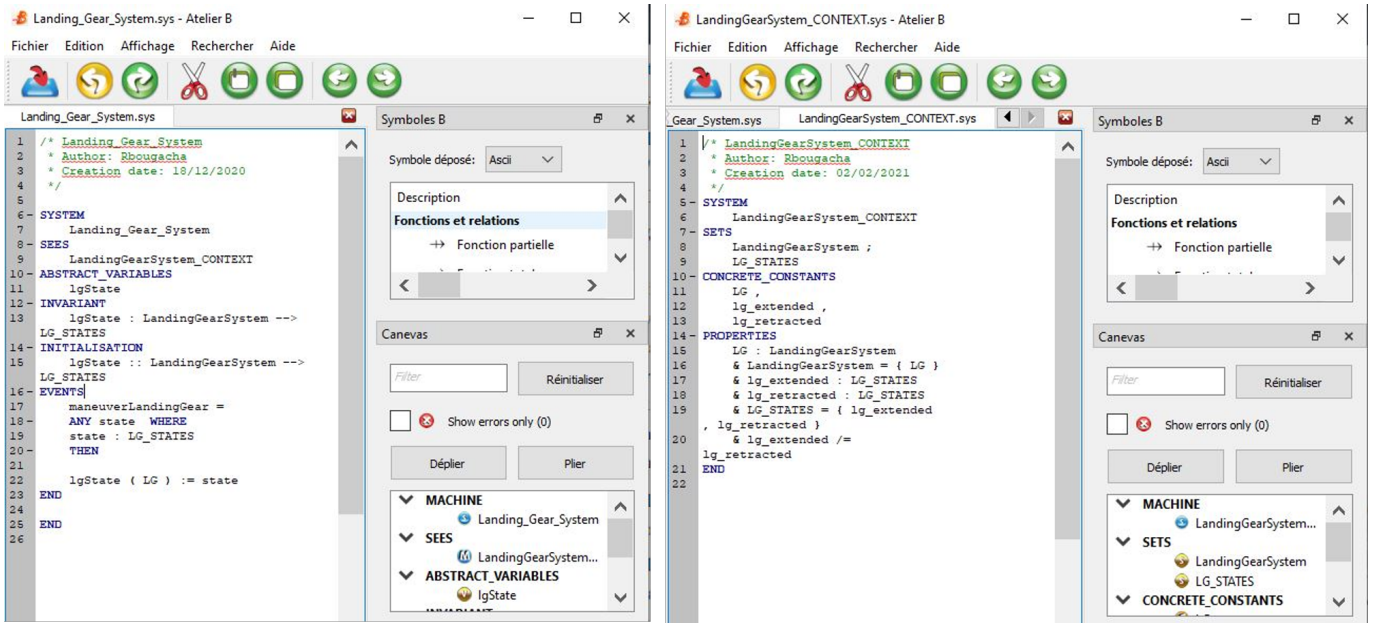

Figure 3.5 – AtelierB Event-B specification

an automatic Event-B translation of these models is executed. To this end, we used the Eclipse development IDE (Integrated Development Environment) [Eclipse, 2009] in which we integrated Papyrus Modeling Environment [Papyrus, 2008] that provides a graphical editing tool and extension mechanisms. We used also Eclipse Modeling Framework (EMF) [Steinberg et al., 2008] to create meta-models. Finally, the HLA models translation into Event-B is implemented using the QVT transformation language.

2.2.1 Eclipse Modeling Framework (EMF)

The EMF project is a modeling framework and code generation facility for building tools. Using MDE, it allows to develop and manage the whole application life cycle based on the transformation of models which are conformed to meta-models. Then, the model is used to generate software artifacts, which will implement the real system. From a model specification described in XMI [1], EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

Its fundamental concepts are:

— EMF Core: The core EMF framework includes a meta-model (Ecore) that describes

1. XMI, XML-based Metadata Interchange, is an interchange format for metadata defined in terms of the MOF standard. In addition to supporting the exchange of complete models, XMI supports the exchange of models in differential form

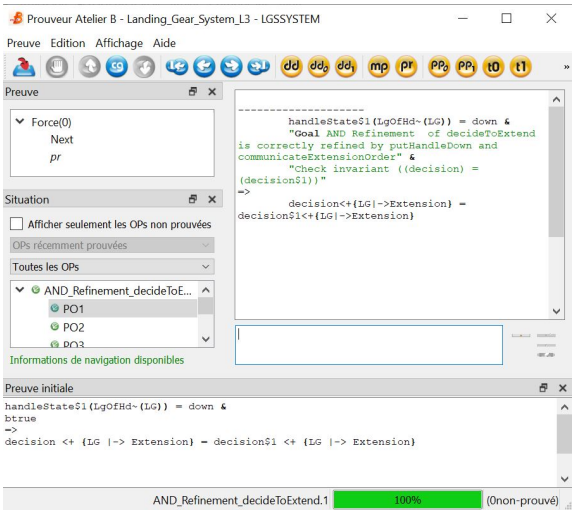

Figure 3.6 – AtelierB interactive proof tool

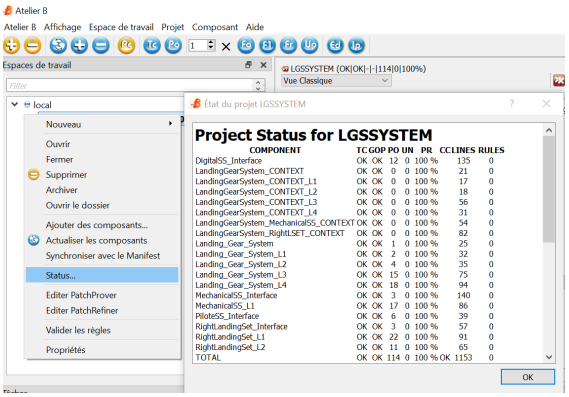

Figure 3.7 – AtelierB proof status

models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.

— EMF.Edit: the EMF.Edit framework includes generic reusable classes for building editors for EMF models. It provides content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop viewers and property sheets.

— EMF.Codegen: The EMF code generation facility is capable to generate everything needed to build a complete editor for an EMF model. It includes a user guide interface from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse.

The most advantage of using EMF is that it is free and open source. It can generate efficient, and easy implementation code from a model specification. Besides, models in EMF can be defined in different ways, for example, Java Interfaces, the UML Class Diagram, the XML Schema, etc...

2.2.2 Papyrus Modeling Environment Overview

Papyrus [Papyrus, 2008] is an environment for editing any kind of EMF models, particularly supporting UML 2 (Unified Modeling Language (UML) version 2.4.1) [OMG, 1997] and related modeling languages such as SysML [OMG, 2007] and MARTE [OMG,

2008]. Papyrus also offers very advanced support for UML profiles that enables users to define editors for DSLs (Domain Specific Languages) based on the UML 2 standard.

Papyrus is a collection of plug-ins and features on top of the Eclipse Modeling Framework that allow to provide several techniques:

— **UML**: Papyrus is a graphical editing tool for UML 2 as defined by the OMG.

— SysML: Papyrus supports also SysML in order to enable model-based system engineering.

— **Model execution**: Papyrus can execute models using a rich and extensible animation and simulation framework with MOKA [Papyrus, 2016].

All Papyrus modeling features are designed to be customizable and to maximize reuse. Therefore, it is possible to adapt the standard Papyrus configuration for a specific domain, modeling practice, notation, use the powerful customization mechanisms of Papyrus to adapt the Papyrus modeling environment to satisfy different needs.

2.2.3 UML profiles

The UML profiles [Fuentes-Fernández and Vallecillo-Moreno, 2004] represent the integration of the light-weight mechanism in order to extend the languages based on the MOF. More specifically, profiles are used to customize UML for a specific domain through extension mechanisms that enrich the semantics and syntax of the language. It allows the specification of a MOF model to deal with the specific concepts and notation required in particular application domains (e.g., real-time, business process modeling, finance, etc.) or implementation technologies (such as .NET, J2EE, or CORBA). A profile-based model can be created and manipulated by any tool that supports standard UML.

A stereotype represents the basic functionality to extend UML. It can be considered as a specialization of an existing concept in UML and which offers the possibility of having concepts for the modeling of a specific domain. Stereotypes can have attributes (also called tagged values) and can be associated with other existing stereotypes or other UML concepts. From a notational point of view, stereotypes can give a different graphical symbol for the elements of the UML model. For example, a class stereotyped Clock might use an image instead of the format of the regular class. In addition, stereotypes can also be influenced by restrictions expressed by constraints. The definition of constraints in MOF-based languages is OCL.

Papyrus aims to provide an easy-to-use integrated environment for editing EMF type models, in particular it supports UML and related modeling languages such as SysML and MARTE. Papyrus also offers very advanced support for UML profiles which allows users to define editors for DSLs (Domain Specific Language) based on the UML 2 standard.

2.2.4 SysML extensions with Event-B refinement and decomposition mechanisms profiles

SysML extensions proposed in Section 3 support the Event-B refinement and decomposition mechanisms. These extensions are applied on two SysML diagrams which are the package and sequence diagrams. The package diagram is extended with two concepts: *HLA_refines* which define the refinement relationship between two packages and *HLA_decompose* which defines the system/sub-systems decomposition relationship between packages. The sequence diagram is extended with the concept *Refines_Message* which represent the refinement relationship between a sequence diagram and the parent system sequence diagram.

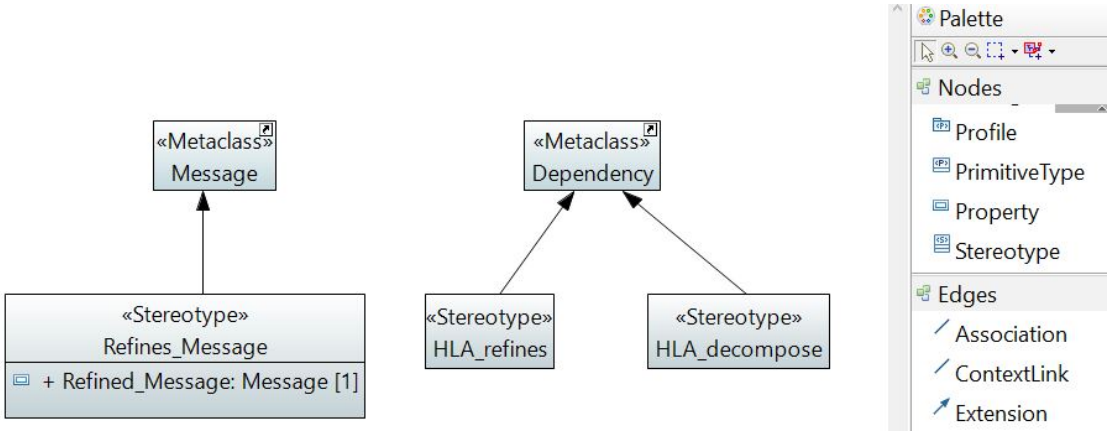

Figure 3.8 – SysML extensions with Event-B refinement and decomposition mechanisms profiles

In our work, we use Papyrus to specify these mechanisms as profiles applied on the package diagram and sequence diagram meta-models. The extensions concepts are defined as stereotypes applied on meta-model meta-classes in order to specialize and enrich them with semantics related to the Event-B refinement and decomposition mechanisms. In fact, these profiles are represented in Figure 3.8. The meta-class *Dependency* is customized with two stereotypes *HLA_refines* and *HLA_decompose* to support the package diagram proposed extensions. In the same way, the sequence diagram message is customized with

the stereotype *Refines_Message* to be distinguished as a refinement message and this stereotype has an attribute called *Refined_Message* of type *Message* to which we will associate the refined message to the *Refines_Message* one.

2.2.5 HLA modeling usage scenario

To graphically model HLA of the case study the user should proceed as follows:

— First, the user should create a new Papyrus project (see Figure 3.9) and give it the type *SysML 1.6* (see Figure 3.10).

— Then the user should add the SysML extensions profiles to the applied profiles of the project (see Figure 3.11).

— After that, the user should model the HLA step by step as illustrated in Section 5.3 by creating the first package and its components then the packages associated to the sub-systems and establish refinement and decomposition customization link between different packages.

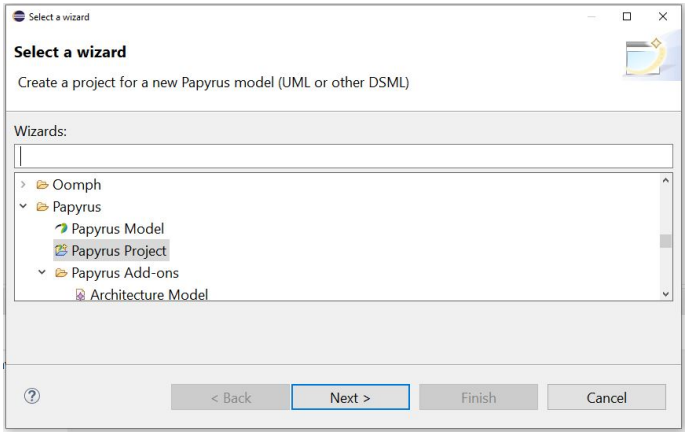

Figure 3.9 – New Papyrus project creation

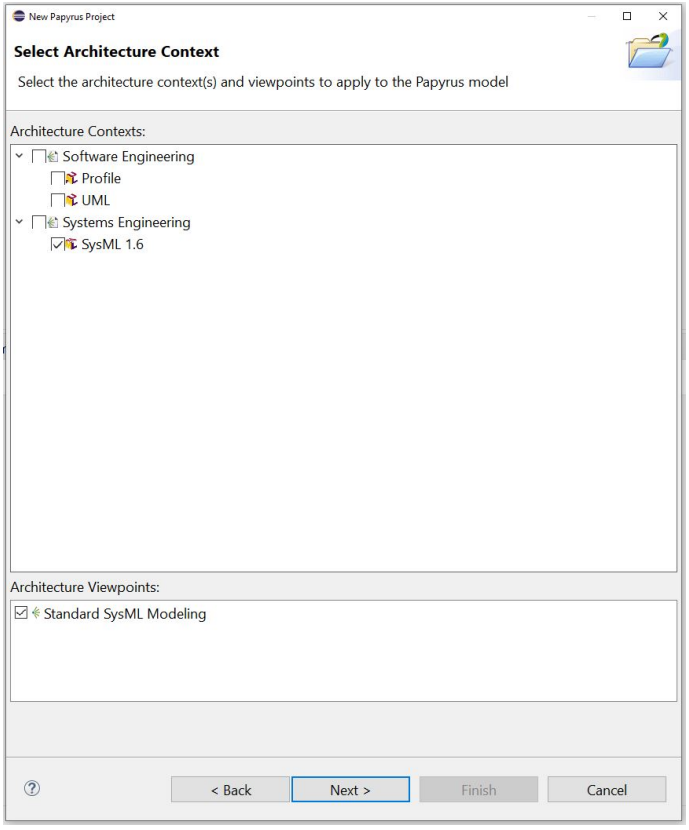

Figure 3.10 – Papyrus project type selection

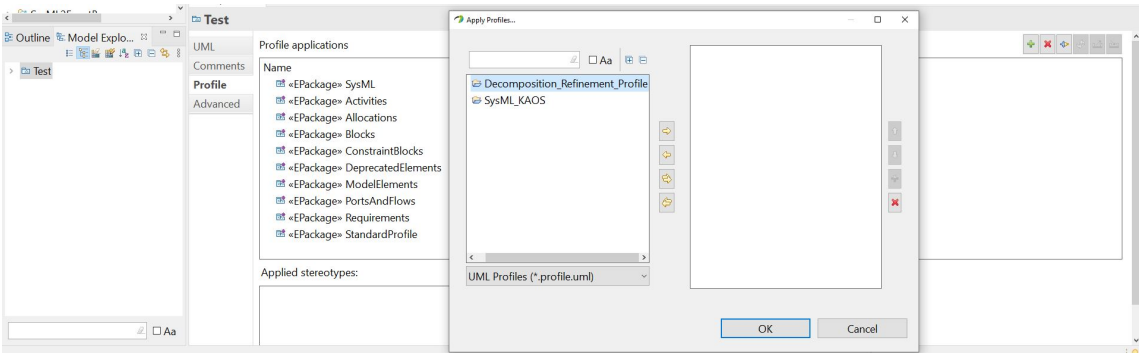

Figure 3.11 – SysML extensions profiles application

2.2.6 HLA formalization into Event-B

2.2.6.1 Event-B meta-model

As already mentioned, model translation is a model-driven process that enables to derive a target model from a source model that are conform to a source and a target meta-models. However, a standardized meta-model for Event-B is still not available. Therefore, to conduct this translation we propose an Event-B meta-model conform to the Event-B notation used in AtelierB and restricted to the presented concepts that are relevant to our use.

EMF is driven by meta-models which can be specified in different formats: Ecore model, Rose .mdl model, XSD Schema, MOF, etc. In our work, we use Ecore models to specify our meta-models. To create our meta-model, we start by creating the new EMF Model File (Ecore File). For that, we need to define the *Name* and the *Namespace URI* relative to the model, which is used to identify it. This is done in the properties view. Then we add a new *Child Element* of type *Class* into the model with the name Event-B_SPEC. As shown in Figure 3.12, Event-B_SPEC presents the root element with a composition link with all other classes. A total of 17 classes with associated relationships with other classes were defined in this meta-model to represent the Event-B notations.

Figure 3.13 exhibits a graphical representation of the EMF model that represents the Event-B meta-model which supports the Event-B notation concepts relevant for specifying HLA such as the SHARED EVENT DECOMPOSITION strategy applied on machines.

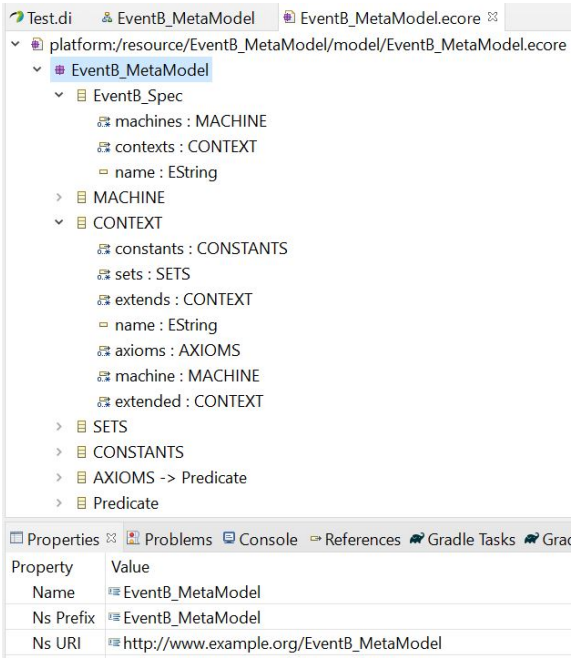

Figure 3.12 – The EMF Model of Event-B Meta-Model

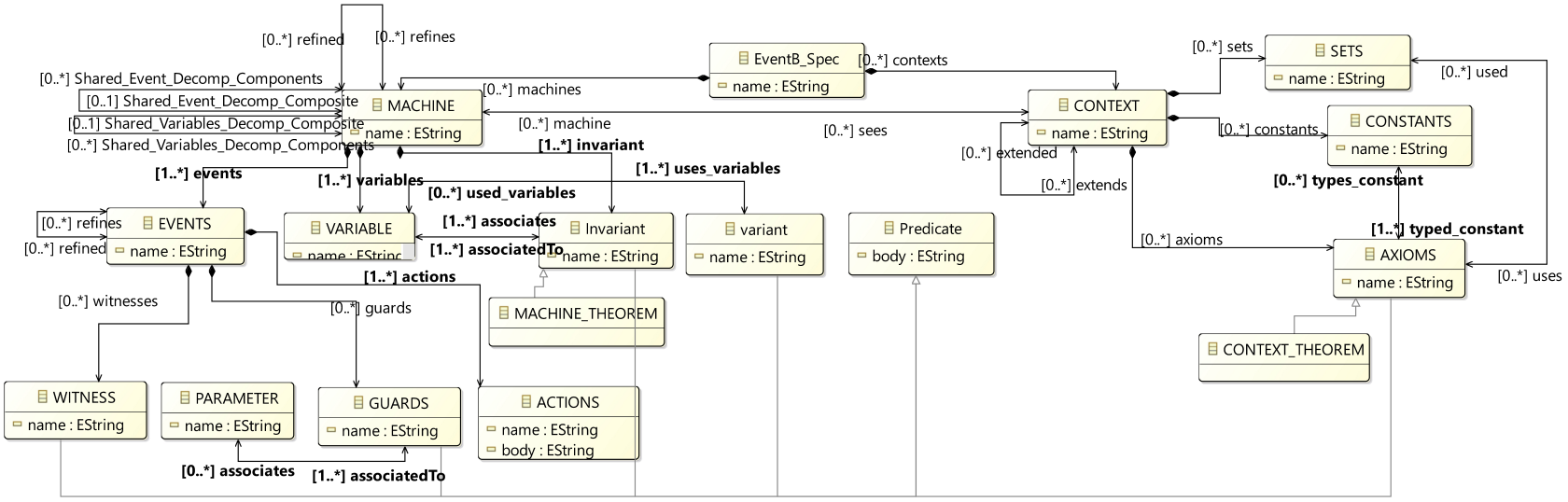

Figure 3.13 – Event-B Meta-Model

2.2.6.2 Extended SysML to Event-B translation: model-to-model transformation.

After the graphical modeling of the system HLA using the extended SysML, the verification and the detection of design errors and the correctness of this HLA are crucial. Consequently, we need to transform extended SysML diagrams used for HLA into Event-B model containing concepts that could generate elements of Event-B notation supported by AtelierB.

To this end, the mapping suggested proposes, at a first step, a model-to-model transformation between SysML meta-model, enriched with refinement and decomposition mechanisms, and the proposed Event-B meta-model. Therefore, our approach opts for model-to-model transformation implemented using QVT transformation language.

To perform this transformation, at first, we need to collect the two meta-models which are based on EMF and contain the meta-classes required for drawing this transformation. Then, we can proceed to define the model-to-model transformation. In the following, we will give an overview on how we implemented our mapping rules. Figure 3.14 presents the transformation architecture of our implementation. It is composed of a main module and three communicating sub-modules dedicated to transform contexts, machines and associated decomposition. Note that, the main module is responsible for the stimulation of the sub-modules.

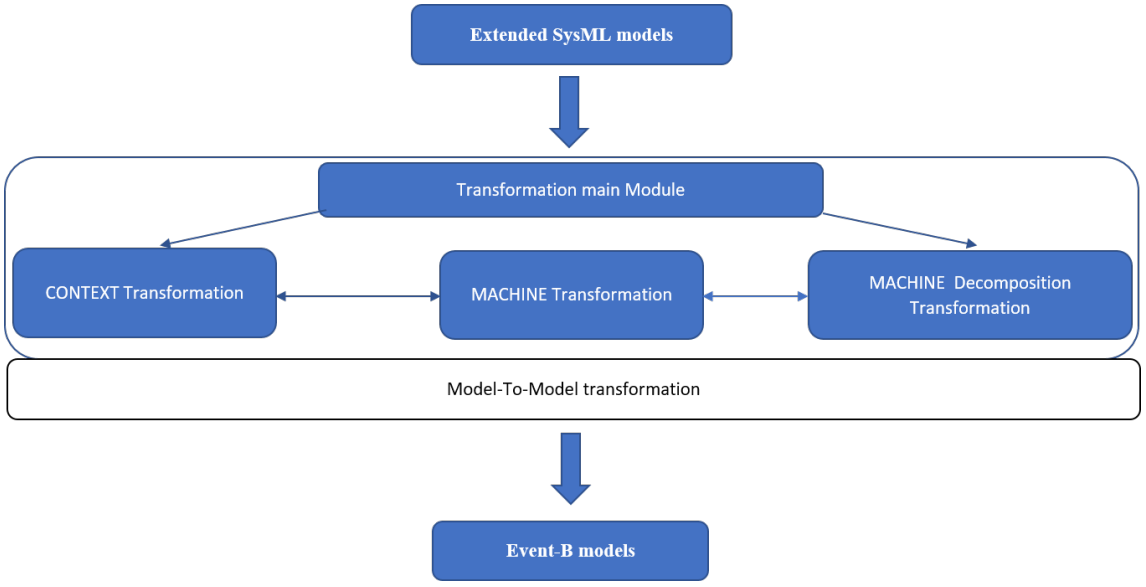

Figure 3.14 – Model-to-model transformation architecture

Listing 31 shows the QVT module header. SysML is a UML profile although it does

not include all of its elements. The part of UML that is reused by SysML is called UML4SysML. As our extended SysML source meta-model inherits from the UML meta-model, we have exploited the principle of *multiple pattern matching* offered by QVT and which allows several source meta-models entrance. Thus, Line 1 of Listing 31 gives the UML meta-model and Lines 2 and 3 give the root of the extended SysML meta-model. These meta-models are uploaded as source meta-models in the main module. Lines 4 and 5 depict the target meta-model which is for this transformation the proposed Event-B meta-model. Lines 7 and 8 define the signature of the transformation which declares the transformation name and the source and target meta-models. in and out keywords indicate source and target model variables. Lines 10 to 13 presents the *Entry point* while the execution of the transformation starts here by executing the instructions in the body of the main operation. In our case, the main operations are called Model2EventBSpec().

```
1  modeltype UML "strict" uses "http://www.eclipse.org/uml2/5.0.0/UML";
2  modeltype SYSML "strict" uses
3  BDD_MetaModel('http://www.example.org/BDD_MetaModel');
4  modeltype EventB "strict" uses
5  EventB_MetaModel('http://www.example.org/EventB_MetaModel');
6
7  transformation SysML2EventB( in uml : UML,
8  in sysml : SYSML, out eventb: EventB);
9
10 main ()
11 {
12     uml.rootObjects()[Model] -> map Model2EventBSpec();
13 }
```

Listing 31 – QVT main module header

Recall that the mapping rules are automatically applied by the QVT transformation engine. Each rule is specified by a name, a set of source patterns mapped to the source elements, a set of target patterns representing the elements created in the target model. A mapping rule is applied once and only once for each source element (match) found in the source models. Listing 32 shows an extract of the implementation of the mapping rule Model2EventBSpec(). This rule takes as input a UML root object of type Model and

allows to create that creates the an Event-B_Spec (see Line 1 of the Listing). It attributes the name of the source element Model to the created target element Event-B_Spec from the EVENT-B meta-model (see Line 2). Model2EventBSpec() also invokes the contexts and machines implementation modules (see lines 3 and 4).

```
1 mapping UML::Model::Model2EventBSpec() : EventB::EventB_Spec {
2     name := self.name;
3     contexts+= self.allSubobjectsOfType(Package)-> map Context() ;
4     machines+= self.allSubobjectsOfType(Package)-> map Machine();
```

Listing 32 – Model2EventBSpec() mapping rule implementation

Listing 33 presents an extract of the implementation of the mapping rule responsible for the transformation of the refinement SYSML extension following these steps:

— First of all, this part of code selects all messages with an applied *Refines_Message* stereotype (Lines 1 and 2).

— Then, for each selected message, it gets the *Refined_Message* attribute value of the stereotype and affect it to a temporary variable that we called *mess*. (see Lines 3 to 8).

Note that, for this part of implementation, refinement between machines is already established.

— After that, we associate the event related to the selected message (*message*) from the machine called MAC with the event associated to *Refined_Message* attribute value (*mess*) from the refined machine *mac._refines*. (see Lines 10 to 12)

— Finally, once the refinement between the two events is established, the event associated to the selected message gets from the refined event each of their actions. (see Lines 13 to 19).

```
1 self.allSubobjectsOfType(Message)
2 ->select(me|me.getAppliedStereotypes()->notEmpty())
3 ->forEach(message){
4     var mess: Message;
5     mess=message.getValue(message.getAppliedStereotype
6     ('Sequence_Diagram_Refinement_Profile::Refines_Message'),
7     'Refined_Message')->selectByType(Message)
```

```
8     ->selectOne(m|m->notEmpty());
9
10    mac.events->selectOne(e|e.name.equalsIgnoreCase(message.name)).
11    _refines:=  mac._refines.events
12    ->select(e|e.name.equalsIgnoreCase(mess.name));
13    mac.events
14    ->selectOne(e|e.name.equalsIgnoreCase(message.name)).
15    _refines.actions->forEach(ra){
16        mac.events
17        ->selectOne(e|e.name.equalsIgnoreCase(message.name)).actions+=
18        ra ->map ActionsAbs2()};
19                    }
```

Listing 33 – Extract from the refinement SysML extension mapping rule implementation

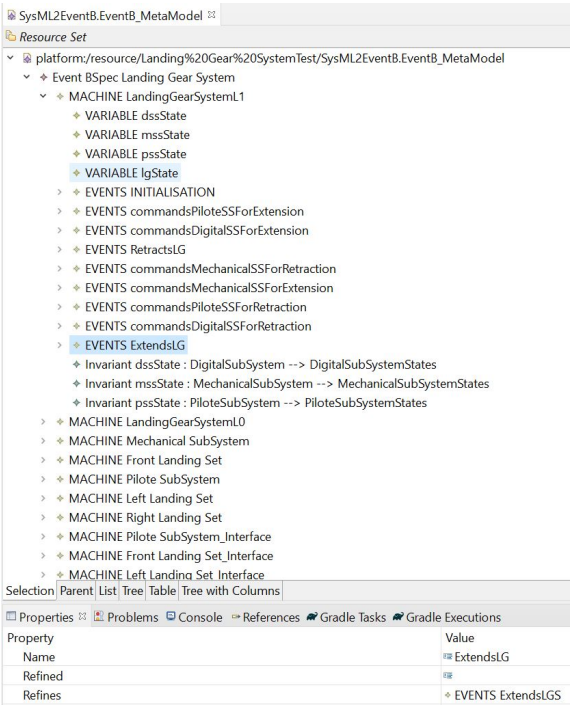

Figure 3.15 – Generated Event-B Model

After running these transformation rules, an Event-B model conform to the Event-B notation defined by the proposed Event-B meta-model is generated. It is shown in Figure 3.15. This Event-B model shows the different machines and contexts generated from HLA and the refinement of the event ExtendsLGS by the event ExtendsLG. We recall that

event ExtendsLG belongs to LandingGearSystemL1 machine which refines LandingGearSystemL0 machine that encompasses the event ExtendsLGS. The model specification behind this graphical representation of EVENT-B model (in Figure 3.15) is described in XMI metadata. An extract of this XMI file is presented in Listing 34. This file will be used as an input for the model-to-text translation step.

```
<?xml version="1.0" encoding="UTF-8"?>
<EventB_MetaModel:EventB_Spec xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:EventB_MetaModel="http://www.example.org/EventB_MetaModel"
name="Landing Gear System">
  <machines name="LandingGearSystemL1"
  sees="//@contexts.0 //@contexts.6"
  refines="//@machines.1"
  Shared_Event_Decomp_Components="//@machines.7 //@machines.10
  //@machines.12">
    <variables name="dssState"
    associatedTo="//@machines.0/@invariant.0"/>
    <variables name="mssState"
    associatedTo="//@machines.0/@invariant.1"/>
    <variables name="pssState"
    associatedTo="//@machines.0/@invariant.2"/>
    <variables name="lgState"/>
    <events name="INITIALISATION">
      <actions
      name="dssState :: {dss} -->DigitalSubSystemStates"
      body="dssState :: {dss} -->DigitalSubSystemStates"/>
      <actions
      name="mssState :: {mss} -->MechanicalSubSystemStates"
      body="mssState :: {mss} -->MechanicalSubSystemStates"/>
      <actions
      name="pssState :: {pss} -->PiloteSubSystemStates"
      body="pssState :: {pss} -->PiloteSubSystemStates"/>
      <actions name="lgState :: {lg} -->LandingGearSystemStates"
      body="lgState :: {lg} -->LandingGearSystemStates"/>
    </events>
    <events name="commandsPiloteSSForExtension">
```

```
    <guards name="lgState(lg)=Retracted"/>
    <guards body="pssState(pss)=orderedForRetractionPiloteSS"
    name="pssState(pss)=orderedForRetractionPiloteSS"/>
    <actions name="pssState(pss):=orderedForExtensionPiloteSS"
    body="pssState(pss):=orderedForExtensionPiloteSS"/>
  </events>
```

Listing 34 – Extract from the refinement SysML extension mapping rule implementation

2.2.6.3 Extended SysML to Event-B translation: model-to-text translation.

The second step suggests, a translation between the Event-B model generated from the first step and a textual specification accepted by the AtelierB tool. This step allows to generate formal specifications for the purpose to be verified. To perform this model-to-text transformation, we implement transformation rules with Acceleo [Musset et al., 2006] transformation language. Acceleo is a template based technology allowing to automatically produce any kind of source code from any data source available in EMF format (XMI).

To perform this transformation, we need to collect the XMI file generated from the previous step (Listing 34). Then, we can proceed to define the model-to-text translation. In the following, we will give an overview on how we implemented our translation rules. Figure 3.16 presents the architecture of the translation composed of three Acceleo templates. The first one is the main template which initiates the translation between Event-B model and Event-B code and invokes contexts and machines creation templates. The *CONTEXT Translation* template is responsible for creating textual specifications for contexts whereas the *MACHINE Translation* template is responsible for creating textual specifications for machines.

Listing 35 presents the translation main template that starts by the declaration of the Event-B meta-model to be used for interpreting the source model (see Line 2). We import the contexts and machines creation templates to invoke the creation operations (Lines 3 and 4). This translation creates a new file with *.sys* extension supported by AtelierB for each founded context in the Event-B model (Lines 8 to 10). After that, it invokes the context creation responsible operation from the context template. Selected context from Event-B model is given as parameter for the operation *aCont.CONTEXT()* (Line 11). In

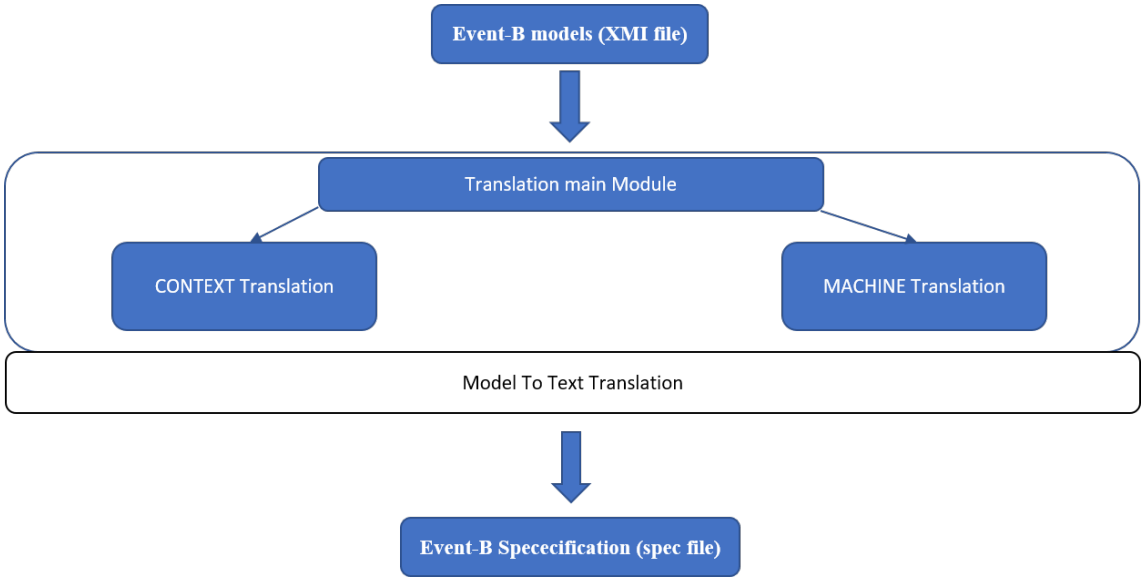

Figure 3.16 – Model-to-text transformation architecture

the same way, not refinement machines are created (Lines 14 to 19). For the refinement ones, the extension supported by AtelierB is *.ref* (Lines 21 to 27).

```
1  [comment encoding = UTF-8 /]
2  [module generate('http://www.example.org/EventB_MetaModel')]
3  [import M2TEventB::main::CONTEXT /]
4  [import M2TEventB::main::MACHINE/]
5
6  [template public generateElement(anEventB_Spec : EventB_Spec)]
7  [comment @main/]
8  [for (aCont : CONTEXT | anEventB_Spec.contexts)]
9  [file (anEventB_Spec.name.concat('/'+aCont.name.replaceAll(' ', '')
10     +'.sys'), false, 'UTF-8')]
11 [aCont.CONTEXT()/]
12 [/file]
13 [/for]
14 [for (aMac : MACHINE | anEventB_Spec.machines)]
15 [if (aMac.refines->isEmpty())]
16 [file (anEventB_Spec.name.concat('/'+aMac.name.replaceAll(' ', '')
17     +'.sys'), false, 'UTF-8')]
18 [aMac.MACHINE()/]
19 [/file]
20 [/if]
21 [if (aMac.refines->notEmpty())]
22 [file (anEventB_Spec.name.concat('/'+aMac.name.replaceAll(' ', '')
```

```
23      +'.ref'), false, 'UTF-8')]
24  [aMac.MACHINE()/]
25  [/file]
26  [/if]
27  [/for]
28  [/template]
```

Listing 35 – Translation main template

Listing 36 presents the context template that shows the structure of the context specification that we would like to be generated. We precise that the main template, shown in Listing 35, invokes this template main operation called *CONTEXT* which takes as a parameter a context (see Line 4 in Listing 36).

The structure of the machine to be generated follows the notation presented in Listing 37 while this template generates the machine with the different Event-B notation clauses namely: SEES, VARIABLES, INVARIANT, EVENTS, etc. The main template, shown in Listing 35, invokes this template main operation called *MACHINE* which takes as a parameter a machine (see Line 4 in Listing 37).

```
1   [comment encoding = UTF-8 /]
2   [module CONTEXT('http://www.example.org/EventB_MetaModel')]
3
4   [template public CONTEXT(aCONTEXT : CONTEXT)]
5   SYSTEM
6           [aCONTEXT.name.replaceAll(' ', '')/]
7   [if (aCONTEXT.extends->notEmpty())]
8   EXTENDS
9       [for(a : CONTEXT | aCONTEXT.extends) separator (';\n') after('\n')]
10          [a.name/][/for]
11  [/if]
12  SETS
13      [for(aset : SETS | aCONTEXT.sets) separator (';\n') after('\n')]
14          [aset.name/][/for]
15  CONSTANTS
16      [for(acons : CONSTANTS | aCONTEXT.constants) separator (',\n')after(
            '\n')]
17          [acons.name/][/for]
18  PROPERTIES
19      [for(aprop : AXIOMS | aCONTEXT.axioms) separator (' &\n')after('\n')
```

```
        ]
20      [aprop.name/][/for]
21  END
22  [/template]
```

Listing 36 – Context template

```
1   [comment encoding = UTF-8 /]
2   [module MACHINE('http://www.example.org/EventB_MetaModel')]
3
4   [template public MACHINE(aMachine : MACHINE)]
5   [if (aMachine.refines->isEmpty())]
6   SYSTEM
7           [aMachine.name.replaceAll(' ', '')/][/if]
8   [if (aMachine.refines->notEmpty())]
9   REFINEMENT
10          [aMachine.name.replaceAll(' ', '')/]
11  REFINES
12      [for(aMac : MACHINE | aMachine.refines) separator(';\n')after('\n')]
13          [aMac.name/][/for]
14  [/if]
15  SEES
16      [for(ac : CONTEXT | aMachine.sees) separator(',\n')after('\n')]
17          [ac.name/][/for]
18  VARIABLES
19      [for(av : VARIABLE | aMachine.variables)separator(',\n')after('\n')]
20          [av.name/][/for]
21  INVARIANT
22      [for(ai :Invariant |aMachine.invariant)separator(' &\n')after('\n')]
23          [ai.name/][/for]
24  [for(ae : EVENTS | aMachine.events->
25  select(e|e.name.equalsIgnoreCase('INITIALISATION'))) separator (' \n')
        after('\n')]
26  INITIALISATION
27      [for(a : ACTIONS | ae.actions) separator (' ||\n') after('\n')]
28          [a.name/][/for]
29  [/for]
30  [if(aMachine.events->notEmpty())]
31  EVENTS
32      [for(ae : EVENTS | aMachine.events->reject(e|e.name.equalsIgnoreCase
```

```
        ('INITIALISATION'))) separator (' ;\n') ]
33      [ae.name/] [if(ae.refines->notEmpty())]ref
34      [for(er : EVENTS | ae.refines) separator (' ,\n')]
35      [er.name/][/for][/if]=
36      [if(ae.guards->notEmpty())]     SELECT
37      [for(g : GUARDS | ae.guards) separator (' &\n') after('\n')]
38      [g.name/][/for]
39      [else]  BEGIN
40      [/if]
41      [if(ae.guards->notEmpty())]     THEN
42      [if (ae.actions->notEmpty())]
43      [for(a : ACTIONS | ae.actions) separator (' ||\n') after('\n')]
44      [a.name/][/for][else] skip [/if]
45      [/if]
46      END
47    [/for]
48  [/if]
49  END
50
51  [/template]
```

Listing 37 – Machine template

2.2.6.4 Extended SysML to Event-B translation: conclusion.

In quantitative terms, our QVT model-to-model transformation is composed of 39 transformation rules (i.e. approximately 600 lines of QVT code) whereas our Acceleo model-to-text translation is about approximately 100 lines of Acceleo code.

As a consistent manner to validate the application of the proposed translations and to verify the correctness of the generated results, we have applied these rules on a set of case studies presented in different publications, i.e. railway systems case studies presented in [Bougacha et al., 2022a], the landing gear system case study presented in [Bougacha et al., 2022b] and the ATO OVER ETCS case study published in [Bon et al., 2022, Bon et al., 2023]. The full Event-B specification resulted from applying the proposed translation on the HLA models of these case studies is presented in [Bougacha, 2022b, Bougacha, 2022a, Bougacha, 2023]. After that, the generated specification is introduced in AtelierB to be formally verified and in ProB to animate the execution scenarios.

2.3 Requirements & high-level architecture alignment

The final step of this methodology aims to align requirements models with HLA models in order to establish traceability between these two entities. It also intents to give a formal specification of these alignment links. We recall that in our methodology, for the requirements modeling we use the Openflexo FORMOD tool environment and for the HLA modeling we use Eclipse development environment IDE with Papyrus. Therefore, to establish alignment links we should bring together all models in the same environment or establish communication between them. To this end, we decided to include the FORMOD SysML/KAOS goals modeling mechanisms into Eclipse development IDE. Two ways to do this:

— The first solution: the FORMOD tool project is available as a MAVEN project in a git repository. This project could be uploaded in Eclipse and SysML/KAOS goal models could be imported in the Eclipse IDE.

Note: Maven is an open-source build tool developed by the Apache Foundation. It facilitates and automates certain tasks of managing a Java project.

— The second solution: We reproduce the SysML/KAOS goal model meta-model used in the FORMOD tool within Eclipse and import the designed goal models from the FORMOD tool into Eclipse IDE. This reproduction could be performed using The EMF models and the UML profiles mechanisms.

In our methodology, we have opted for the second solution while already our HLA modeling is based on SysML and its proposed extensions and also, SysML/KAOS is based on SysML extended with KAOS concepts. Adding to that, the second solution is more light-weight solution than the first one because there is no need to upload all the FORMOD project while we don't need all its proposed features. We are, only interested in SysML/KAOS goal model. Figure 3.17 shows an extract of the SysML/KAOS reproduction profile.

The graphical alignment mechanism presented in Section 4.2 of Chapter II supports three kinds of alignment *Satisfy*, *And_Satisfy* and *Milestone_Satisfy*. To specify these alignment links we have used Papyrus. The extensions concepts are defined as stereotypes applied on the meta-class *Dependency* which is customized to support the *Satisfy*, *And_Satisfy* and *Milestone_Satisfy* links. This profile is shown in Figure 3.18.

The Event-B formalization of these alignment links follows the translation rules pre-

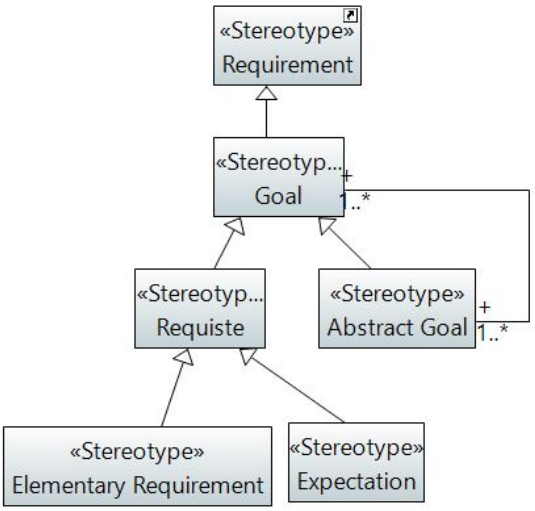

Figure 3.17 – SysML/KAOS reproduction profile

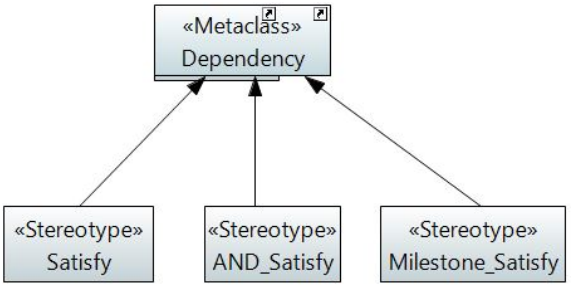

Figure 3.18 – Alignment profile

sented in Section 4.3 of Chapter II. This formalization is performed using the AtelierB tool as described is the usage scenario 2.1.1. It should be noted that, to conduct this translation Event-B specification issued from SysML/KAOS models and Event-B specification generated automatically from HLA models should be included in the same AtelierB project. This is to allow the alignment and communication between the two Event-B specification components.

The formalization rules presented in Section 4.3 which allows the automatic generation of the Event-B specification of the alignment links are performed as follows:

— Listing 38 shows the QVT alignment module header. Thus, this transformation takes as input meta-models the UML, the extended SysML and the Event-B. The Event-B meta-model is used as a source and a target meta-model. Lines 7 and 8 define the signature of the transformation which declares the transformation name and the source and target meta-models. in and inout keywords indicate source and target model variables. Lines 10 to 15 presents the Entry point while the execution of the transformation starts here by executing the instructions in the body of the main operation. In our case, the main operation is called Model2SatisfactionMachine(spec) and it takes the root of the Event-B model of type "EventB_Spec" as a parameter.

— Listing 39 presents an extract of the implementation of the mapping rule called Model2SatisfactionMachine (spec) responsible for the transformation of alignment links. First of all, this part of code selects all Abstraction links with an ap-

plied 'SysML_KAOS::Satisfy' stereotype and calls the AlignMachine(spec) mapping rule responsible of establishing alignment Event-B machine (Lines 4 to 16). If Abstraction link is stereotyped with 'SysM_KAOS::AND_Satisfy' or 'SysML_KAOS::Milestone_Satisfy', the operation AligAND_MLSMachine(spec) is called (Lines 17 to 36).

```
1  modeltype UML "strict" uses "http://www.eclipse.org/uml2/5.0.0/UML";
2  modeltype SYSML "strict" uses
3  BDD_MetaModel('http://www.example.org/BDD_MetaModel');
4  modeltype EventB "strict" uses
5  EventB_MetaModel('http://www.example.org/EventB_MetaModel');
6
7  transformation Alignement2EventB( in uml : UML,
8  in sysml : SYSML, inout eventb: EventB);
9
10 main()
11 {
12 var spec : EventB::EventB_Spec=
13 eventb.objectsOfType(EventB_Spec)->selectOne(a|a->notEmpty());
14 uml.rootObjects()[Model] -> map Model2SatisfactionMachine(spec);
15 }
```

Listing 38 – QVT Alignement module header

```
1  mapping UML::Model::Model2SatisfactionMachine(
2  inout spec : EventB::EventB_Spec) {
3
4  self.allSubobjectsOfType(Abstraction)->
5  select(s| s.getAppliedStereotype('SysML_KAOS::Satisfy')->notEmpty())
6  ->forEach(i)
7      {
8      if(i.supplier->selectOne(a|a->notEmpty() and spec.machines
9      ->select(m|m.name.equalsIgnoreCase
10     (a.name+"_Goal_Satisfaction_Interface"))->isEmpty())
11     ->notEmpty())then
12         {
13         spec.machines+= i->map AlignMachine(spec) ;
```

```
14          }
15      endif;
16      };
17      self.allSubobjectsOfType(Abstraction)
18  ->select(s|s.getAppliedStereotype('SysML_KAOS::AND_Satisfy')
19  ->notEmpty() or
20  s.getAppliedStereotype('SysML_KAOS::Milestone_Satisfy')->notEmpty())
21  ->forEach(i)
22      {
23      if(i.supplier->selectOne(a|a->notEmpty() and spec.machines
24  ->select(m|m.name.equalsIgnoreCase
25  (a.name+"_Goal_Satisfaction_Interface"))->isEmpty())
26  ->notEmpty())then
27          {
28          spec.machines+= i->map AligAND_MLSMachine(spec) ;
29          }
30      else
31          {
32          spec.machines+= i->map AligAND_MLSMachine2(spec) ;
33          }
34      endif;
35      };
36  }
```

Listing 39 – QVT Alignement module header

3 Conclusion

In this chapter we described the implementation concepts of the proposed methodology. they are considered as a framework for modeling requirements and HLA of complex systems and their formal verification. Adding to that, this framework gives a formally verifiable alignment between requirements and HLA models to guarantee the traceability between them.

This framework is implemented using a set of Plug-ins and tools that can be integrated in the Eclipse Platform. This choice allows to develop a complete environment to manage

the life cycle of complex systems from requirements to HLA. The integration with the Eclipse Platform allows indeed the exploitation of all features offered by the Eclipse Java IDE.

Chapter IV

CONCLUSION AND PERSPECTIVES

This final chapter concludes the thesis and exhibits the contributions presented in this work. We first provide a summary and draw some conclusions; we discuss some limitations of our work and propose some perspectives to overcome them and to go further.

1 Summary of contributions

The aim of the thesis is to propose a methodology for high-level architecture modeling aligned with system requirements.

In the first part, an overview of the existing approaches on RE, architecture modeling and Event-B formal specification is conducted. To show the usefulness of our proposed methodology, we presented a rich evaluation and discussion which emphasizes the limitations of existing research works. The primitive support for traceability between HLA and requirements has been recognized as one of the most significant limitations of current complex systems. similarly to the fact that the formal reasoning behind the safety-critical complex systems HLA and requirements is crucial.

The contributions presented in this thesis are manifold. The central contribution is to propose a holistic process to define system HLA aligned with system requirements. The approach places a great deal of emphasis on HLA complexity mastering, rigorous and formal reasoning, and traceability vis-a-vis stakeholders needs. To sum up, the main results and contributions of this thesis are:

- **High-level architecture modeling**

  We proposed first, a model based approach which enables HLA modeling using SysML diagrams. Four kinds of SysML diagrams have been used: package, block definition, state machine and sequence diagrams. These models allow to represent

HLA as a hierarchy of systems/sub-systems levels and answering a part of research question RQ1 dealing with the functional of the HLA. The second part of the research question RQ1 is about the non-functional part of HLA modeling which raises the first perspective of this thesis. SysML extensions were proposed to be aligned with Event-B refinement and decomposition mechanisms in order to automatically translate SysML models of HLA to Event-B specifications. These extensions are applied on two SysML parts: package diagrams are customized to represent the decomposition of system/sub-systems hierarchies and the refinement of a system by its sub-systems interplay; Sequence diagrams are extended with stereotypes applied on messages to refine a parent system task by the collaboration of its sub-systems processes. These extensions allow to answer research question RQ2.

- **High-level architecture formalization with** Event-B

  With the aim of ensuring the correctness of HLA models, a set of translation rules is defined to translate SysML models into Event-B specifications in order to formally verify them using AtelierB. This translation is conducted using QVT and Acceleo transformation language and comprises three sets of rules: translating package elements, translating SysML refinement extension to Event-B and translating SysML decomposition extension to Event-B. These translations take meta-models as input (extended SysML meta-models) and output (Event-B meta-model). This formalization allows to answer research question RQ3.

- **Requirements & high-level architecture alignment** In the context of traceability between HLA and system requirements, a model-based approach is defined to align complex systems HLA models with SysML/KAOS requirements models. This approach encompasses two steps. First, it allows graphical specification of the alignment of a leaf goal with HLA elements responsible for its satisfaction. For this purpose, three kinds of alignment links are defined and a new alignment meta-model is proposed. Second, it formalizes the alignment links in Event-B in order to be verified. To produce this Event-B specification, a set of transformation rules is proposed. The semantics of the alignments links are given by these rules and the definition of new proof obligations that can be discharged using AtelierB. This proposed graphical and formal alignment allows to satisfy research question RQ4.

- **Implementation of the methodology**

  We implemented the methodology with the help of the Eclipse IDE, which is an

Eclipse environment allowing designers to easily manage requirements modeling, HLA modeling and formal specification generation and mainly establishing alignment between these entities. More precisely, it enables to verify the consistency and correctness and gives a formal and rigorous reasoning of requirements and HLA using an automatic mappings of SysML/KAOS models and HLA models to Event-B formal specification. After that, mechanisms to establish graphically and formally alignment between HLA and requirements entities in order to prove the traceability between them is implemented. This implementation is presented as an Eclipse plugin which allows graphical modeling and the formalization of these models. However, this implementation lacks of formal basis about the used meta-models and applied transformations.

An illustration of this methodology was applied on the landing gear system case study enhanced with a second example taken from the train control system case study. Through this work, new problems have been emerged leaving wide horizon for other researches.

2 Limitations and perspectives

The experience that has been gained from the proposed methodology helps us to direct the further research in HLA specification, formal verification and traceability. Therefore, we present some directions for future work:

— This work only considers functional properties of requirements and HLA. In particular we do not consider non-functional properties of systems. For the sake of completeness, an extension of our methodology would be to investigate these non-functional properties integration. This integration process should follow the methodology steps which consists in graphically modeling these properties in the requirements and HLA models and formalizing them.

— The proposed mappings from HLA models to Event-B formal specifications are only in this direction. Consequently, this translation supports only HLA modeling updates to be propagated on the Event-B specification. However, when updates are produced directly on the Event-B specification they are not propagated to the HLA models. To deal with this issue, defining bidirectional translation would be beneficent to propagate updates in the two directions. This bidirectional process provides traceability between HLA models and Event-B specifications and allows

to identify errors on the original HLA models and vice versa.

— Our methodology proposes models translation steps that are described using QVT and Acceleo model transformation languages. However, these languages are not formal whereas a formal context is crucial in the scope of safety-critical system. Therefore, formally defining the rules in Event-B and discharging the associated proof obligations allows to prove their consistency, to animate them using ProB which shows the one to one mapping and to reveal several constraints that were missing when designing the rules informally or when specifying the meta-models. Another solution could be to use EB4EB, an Event-B based modelling framework allowing to manipulate Event-B features explicitly based on meta-modelling concepts. It preserves the core logical foundation, including semantics, of original Event-B models. [Riviere et al., 2022].

— HLA and requirements alignment is discussed and proposed in the methodology with three kinds of satisfaction links. This traceability allows only to link requirements to HLA elements responsible for their satisfaction, however, the impact of updates on requirements models and/or HLA models on other models and on established alignment links is not yet supported by our methodology.

— Our proposed methodology is limited to the requirements and HLA stages. However, supporting other software development life cycle stages has also been a prominent area of research for ensuring correctness of all systems artefacts. It is well known that not all desired requirements and properties of a system may be verified only at the requirements and HLA stages. We therefore propose, as future work, to support all system development life cycle stages such as other levels of the architecture, development, testing, deployment etc... and their formal specification.

— A digital twin [IBM, 2019] is a virtual representation of an object or system that covers its life cycle and allows to understand and predict its performance characteristics. It is updated from real-time data and uses simulation, machine learning and reasoning to aid decision-making and early errors prediction. Digital twins can show the impact of design changes, usage scenarios, environmental conditions, and countless other variables. This eliminates the need for physical prototypes, reducing development time and improving the quality of the final product or process. Digital twins could be a beneficial solution for safety-critical systems which require a high level of integrity and strong risks management guarantees while it allows to predict risks on

the virtual representation before its production on the physical system. Therefore, the use of digital twins is well suitable for safety-critical complex systems such as railway systems.

Chapter V

Résumé Étendu en Français

Contents

1 Introduction

Les systèmes complexes sont un ensemble de sous-systèmes reliés entre eux de manière significative pour représenter un tout intégré. La conception de l'architecture de haut niveau de tels systèmes devrait tenir compte des interrelations entre les sous-systèmes.

Les systèmes complexes sont considérés comme critiques pour la sécurité. Cependant, les outils traditionnels de conception graphique sont semi-formels, ce qui ne permet pas le raisonnement formel et rigoureux nécessaire aux systèmes critiques pour lesquels la sûreté et la sécurité sont des préoccupations majeures.

Le projet Train de Fret Autonome (TFA) du programme Train Autonome [1] est l'un des programmes de R&D et d'innovation de l'IRT Railenium [2], un centre d'essais et de recherche appliquée pour l'industrie ferroviaire en France, avec la coopération de plusieurs partenaires (SNCF, Alstom Transport, Hitachi Rail STS, Capgemini Engineering et Apsys). Ils visent l'amélioration des performances du système grâce à la mise en place de l'autonomie dans l'exploitation ferroviaire. Ce système est considéré comme un système complexe critique pour la sûreté, où il dépend de plus en plus de solutions efficaces qui peuvent répondre à l'hétérogénéité et à l'interaction des éléments physiques et logiciels. De plus, l'utilisation d'approches de vérification modernes peut être le facteur de différenciation afin de garantir la cohérence de ces systèmes. Il convient de noter que le TFA, dans le but d'éviter l'apparition de plusieurs problèmes tels que la perte de vies humaines, les blessures, les dommages environnementaux graves et les pertes économiques, par exemple, doit garantir la cohérence des fonctionnalités des systèmes.

De plus, la qualité d'un système dépend du degré de performance qu'il atteint dans la réponse à ses exigences. La traçabilité des exigences est largement reconnue comme un élément crucial de tout processus de développement de système rigoureux, en particulier pour la conception de systèmes complexes critiques.

Comme la conception graphique ne permet pas le raisonnement formel et rigoureux nécessaire aux systèmes critiques, l'utilisation de méthodes formelles est fortement recommandée pour la spécification de système. Cependant, l'un des principaux obstacles à leur adoption réside dans l'obtention de la spécification formelle du système.

1. Le programme Autonomous Train https://railenium.eu/train-autonome/
2. Railenium. http://railenium.eu/fr/

2 problème et motivation

La conception de systèmes complexes critiques pour la sureté, tels que le TFA, dépend de solutions qui permettent l'interaction entre leurs sous-systèmes. Ces sous-systèmes interagissent en échangeant des informations afin d'accomplir l'objectif principal du système global. Par conséquent, un modèle d'architecture de haut niveau HLA prenant en charge la hiérarchie en couches des composants est nécessaire. Dans le projet AFT, des représentations graphiques des composants du système sont recommandées pour spécifier, visualiser, comprendre et documenter le système de manière simple. De telles représentations permettent à toutes les parties prenantes de discuter et de s'accorder sur les principales caractéristiques du système à construire et permettent de vérifier si son HLA correspond à leurs besoins attendus. Alors que les normes ferroviaires d'ingénierie de la sécurité affirment que la modélisation des systèmes ferroviaires à l'aide de SysML permet de générer des composants corrects par construction, ce qui est un moyen de s'assurer que les besoins remplis par le modèle initial resteront respectés alors qu'il est censé être facilement compris par les experts ferroviaires. Dans cette perspective, nous formulons la question de recherche suivante:

QR1: Pouvons-nous fournir un langage de modélisation commun pour les architectures de haut niveau pouvant prendre en charge l'interaction des sous-systèmes?

La complexité de ces systèmes augmente en raison à la fois du grand nombre de composants de base intégrés à plusieurs niveaux et de l'importante hétérogénéité scientifique et technologique de ces systèmes. Cela soulève de multiples problèmes liés à l'exhaustivité, la cohérence, l'absence d'ambiguïté et l'exactitude de la conception du système. Cependant, lorsque les systèmes sont complexes, leur structure ne peut être décrite à un seul niveau ou avec une seule vue; des descriptions multi-niveau sont nécessaires pour les comprendre. Leur comportement émergent, dérivé des relations entre leurs éléments et avec l'environnement, via des boucles de rétroaction internes et externes, qui peuvent ne pas être compris ou prédits. Cela vient du fait qu'un système complexe est un ensemble de sous-systèmes présentés comme un tout intégré travaillant ensemble pour réaliser une mission principale et de plus, ces sous-systèmes ont leur propre vie et peuvent exister indépendamment de leur participation à la mission principale. Sur la base de ces considérations, il est possible de formuler une deuxième question de recherche:

QR2: Comment maîtriser la complexité de ces systèmes complexes?

Généralement, la conception graphique des systèmes complexes HLA est semi-formelle et sa sémantique est donnée en langage naturel, ce qui ne permet pas le raisonnement formel et rigoureux nécessaire aux systèmes critiques. D'ou, l'industrie a besoin d'une approche efficace pour la vérification des systèmes critiques afin de garantir leur consustance. Dans ce contexte, l'utilisation de méthodes formelles est fortement recommandée pour la spécification de la HLA des systèmes lors du développement de systèmes ferroviaires par exemple. La spécification formelle permet de vérifier formellement la preuve de la cohérence du système en modélisant sa structure et son comportement. Ainsi, il est possible de formuler la troisième question de recherche ainsi:

QR3: Comment fournir une spécification formelle des architectures de haut niveau pour vérifier leur cohérence?

Au-delà du fait que la HLA des systèmes complexes doivent être cohérents et vérifiés, il est nécessaire de vérifier que ses modèles sont alignés avec les exigences système qui doivent être aussi précises que possible. En effet, la conception de HLA est le plus souvent difficile car il peut être difficile de vérifier que les besoins des parties prenantes sont satisfaits. C'est pourquoi, si la spécification formelle permet de prouver la cohérence du système, il est recommandé de prouver également la cohérence de cet alignement. Ainsi, des activités de validation et de vérification doivent participer pour assurer l'exactitude de la conception par rapport aux besoins initialement spécifiés. Dans ce contexte, une dernière question de recherche peut être formulée comme suit:

QR4: Comment établir et vérifier les liens d'alignement entre les architectures de haut niveau et les exigences système?

3 Contexte de recherche

Pour repondre à ces questions de recherche, cette thèse propose une définition d'une méthodologie d'alignement entre les modèles d'exigences et les modèles d'architectures de haut niveau permettant de définir une traçabilité entre ces deux entités et de garantir ainsi que les modèles d'architectures de haut niveau répondent aux besoins des parties prenantes. C'est réalisé grâce à un couplage avec des spécifications formelles qui vérifient, d'une part, l'exactitude et la cohérence des modèles d'architectures de haut niveau requis pour les systèmes critiques et, d'autre part, la cohérence des liens d'alignement établis.

4 Une approche formelle pour la modélisation d'architectures de haut niveau de systèmes complexes alignées avec les modèles d'exigences

4.1 Aperçu de la méthodologie

La méthodologie proposée est basée sur les modèles. Elle est résumée dans la figure 5.1, composée de trois phases: (1) SysML/KAOS, (2) Modélisation de HLA en SysML et (3) Alignement.

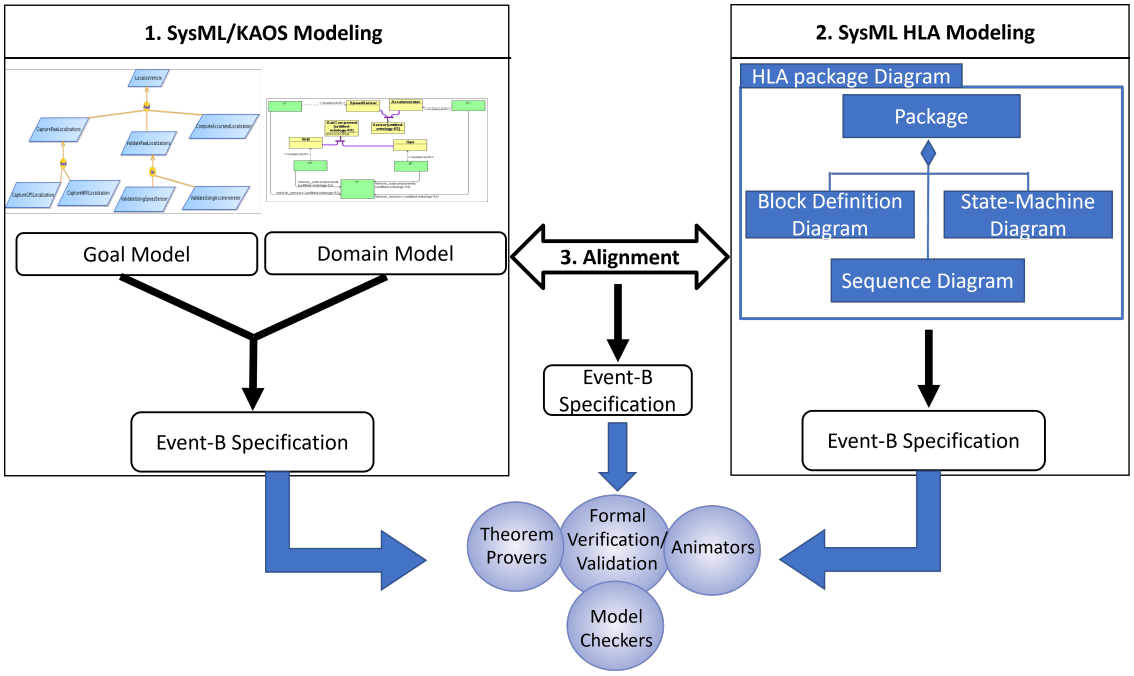

Figure 5.1 – Methodology Aperçu

Le but de cette méthodologie est de modéliser les exigences des parties prenantes, de représenter HLA dans une hiérarchie en couches de relations système/sous-systèmes et enfin d'aligner les éléments de HLA avec les exigences pour garantir la traçabilité entre eux.

La Modélisation en SysML/KAOS [Laleau et al., 2010]: Consiste en une phase de RE dans laquelle les exigences sont modélisées à l'aide de modèles de buts SysML/KAOS enrichis par des modèles de domaine. Ensuite, ces modèles d'exigences sont formalisés à l'aide de Event-B afin d'être formellement vérifiés. Nous avons choisi l'approche SysML/KAOS pour les raisons suivantes:

- ✓ Il fournit une forte expressivité sémantique.
- ✓ SysML est bien recommandé par les partenaires du projet TFA
- ✓ Event-B est spécialisé dans la modélisation de systèmes et a été utilisé dans des applications de systèmes critiques pour la sureté telles que les lignes du métro parisien [Behm et al., 2003, Abrial, 2006]
- ✓ Les mécanismes de raffinement et de décomposition de SysML/KAOS sont bien adaptés à Event-B.

Cette phase est composée de deux étapes :

- Tout d'abord, une modélisation graphique des exigences du système sous la forme d'une hiérarchie de buts est réalisée en utilisant la modélisation SysML/KAOS et des modèles de raffinement. Ensuite, ces modèles de buts sont enrichis par des modèles de domaine qui définissent la structure du système et les concepts utilisés pour définir les buts.
- Deuxièmement, une formalisation en Event-B de ces modèles de but et de domaine est effectuée.

Une étape de vérification formelle utilisant des démonstrateurs Event-B, des vérificateurs de modèles et des animateurs est appliquée sur la spécification formalisée pour prouver l'exactitude et la cohérence des exigences du système.

La Modélisation de HLA en SysML: Correspond à la définition du HLA des systèmes complexes critiques qui doit être validée par des experts du domaine et sa traduction en Event-B spécification formelle pour prouver sa correction et cohérence.

Pour mener à bien cette phase, un processus en deux étapes est effectué:

- Tout d'abord, une modélisation graphique de HLA dans une hiérarchie système/sous-système en couches est présentée en utilisant les mécanismes d'extension de SysML que nous avons définis.
- Deuxièmement, une traduction automatique des modèles HLA en modèles Event-B est établie.

Les résultats présentés dans cette phase ont été publiés dans [Bougacha et al., 2022b, Bougacha et al., 2022a].

Alignment: Vise à définir des liens d'alignement entre les éléments de SysML/KAOS et les éléments de HLA. Ces liens d'alignement permettent de garantir des relations de

traçabilité entre les entités participantes pour garantir que les éléments de HLA satisfont les objectifs du système et donc satisfont les besoins des parties prenantes.

Le processus d'alignement comprend deux étapes:

- Tout d'abord, les liens d'alignement sont spécifiés graphiquement afin d'être bien compris, documenté et validé par toutes les parties prenantes. Pour implémenter ces graphes, trois types de relations de satisfaction ont été proposées dans le méta-modèle d'alignement que nous avons défini.
- Dans un second temps, une formalisation de ces liens d'alignement est réalisée pour prouver leurs cohérences. Cette formalisation permet de générer une spécification en Event-B adaptée à l'alignement proposé à l'aide d'un ensemble de règles de traduction. De nouvelles obligations de preuve sont générées en plus des obligations de preuve existantes de type préservation d'invariant, de faisabilité d'actions non déterministes etc.

Une étape de vérification formelle est effectuée pour vérifier la spécification en Event-B résultante. Le déchargement de toutes les obligations de preuve (obligations de preuve existantes et d'alignement) permet de prouver que les exigences du système sont formellement alignées avec les éléments de HLA.

4.2 Modélisation d'architecture de haut niveau et vérification formelle

Les systèmes complexes sont des systèmes composés de nombreux composants qui peuvent interagir entre eux, tels que le système de gestion du trafic aérien, les systèmes ferroviaires, le réseau intelligent, les systèmes automobiles autonomes, la surveillance médicale, les systèmes de contrôle industriel, les systèmes robotiques, etc. Leur comportement est difficile à modéliser due aux dépendances, compétitions, relations ou autres types d'interactions entre leurs parties ou entre un système donné et son environnement. Dans de nombreux cas, il est utile de représenter un tel système comme une hiérarchie où les nœuds représentent les composants relié par leurs interactions. Par conséquent, la conception de HLA de ces systèmes dépend de solutions qui peuvent traiter l'interaction entre leurs sous-systèmes. Ce HLA doit être représenté comme une hiérarchie en couches de sous-systèmes. Il doit permettre de spécifier les principaux éléments fonctionnels du système, ainsi que ses interfaces et interactions.

Pour cela, nous proposons de combiner SysML et la méthode formelle Event-B. Le choix se porte sur SysML plutôt que sur UML alors qu'il propose un ensemble de con-

cepts plus pertinents pour modéliser les systèmes et comme présenté précédemment, est recommandé par les partenaires du projet TFA. En effet, le projet TFA réutilise le RailTopoModel [3] qui contient une ontologie fonctionnelle basée sur SysML d'une infrastructure ferroviaire. De plus, l'initiative européenne EULYNX [4] a défini un modèle standard basé sur SysML des composants du système de signalisation ferroviaire. Event-B permet de spécifier des systèmes plutôt que simplement des logiciels et il est déjà utilisé dans de nombreux systèmes critiques pour la sécurité [Lecomte et al., 2017]. Son utilisation est également préconisée dans l'étude de ASTRAIL [ASTRAIL, 2017]

La figure 5.2 présente un aperçu du processus de modélisation de HLA et de sa formalisation en modèles Event-B.

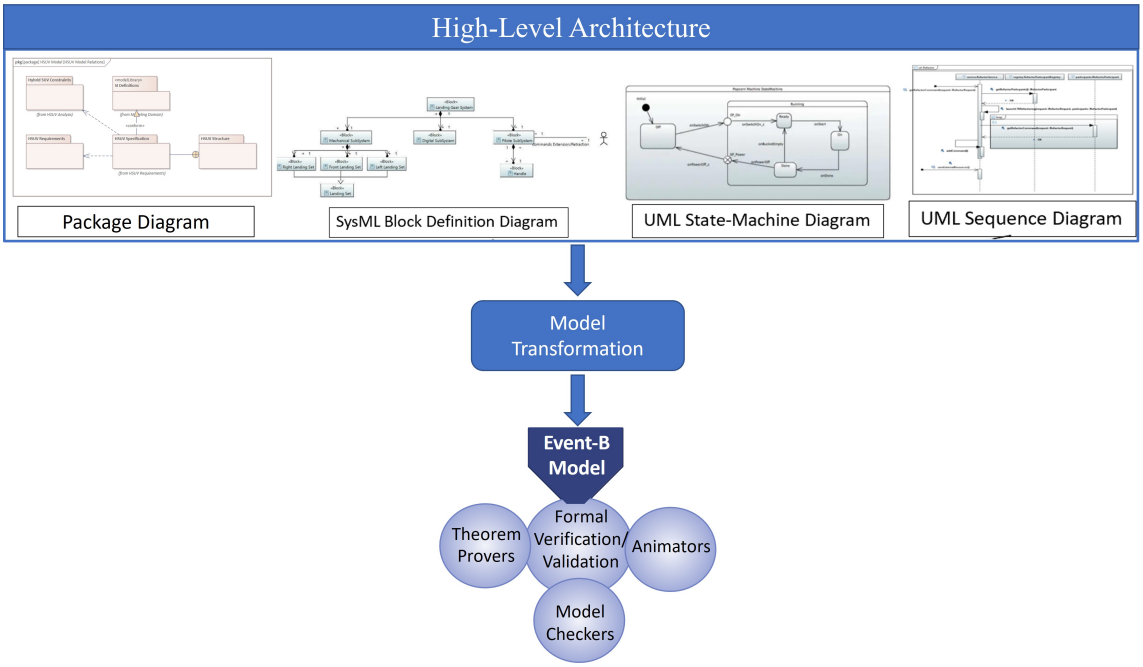

Figure 5.2 – HLA modeling and formalization approach

Ce processus est composé de deux étapes

— La première étape consiste à modéliser des architectures de haut niveau à l'aide de diagrammes SysML. Quatre types de diagrammes SysML ont été sélectionnés: package, définition de bloc, état-transition et diagrammes de séquence.

Les mécanismes de raffinement et de décomposition sont des caractéristiques intéressantes de Event-B qui facilitent une conception pas à pas et font des preuves plus

3. http://www.railtopomodel.org/en/. Il s'agit d'un standard pour la représentation des données liées à l'infrastructure ferroviaire
4. https://www.eulynx.eu/

facile à décharger. Cela permet de maîtriser la complexité des systèmes complexes. Par conséquent, nous proposons d'étendre SysML avec ces mécanismes pertinents pour permettre une traduction automatique. Ces extensions sont appliquées sur deux diagrammes SysML, les diagrammes de package et de séquence.

— La deuxième étape consiste à traduire les diagrammes SysML en modèles Event-B. Cette traduction de modèle est mise en œuvre à l'aide de trois ensembles de règles: un ensemble pour les éléments liés à un package; un ensemble pour les extensions de raffinement SysML; un ensemble pour les extensions de décomposition SysML.

Cette étape se déroule en deux phases.

— Une transformation de modèle à modèle pour implémenter les règles ci-dessus. Il prend en entrée le méta-modèle étendu de SysML et produit en sortie un modèle Event-B conforme au méta-modèle de Event-B.

— Une transformation modèle-texte pour générer Event-B des spécifications formelles textuelles. Cette spécification textuelle peut être introduite dans des prouveurs tels que AtelierB [AtelierB, 1990], des model-checkers et des animateurs tels que ProB [ProB, 2003] pour vérifier la cohérence des modèles de HLA du système et être validés par des experts du domaine à travers des animations de modèles.

4.3 Alignement entre exigences & architecture de haut niveau

La qualité d'un système est la principale mesure de son succès, qui dépend de la mesure dans laquelle il répond à ses exigences. D'où, des liens d'alignement entre les modèles d'exigences et les modèles de HLA doivent être établis. Ces liens sémantiques peuvent être le support pour prouver la conformité de la spécification de HLA avec l'expression des exigences système. La troisième partie de notre travail vise à proposer une approche basée sur les modèles pour établir graphiquement des liens d'alignement entre les modèles SysML/KAOS et les modèles de HLA puis la traduction de ces liens en spécification formelle en Event-B pour les vérifier formellement (voir Figure 5.3). SysML/KAOS [Laleau et al., 2010, Gnaho et al., 2013b] est une méthode de l'ingénierie des exigences RE qui permet la modélisation des exigences fonctionnelles et non fonctionnelles et le modele de domaine domaine d'un système. Cette approche vise à dériver une spécification Event-B directement des modèles SysML/KAOS.

Le processus est composé de deux étapes :

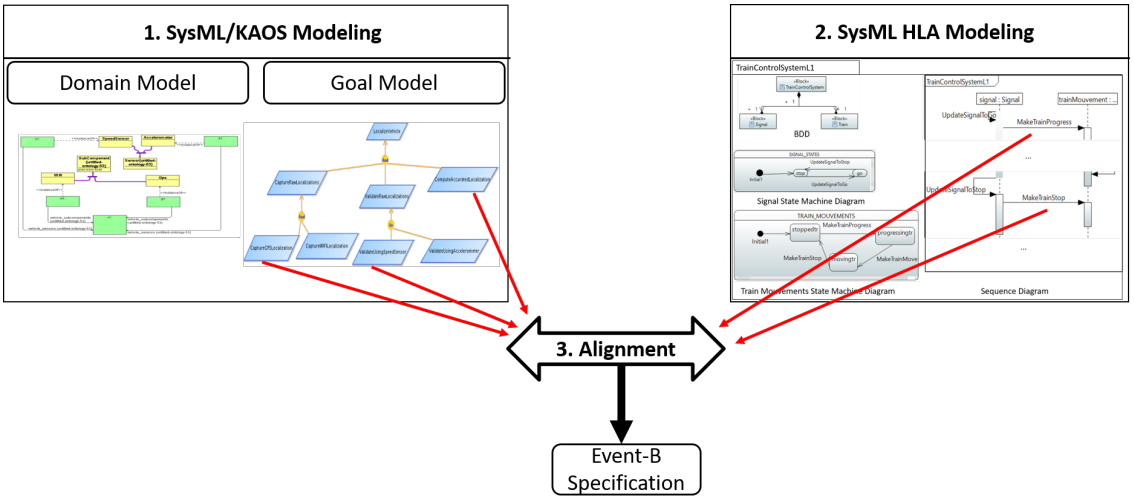

Figure 5.3 – Approche d'alignement entre les modèles SysML/KAOS et les modèles SysML HLA

— **Alignement graphique:** La première étape consiste à modéliser graphiquement les liens d'alignement entre les buts feuilles et les messages du diagramme de séquence. Cette étape de modélisation graphique est une manière simple d'établir des liens d'alignement et de les valider par les acteurs du système. Pour élaborer cette étape, la partie des exigences est représenté par un modèle de but SysML/KAOS enrichi par un modèle de domaine et la partie de l'architecture est représenté par un modèle de HLA en SysML étendu.

Pour spécifier les liens d'alignement entre ces deux parties, le choix se porte sur les buts feuilles car ils sont les objectifs les plus concrets d'un modèle de buts. Un but feuille est affecté à un agent (environnement, logiciel ou sous-système agent) responsable de la satisfaction des but. les messages d'un diagramme de séquence représentent les interactions entre les composants. Chaque message correspond à une transition dans le digramme d'état-transition du bloc associé au composant cible. Un lien d'alignement est défini pour chaque but feuille attribué à un sous-système. Trois types de liens ont été proposés :

- Le premier type est appelé *Satisfy*. Il est défini pour représenter un lien d'alignement lorsqu'un message peut satisfaire un but.
- Le deuxième type est appelé *And_Satisfy*. Il représente une lien d'alignement lorsqu'un but est satisfait par un ensemble de messages, c'est-à-dire l'exécution de tous les messages, dans n'importe quel ordre, est nécessaire pour la satisfaction du but.

- Le dernier type est appelé *Milestone_Satisfy*. Ce lien représente le fait qu'une exécution séquentielle d'un ensemble de messages dans un ordre spécifique est nécessaire pour satisfaire un but.

— **Formalisation des liens d'alignement graphique:** La deuxième étape consiste à formaliser ces liens d'alignement en une spécification Event-B afin d'être intégrés dans AtelierB pour étre vérifier formellement. Pour effectuer cette formalisation, un ensemble de règles de traduction est défini. La formalisation des modèles SysML/KAOS en Event-B est décrite dans [Matoussi et al., 2011a, Tueno Fotso et al., 2018] et la formalisation des modèles HLA en SysML étendu est présentée dans [Bougacha et al., 2022b]. Chaque but, y compris les buts feuilles, est transformé en événement dans la spécification Event-B des modèles d'exigences et chaque message du diagramme de séquence est également transformé en événement dans la spécification Event-B de modèles HLA.

L'idée principale est de définir un lien d'alignement entre un ensemble de messages et un but feuille comme un lien de raffinement entre les événements Event-B correspondants. Il existe de nombreuses raisons pour lesquelles il n'est pas possible d'utiliser le raffinement Event-B standard.

- Premièrement, la sémantique du raffinement de Event-B ne correspond pas à notre sémantique d'alignement. En effet, le processus de raffinement de Event-B permet d'enrichir progressivement les différentes parties qui constituent le système avec plus de détails fonctionnels, de sécurité, etc., en partant d'un modèle abstrait vers un modèle plus concret.
- Deuxièmement, les messages peuvent appartenir à des machines Event-B distinctes alors que dans Event-B il n'est pas possible de spécifier qu'un ensemble de machines raffine une machine.
- De plus, comme la formalisation des modèles SysML/KAOS et des modèles de HLA en SysML étendu est effectuée dans Event-B, nous pensons qu'il serait plus approprié de formaliser l'alignement avec Event-B.

Notre solution est donc de construire une nouvelle machine Event-B pour chaque lien d'alignement. De nouveaux ensembles d'obligations de preuve de raffinement sont ensuite spécifiés, un pour chaque type d'alignement. Le déchargement de ces obligations de preuve permet de vérifier formellement la satisfaction d'un but feuille par un ensemble de messages de HLA.

5 Conclusion

L'objectif de la thèse est de proposer une méthodologie de modélisation d'architecture de haut niveau alignée avec les exigences système.

Dans la première partie, un aperçu des approches existantes sur les RE, la modélisation d'architecture et la spécification formelle en Event-B est réalisé. Pour montrer l'utilité de notre méthodologie proposée, nous avons présenté une évaluation et une discussion riches qui soulignent les limites des travaux de recherche existants. La prise en charge primitive de la traçabilité entre HLA et les exigences a été reconnue comme l'une des limitations les plus importantes des systèmes complexes actuels. Aussi, le raisonnement formel derrière les modèles de HLA et des exigences des systèmes complexes critiques est crucial.

Les contributions présentées dans cette thèse sont multiples. La contribution centrale est de proposer un processus holistique pour définir le HLA aligné avec les exigences du système. L'approche accorde une grande importance à la maîtrise de la complexité HLA, au raisonnement rigoureux et formel, et à la traçabilité vis-à-vis des besoins des parties prenantes. En résumé, les principaux résultats et contributions de cette thèse sont:

— La modélisation de l'architecture de haut niveau. (répondre à une partie de la question de recherche QR1 traitant la partie fonctionnel de la modélisation de HLA).

— La proposition de mécanismes de raffinement et de décomposition pour maîtriser la complexité des modèles de HLA (répondre à la question de recherche QR2).

— Formalisation de l'architecture de haut niveau avec Event-B (répondre à la question de recherche QR3).

— Alignement de l'architecture de haut niveau avec les exigences systèmes (répondre à la question de recherche QR4).

— Implémentions de la méthodologie.

Une illustration de cette méthodologie a été appliquée sur l'étude de cas du système de train d'atterrissage enrichi d'un deuxième exemple de l'étude de cas du système ferroviaire. À travers ce travail, de nouveaux problèmes sont apparus laissant un large horizon pour d'autres recherches.

Dans ce contexte, notre travail ne considère que les propriétés fonctionnelles des exigences et de HLA. En particulier, une extension de notre méthodologie serait d'étudier ces non-fonctionnels intégration des propriétés. Aussi, Les transformations proposés des modèles de HLA aux spécifications formelles en Event-B sont uniquement dans ce sens.

Par conséquent, cette traduction ne prend en charge que les mises à jour de la modélisation de HLA à propager sur la spécification EVENT-B. Cependant, lorsque les mises à jour sont produits directement sur la spécification EVENT-B ils ne sont pas propagés aux modèles de HLA. Pour résoudre ce problème, une autre perspective est envisagé. Il consiste à définir une traduction bidirectionnelle qui serait bénéfique pour propager les mises à jour dans les deux sens ce qui permet d'assure la traçabilité entre les modèles de HLA et les spécifications EVENT-B et d'identifier les erreurs sur les modèles de HLA originaux et vice versa.

Une autre perspective est de formaliser les règles de transformation en EVENT-B tandis que notre méthodologie propose des étapes de traduction de modèles décrites à l'aide des langages de transformation de modèles non formels comme QVT et Acceleo alors qu'un contexte formel est crucial dans le cadre d'un système critique. Par conséquent, définir formellement les règles en EVENT-B et de décharger les obligations de preuve associées permet de prouver leur cohérence, de les animer à l'aide de ProB qui révéle plusieurs contraintes qui manquaient lors de la conception informelle des règles ou lors de la spécification des méta-modèles.

L'alignement entre HLA et les exigences est discuté et proposé dans la méthodologie avec trois types de liens. Cette traçabilité permet uniquement de lier les exigences aux éléments de HLA responsables de leur satisfaction, cependant, l'impact des mises à jour des modèles d'exigences et/ou des modèles de HLA sur les autres modèles et sur les liens d'alignement établis n'est pas encore supporté par notre méthodologie. D'où, une nouvelle perspective est envisagé.

Une dernière perspective intéressante pour notre travail est la prise en charge d'autres étapes du cycle de vie du développement logiciel. Ce cycle de vie est également un domaine de recherche important pour garantir l'exactitude de tous les artefacts du système tandis que notre méthodologie proposée se limite aux étapes d'exigences et de HLA.

Bibliography

[Abrial et al., 2010] Abrial, J., Butler, M. J., Hallerstede, S., Hoang, T. S., Mehta, F., and Voisin, L. (2010). Rodin: an open toolset for modelling and reasoning in Event-B. *Int. J. Softw. Tools Technol. Transf.*, 12(6):447–466. (Cited in pages 38 and 40.)

[Abrial, 2006] Abrial, J.-R. (2006). Formal methods in industry: achievements, problems, future. In *Proceedings of the 28th international conference on Software engineering*, pages 761–768. (Cited in pages 63 and 194.)

[Abrial, 2009] Abrial, J.-R. (2009). Event model decomposition. *Technical report/[ETH, Department of Computer Science*, 626. (Cited in pages 37, 45, 56 and 57.)

[Abrial, 2010] Abrial, J.-R. (2010). *Modeling in Event-B: system and software engineering*. Cambridge University Press. (Cited in pages 24, 34, 38 and 79.)

[Abrial and Hallerstede, 2007] Abrial, J.-R. and Hallerstede, S. (2007). Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, 77(1-2):1–28. (Cited in pages 25 and 45.)

[Abrial and Hoare, 1996] Abrial, J.-R. and Hoare, A. (1996). *The B-book: assigning programs to meanings*, volume 1. Cambridge university press Cambridge. (Cited in pages 34 and 39.)

[Albinet et al., 2008] Albinet, A., Begoc, S., Boulanger, J.-L., Casse, O., Dal, I., Dubois, H., Lakhal, F., Louar, D., Peraldi-Frati, M.-A., Sorel, Y., et al. (2008). The MeMVaTEx methodology: from requirements to models in automotive application design. In *Embedded Real Time Software and Systems (ERTS2008)*. (Cited in pages 49 and 54.)

[ANR, 2014] ANR (2014). Formal Verification of Properties. `https://formose.lacl.fr/D3.2.b_c.pdf`. (Cited in page 24.)

[ANR-14-CE28-0009, 2014] ANR-14-CE28-0009 (2014). Formose ANR project (2017). `http://formose.lacl.fr/`. (Cited in page 24.)

[ASTRAIL, 2017] ASTRAIL (2017). ASTRAIL European project D4.1 - Report on Analysis and on Ranking of Formal Methods. http://www.astrail.eu/download.aspx?id=bb46b81b-a5bf-4036-9018-cc6e7d91e2c2. (Cited in pages 4, 39, 66 and 196.)

[AtelierB, 1990] AtelierB (1990). Atelier B tool. https://www.atelierb.eu/en/atelier-b-tools/. (Cited in pages 24, 38, 67, 86, 124, 157 and 197.)

[Barendrecht, 2010] Barendrecht, P. J. (2010). Modeling transformations using QVT operational mappings. *Research project report. Eindhoven: Eindhoven University of Technology Department of Mechanical Engineering Systems Engineering Group.* (Cited in page 155.)

[Bārzdiņš et al., 2010] Bārzdiņš, J., Bārzdiņš, G., Čerāns, K., Liepiņš, R., and Sproģis, A. (2010). UML style graphical notation and editor for OWL 2. In *Perspectives in Business Informatics Research: 9th International Conference, BIR 2010, Rostock Germany, September 29–October 1, 2010. Proceedings 9*, pages 102–114. Springer. (Cited in page 156.)

[Behjati et al., 2011] Behjati, R., Yue, T., Nejati, S., Briand, L., and Selic, B. (2011). An AADL-based SysML profile for architecture level systems engineering: approach, metamodels, and experiments. *ModelME! Rep.*, pages 2001–03. (Cited in pages XIII, 31 and 32.)

[Behm et al., 1999] Behm, P., Benoit, P., Faivre, A., and Meynadier, J.-M. (1999). Meteor: A successful application of b in a large project. In *World Congress on Formal Methods*, volume 1708, pages 369–387. (Cited in page 5.)

[Behm et al., 2003] Behm, P., Benoit, P., Faivre, A., and Meynadier, J.-M. (2003). A successful application of b in a large project. In *FM'99-Formal Methods: World Congress on Formal Methods in the Developement of Computing Systems, Toulouse, France, September 20-24, 1999, Proceedings, Volume I*, page 369. Springer. (Cited in pages 63 and 194.)

[Berglehner et al., 2019] Berglehner, R., Rasheeq, A., and Cherif, I. (2019). An approach to improve SysML railway specification using UML-B and EVENT-B. *Poster presented at RSSRail.* (Cited in page 42.)

[Bézivin et al., 2003] Bézivin, J., Dupé, G., Jouault, F., Pitette, G., and Rougui, J. E. (2003). First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, volume 37, page 46. (Cited in page 155.)

[Bézivin et al., 2005] Bézivin, J., Jouault, F., Rosenthal, P., and Valduriez, P. (2005). Modeling in the large and modeling in the small. In *Model Driven Architecture: European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, June 26-27, 2003 and Linköping, Sweden, June 10-11, 2004. Revised Selected Papers*, pages 33–46. Springer. (Cited in page 155.)

[Bon et al., 2022] Bon, P., Dutilleul, S. C., and Bougacha, R. (2022). ATO over ETCS: A system analysis. In *18th International Conference on Railway Engineering Design and Operation (COMPRAIL 2022)*. (Cited in pages 65 and 176.)

[Bon et al., 2023] Bon, P., Dutilleul, S. C., Bougacha, R., and Laleau, R. (2023). MODEL ENGINEERING FOR CRITICAL SYSTEMS: THE ATO OVER ETCS FOR FREIGHT TRAINS USE CASE. *International Journal of Transport Development and Integration*. (Cited in pages 61 and 176.)

[Boniol and Wiels, 2014] Boniol, F. and Wiels, V. (2014). The landing gear system case study. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 1–18. Springer. (Cited in pages 9, 10, 107 and 123.)

[Bonvoisin, 2016] Bonvoisin, D. (2016). 25 years of formal methods at RATP. In *International Railway Safety Council (IRSC2016)*. (Cited in pages 39 and 62.)

[Bougacha, 2020] Bougacha, R. (2020). A formal approach for the modeling of high-level architectures aligned with system requirements. In *Rigorous State-Based Methods: 7th International Conference, ABZ 2020, Ulm, Germany, May 27–29, 2020, Proceedings 7*, pages 409–413. Springer. (Cited in page 61.)

[Bougacha, 2022a] Bougacha, R. (2022a). A holistic approach for modeling railway systems on the ATO over ERTMS case study: high-level architecture models. `https://github.com/RacemBougacha/Railway-System.git`. (Cited in page 176.)

[Bougacha, 2022b] Bougacha, R. (2022b). The Landing Gear System case study. https://github.com/RacemBougacha/Landing-Gear-System.git. (Cited in pages 124, 129, 138 and 176.)

[Bougacha, 2023] Bougacha, R. (2023). ATO over ETCS case study. https://github.com/RacemBougacha/ATO-over-ETCS.git. (Cited in page 176.)

[Bougacha et al., 2022a] Bougacha, R., Laleau, R., Bon, P., and Collart-Dutilleul, S. (2022a). Modeling train systems: from high-level architecture graphical models to formal specifications. In *Risks and Security of Internet and Systems - 17th International Conference*. Springer, To be published. (Cited in pages 61, 64, 75, 86, 97, 176 and 194.)

[Bougacha et al., 2023] Bougacha, R., Laleau, R., and Collart-Dutilleul, S. (2023). Formal alignment of requirements models with high-level architecture models. In *27th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2023)*. IEEE. (Cited in pages 61 and 107.)

[Bougacha et al., 2022b] Bougacha, R., Laleau, R., Collart-Dutilleul, S., and Ayed, R. B. (2022b). Extending SysML with Refinement and Decomposition Mechanisms to Generate Event-B Specifications. In Aït-Ameur, Y. and Crăciun, F., editors, *Theoretical Aspects of Software Engineering*, pages 256–273, Cham. Springer International Publishing. (Cited in pages 61, 64, 91, 176, 194 and 199.)

[Budinsky et al., 2004] Budinsky, F., Ellersick, R., Steinberg, D., Grose, T. J., and Merks, E. (2004). *Eclipse modeling framework: a developer's guide*. Addison-Wesley Professional. (Cited in page 28.)

[Butler, 2009a] Butler, M. (2009a). Decomposition structures for Event-B. In *International Conference on Integrated Formal Methods*, pages 20–38. Springer. (Cited in pages 45 and 56.)

[Butler, 2009b] Butler, M. J. (2009b). Decomposition Structures for Event-B. In Leuschel, M. and Wehrheim, H., editors, *Integrated Formal Methods, 7th International Conference, IFM 2009, Düsseldorf, Germany, February 16-19, 2009. Proceedings*, volume 5423 of

Lecture Notes in Computer Science, pages 20–38. Springer. (Cited in pages 37, 57 and 85.)

[CATIA, 2011] CATIA (2011). MAGICDRAW. https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/. (Cited in page 18.)

[Chung et al., 2000] Chung, L., Nixon, B., Yu, E., and Mylopoulos, J. (2000). Non-functional requirements. *Software Engineering*. (Cited in page 19.)

[Cicchetti et al., 2012] Cicchetti, A., Ciccozzi, F., Mazzini, S., Puri, S., Panunzio, M., Zovi, A., and Vardanega, T. (2012). CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems. In Goedicke, M., Menzies, T., and Saeki, M., editors, *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, pages 362–365. ACM. (Cited in page 41.)

[Clarke et al., 1993] Clarke, D., Lee, I., and Xie, H.-L. (1993). VERSA: A tool for the specification and analysis of resource-bound real-time systems. (Cited in page 28.)

[Clavel et al., 2003] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. (2003). The maude 2.0 system. In *International Conference on Rewriting Techniques and Applications*, pages 76–87. Springer. (Cited in page 52.)

[Clements, 1996] Clements, P. C. (1996). A survey of architecture description languages. In *Proceedings of the 8th international workshop on software specification and design*, pages 16–25. IEEE. (Cited in page 26.)

[Dahmann et al., 1997] Dahmann, J. S., Fujimoto, R. M., and Weatherly, R. M. (1997). The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*, pages 142–149. (Cited in page 26.)

[Dahmann et al., 1998] Dahmann, J. S., Kuhl, F. S., and Weatherly, R. M. (1998). Standards for simulation: As simple as possible but not simpler the high level architecture for simulation. *Simul.*, 71(6):378–387. (Cited in page 4.)

[Dardenne et al., 1993] Dardenne, A., Van Lamsweerde, A., and Fickas, S. (1993). Goal-directed requirements acquisition. *Science of computer programming*, 20(1-2):3–50. (Cited in page 19.)

[Eclipse, 2009] Eclipse, I. (2009). Eclipse ide. *Website www. eclipse. org Last visited: July*, pages 1–20. (Cited in page 159.)

[Espinoza et al., 2009] Espinoza, H., Cancila, D., Selic, B., and Gérard, S. (2009). Challenges in combining SysML and MARTE for model-based design of embedded systems. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 98–113. Springer. (Cited in page 33.)

[Faugere et al., 2007] Faugere, M., Bourbeau, T., De Simone, R., and Gerard, S. (2007). Marte: Also an uml profile for modeling aadl applications. In *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*, pages 359–364. IEEE. (Cited in page 31.)

[Fayolle, 2017] Fayolle, T. (2017). *Combinaison de méthodes formelles pour la spécification de systèmes industriels*. PhD thesis, Paris Est. (Cited in pages 41, 43 and 44.)

[Feiler and Greenhouse, 2005] Feiler, P. and Greenhouse, A. (2005). Plug-in Development for the Open Source AADL Tool Environment. (Cited in page 28.)

[Feiler et al., 2006a] Feiler, P. H., Gluch, D. P., and Hudak, J. J. (2006a). The architecture analysis & design language (AADL): An introduction. Technical report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst. (Cited in page 26.)

[Feiler et al., 2006b] Feiler, P. H., Lewis, B. A., and Vestal, S. (2006b). The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems. In *2006 ieee conference on computer aided control system design, 2006 ieee international conference on control applications, 2006 ieee international symposium on intelligent control*, pages 1206–1211. IEEE. (Cited in page 26.)

[Fotso et al., 2018a] Fotso, S. J. T., Frappier, M., Laleau, R., and Mammar, A. (2018a). Back Propagating B System Updates on SysML/KAOS Domain Models. In *2018 23rd In-*

ternational Conference on Engineering of Complex Computer Systems (ICECCS), pages 160–169. IEEE. (Cited in pages 25 and 92.)

[Fotso et al., 2018b] Fotso, S. J. T., Frappier, M., Laleau, R., Mammar, A., and Barradas, H. R. (2018b). The Generic SysML/KAOS Domain Metamodel. *arXiv preprint arXiv:1811.04732*. (Cited in page 92.)

[Fotso et al., 2018c] Fotso, S. J. T., Frappier, M., Laleau, R., Mammar, A., and Leuschel, M. (2018c). Formalisation of SysML/KAOS goal assignments with B system component decompositions. In *International Conference on Integrated Formal Methods*, pages 377–397. Springer. (Cited in pages XIII, 25, 46, 47 and 48.)

[Fotso et al., 2018d] Fotso, S. J. T., Mammar, A., Laleau, R., and Frappier, M. (2018d). Event-B expression and verification of translation rules between SysML/KAOS domain models and B system specifications. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 55–70. Springer. (Cited in page 24.)

[Frappier et al., 2014] Frappier, M., Gervais, F., Laleau, R., and Milhau, J. (2014). Refinement patterns for ASTDs. *Formal Aspects of Computing*, 26(5):919–941. (Cited in page 44.)

[Fuentes-Fernández and Vallecillo-Moreno, 2004] Fuentes-Fernández, L. and Vallecillo-Moreno, A. (2004). An introduction to UML profiles. *UML and Model Engineering*, 2(6-13):72. (Cited in page 161.)

[Gaufillet, 2005] Gaufillet, P. (2005). TOPCASED-Toolkit In Open source for Critical Applications & systems Development. In *AADL Workshop*. (Cited in page 28.)

[Gnaho and Semmak, 2010] Gnaho, C. and Semmak, F. (2010). Une extension SysML pour l'ingénierie des exigences dirigée par les buts. In *INFORSID*, pages 277–292. (Cited in pages 22 and 24.)

[Gnaho et al., 2013a] Gnaho, C., Semmak, F., and Laleau, R. (2013a). An overview of a SysML extension for goal-oriented NFR modelling: Poster paper. In *IEEE 7th Inter-*

national Conference on Research Challenges in Information Science (RCIS), pages 1–2. IEEE. (Cited in page 22.)

[Gnaho et al., 2013b] Gnaho, C., Semmak, F., and Laleau, R. (2013b). Modeling the impact of non-functional requirements on functional requirements. In *International Conference on Conceptual Modeling*, pages 59–67. Springer. (Cited in pages 20, 92 and 197.)

[Goknil et al., 2014] Goknil, A., Kurtev, I., and Van Den Berg, K. (2014). Generation and validation of traces between requirements and architecture based on formal trace semantics. *Journal of Systems and Software*, 88:112–137. (Cited in pages 52, 54, 56 and 57.)

[Gotel and Finkelstein, 1994] Gotel, O. C. and Finkelstein, C. (1994). An analysis of the requirements traceability problem. In *Proceedings of ieee international conference on requirements engineering*, pages 94–101. IEEE. (Cited in page 5.)

[Hammoudi et al., 2008] Hammoudi, S., Alouini, W., and Lopes, D. (2008). Towards a Semi-Automatic Transformation Process in MDA - Architecture and Methodology. In *International Conference on Enterprise Information Systems*. (Cited in pages XIV and 154.)

[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 45 and 46.)

[Holt and Perry, 2008] Holt, J. and Perry, S. (2008). *SysML for systems engineering*, volume 7. IET. (Cited in pages 16 and 67.)

[Horkoff et al., 2016] Horkoff, J., Aydemir, F. B., Cardoso, E., Li, T., Maté, A., Paja, E., Salnitri, M., Mylopoulos, J., and Giorgini, P. (2016). Goal-oriented requirements engineering: a systematic literature map. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 106–115. IEEE. (Cited in pages XIII, 19 and 20.)

[IBM, 2013] IBM (2013). IBM Engineering Systems Design Rhapsody - Developer. https://www.ibm.com/products/systems-design-rhapsody. (Cited in page 18.)

[IBM, 2019] IBM (2019). IBM Digital Twin. (Cited in page 186.)

[Idani and Ledru, 2015] Idani, A. and Ledru, Y. (2015). B for modeling secure information systems. In *International Conference on Formal Engineering Methods*, pages 312–318. Springer. (Cited in page 40.)

[Kent, 2002] Kent, S. (2002). Model driven engineering. In *Integrated Formal Methods: Third International Conference, IFM 2002 Turku, Finland, May 15–18, 2002 Proceedings*, pages 286–298. Springer. (Cited in page 62.)

[Knight, 2002] Knight, J. C. (2002). Safety critical systems: challenges and directions. In *Proceedings of the 24th international conference on software engineering*, pages 547–550. (Cited in page 3.)

[Kordon et al., 2013] Kordon, F., Hugues, J., Canals, A., and Dohet, A. (2013). *Embedded systems: analysis and modeling with SysML, UML and AADL*. John Wiley & Sons. (Cited in page 30.)

[Laleau and Mammar, 2000] Laleau, R. and Mammar, A. (2000). An Overview of a Method and Its Support Tool for Generating B Specifications from UML Notations. In *The Fifteenth IEEE International Conference on Automated Software Engineering, ASE 2000, Grenoble, France, September 11-15, 2000*, pages 269–272. IEEE Computer Society. (Cited in page 81.)

[Laleau et al., 2010] Laleau, R., Semmak, F., Matoussi, A., Petit, D., Hammad, A., and Tatibouet, B. (2010). A first attempt to combine SysML requirements diagrams and B. *Innovations in Systems and Software Engineering*, 6(1):47–54. (Cited in pages XIII, 20, 21, 63, 92, 193 and 197.)

[Lamsweerde, 2008] Lamsweerde, A. v. (2008). Systematic Requirements Engineering-From System Goals to UML Models to Software Specifications. (Cited in pages 10 and 139.)

[Lecomte et al., 2017] Lecomte, T., Déharbe, D., Prun, É., and Mottin, E. (2017). Applying a formal method in industry: a 25-year trajectory. In *Formal Methods: Founda-*

tions and Applications: 20th Brazilian Symposium, SBMF 2017, Recife, Brazil, November 29—December 1, 2017, Proceedings 20, pages 70–87. Springer. (Cited in pages 66 and 196.)

[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 5.)

[Leveson, 2016] Leveson, N. G. (2016). *Engineering a safer world: Systems thinking applied to safety*. The MIT Press. (Cited in page 61.)

[Lima et al., 2017] Lima, L., Miyazawa, A., Cavalcanti, A., Cornélio, M., Iyoda, J., Sampaio, A., Hains, R., Larkham, A., and Lewis, V. (2017). An integrated semantics for reasoning about SysML design models using refinement. *Softw. Syst. Model.*, 16(3):875–902. (Cited in pages 44, 56 and 57.)

[Lodderstedt et al., 2002] Lodderstedt, T., Basin, D., and Doser, J. (2002). SecureUML: A UML-based modeling language for model-driven security. In *International Conference on the Unified Modeling Language*, pages 426–441. Springer. (Cited in page 40.)

[Mallet, 2015] Mallet, F. (2015). MARTE/CCSL for modeling cyber-physical systems. In *Formal Modeling and Verification of Cyber-Physical Systems*, pages 26–49. Springer. (Cited in page 30.)

[Mammar and Laleau, 2016] Mammar, A. and Laleau, R. (2016). On the use of domain and system knowledge modeling in goal-based Event-B specifications. In *International Symposium on Leveraging Applications of Formal Methods*, pages 325–339. Springer. (Cited in page 23.)

[Marques et al., 2014] Marques, M. R. S., Siegert, E., and Brisolara, L. (2014). Integrating UML, MARTE and SysML to improve requirements specification and traceability in the embedded domain. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 176–181. IEEE. (Cited in pages 51 and 54.)

[Matoussi et al., 2011a] Matoussi, A., Gervais, F., and Laleau, R. (2011a). A goal-based

approach to guide the design of an abstract Event-B specification. In *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, pages 139–148. IEEE. (Cited in pages 23, 24, 98, 102, 123, 144 and 199.)

[Matoussi et al., 2011b] Matoussi, A., Gervais, F., and Laleau, R. (2011b). Une premiere approche de tracabilite entre modeles d'exigences non-fonctionnelles et specifications abstraites Event-B. In *Actes du 29eme Congres INFORSID*, Unknown, Unknown Region. (Cited in pages 22 and 24.)

[Mazzini et al., 2016] Mazzini, S., Favaro, J. M., Puri, S., and Baracchi, L. (2016). CHESS: an Open Source Methodology and Toolset for the Development of Critical Systems. In Bordeleau, F., Bruel, J., Dingel, J., Gérard, S., Muccini, H., Mussbacher, G., and Voss, S., editors, *Joint Proceedings of the 12th Educators Symposium (EduSymp 2016) and 3rd International Workshop on Open Source Software for Model Driven Engineering (OSS4MDE 2016) co-located with the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), Saint Malo, France, October 3, 2016*, volume 1835 of *CEUR Workshop Proceedings*, pages 59–66. CEUR-WS.org. (Cited in page 41.)

[Mentré, 2016] Mentré, D. (2016). SysML2B: automatic tool for B project graphical architecture design using SysML. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 308–311. Springer. (Cited in page 42.)

[Miyazawa and Cavalcanti, 2014] Miyazawa, A. and Cavalcanti, A. (2014). Formal Refinement in SysML. In Albert, E. and Sekerinski, E., editors, *Integrated Formal Methods - 11th International Conference, IFM 2014, Bertinoro, Italy, September 9-11, 2014, Proceedings*, volume 8739 of *Lecture Notes in Computer Science*, pages 155–170. Springer. (Cited in pages 44, 56 and 57.)

[Muller et al., 2005] Muller, P.-A., Fleurey, F., and Jézéquel, J.-M. (2005). Weaving executability into object-oriented meta-languages. In *Model Driven Engineering Languages and Systems: 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005. Proceedings 8*, pages 264–278. Springer. (Cited in page 155.)

[Musset et al., 2006] Musset, J., Juliot, É., Lacrampe, S., Piers, W., Brun, C., Goubet, L., Lussaud, Y., and Allilaire, F. (2006). Acceleo user guide. *See also http://acceleo.org/doc/obeo/en/acceleo-2.6-user-guide.pdf*, 2:157. (Cited in pages 42, 86 and 172.)

[OMG, 1997] OMG (1997). OMG unified modeling language, uml. https://www.omg.org/spec/UML/1.1/. (Cited in pages 16 and 160.)

[OMG, 2005] OMG (2005). OMG: UML Profile for Schedulability, Performance, and Time Specification, v1.1. Object Management Group (January 2005), formal/05-01-02. (Cited in page 29.)

[OMG., 2006] OMG. (2006). Meta Object Facility (MOF) 2.0 Core Specification. (Cited in page 155.)

[OMG, 2007] OMG (2007). OMG systems modeling language, version 1.3. http://www.omgsysml.org/. (Cited in pages 16, 20, 26, 33, 49 and 160.)

[OMG, 2008] OMG (2008). OMG, UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2. (Cited in pages 29, 33 and 161.)

[OMG, 2014] OMG (2014). THE OBJECT CONSTRAINT LANGUAGE SPECIFICATION VERSION 2.4. https://www.omg.org/spec/OCL/. (Cited in page 155.)

[Openflexo, 2019] Openflexo (2019). Openflexo SysML/KAOS Tool. https://downloads.openflexo.org/Formose/. (Cited in page 156.)

[Papyrus, 2008] Papyrus (2008). Eclipse Papyrus Modeling environment. https://www.eclipse.org/papyrus/. (Cited in pages 18, 42, 159 and 160.)

[Papyrus, 2016] Papyrus (2016). org.eclipse.papyrus-moka.git. (Cited in page 161.)

[Pierra, 2006] Pierra, G. (2006). Context-explication in conceptual ontologies: Plib ontologies and their use for industrial data. *Journal of Advanced Manufacturing Systems*, 5:243–254. (Cited in page 92.)

[Poorhadi et al., 2022] Poorhadi, E., Troubitsyna, E., and Dán, G. (2022). Analysing the Impact of Security Attacks on Safety Using SysML and Event-B. In *International Symposium on Model-Based Safety and Assessment*, pages 170–185. Springer. (Cited in pages 43 and 57.)

[ProB, 2003] ProB (2003). The ProB Animator and Model Checker. `https://prob.hhu.de/`. (Cited in pages 38, 67, 86, 91 and 197.)

[Riviere et al., 2022] Riviere, P., Singh, N. K., and Ameur, Y. A. (2022). EB4EB: A framework for reflexive Event-B. In *26th International Conference on Engineering of Complex Computer Systems, ICECCS 2022, Hiroshima, Japan, March 26-30, 2022*, pages 71–80. IEEE. (Cited in page 186.)

[Rodhe and Karresand, 2015] Rodhe, I. and Karresand, M. (2015). *Overview of formal methods in software engineering*. Totalförsvarets forskningsinstitut (FOI). (Cited in page 34.)

[Ross and Schoman Jr, 1976] Ross, D. T. and Schoman Jr, K. E. (1976). Structured analysis for requirements definition. In *Proceedings of the 2nd international conference on Software engineering*, page 1. IEEE Computer Society Press. (Cited in page 18.)

[Said et al., 2009] Said, M. Y., Butler, M., and Snook, C. (2009). Language and tool support for class and state machine refinement in UML-B. In *International Symposium on Formal Methods*, pages 579–595. Springer. (Cited in page 44.)

[Salunkhe et al., 2021] Salunkhe, S., Berglehner, R., and Rasheeq, A. (2021). Automatic Transformation of SysML Model to Event-B Model for Railway CCS Application. In *International Conference on Rigorous State-Based Methods*, pages 143–149. Springer. (Cited in pages 42 and 57.)

[Schürr, 1994] Schürr, A. (1994). Specification of graph translators with triple graph grammars. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163. Springer. (Cited in page 42.)

[Sengupta and Hitzler, 2014] Sengupta, K. and Hitzler, P. (2014). *"Web Ontology Lan-*

guage (OWL)", pages 2374–2378. Springer New York, New York, NY. (Cited in pages 92 and 156.)

[Singhoff et al., 2005] Singhoff, F., Legrand, J., Nana, L., and Marcé, L. (2005). Scheduling and memory requirements analysis with AADL. *ACM SIGAda Ada Letters*, 25(4):1–10. (Cited in page 28.)

[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 pages 39 and 42.)

[Sokolsky, 2005] Sokolsky, O. (2005). The montana toolset: OSATE plugins for analysis and code generation. In *Proceedings of AADL Workshop*. (Cited in page 28.)

[Steinberg et al., 2008] Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. (2008). *EMF: eclipse modeling framework*. Pearson Education. (Cited in page 159.)

[Tueno et al., 2017a] Tueno, S., Laleau, R., Mammar, A., and Frappier, M. (2017a). Formal Representation of SysML/KAOS Domain Model (Complete Version). *arXiv preprint arXiv:1712.07406*. (Cited in page 123.)

[Tueno et al., 2017b] Tueno, S., Laleau, R., Mammar, A., and Frappier, M. (2017b). The SysML/KAOS Domain Modeling Approach. *arXiv preprint arXiv:1710.00903*. (Cited in page 23.)

[Tueno et al., 2017c] Tueno, S., Laleau, R., Mammar, A., and Frappier, M. (2017c). Towards using ontologies for domain modeling within the SysML/KAOS approach. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 1–5. IEEE. (Cited in pages 23 and 92.)

[Tueno Fotso, 2019] Tueno Fotso, S. J. (2019). *Vers une approche formelle d'ingénierie des exigences outillée et éprouvée*. PhD thesis. Thèse de doctorat dirigée par Laleau, Régine-Frappier, Marc et Mammar, Amel Informatique Paris Est 2019. (Cited in page 157.)

[Tueno Fotso et al., 2018] Tueno Fotso, S. J., Mammar, A., Laleau, R., and Frappier, M.

(2018). Event-B expression and verification of translation rules between SysML/KAOS domain models and B system specifications. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 55–70. Springer. (Cited in pages 98, 101, 144 and 199.)

[Van Lamsweerde, 2000] Van Lamsweerde, A. (2000). Requirements engineering in the year 00: a research perspective. In *Proceedings of the 22nd international conference on Software engineering*, pages 5–19. ACM. (Cited in page 19.)

[Van Lamsweerde, 2001] Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In *Proceedings fifth ieee international symposium on requirements engineering*, pages 249–262. IEEE. (Cited in page 19.)

[Van Lamsweerde, 2009] Van Lamsweerde, A. (2009). *Requirements engineering: From system goals to UML models to software*, volume 10. Chichester, UK: John Wiley & Sons. (Cited in pages 20, 21 and 50.)

[Vergnaud, 2005] Vergnaud, T. (2005). The Ocarina Tool Suite. In *AADL Workshop*. (Cited in page 28.)

[Wang et al., 2019] Wang, F., Yang, Z.-B., Huang, Z.-Q., Liu, C.-W., Zhou, Y., Bodeveix, J.-P., and Filali, M. (2019). An approach to generate the traceability between restricted natural language requirements and AADL models. *IEEE Transactions on Reliability*, 69(1):154–173. (Cited in pages 53, 54, 56 and 57.)

[Weidmann et al., 2019] Weidmann, N., Anjorin, A., Robrecht, P., and Varró, G. (2019). Incremental (unidirectional) model transformation with emoflon:: Ibex. In *International Conference on Graph Transformation*, pages 131–140. Springer. (Cited in page 43.)

[Wing, 1990] Wing, J. M. (1990). A specifier's introduction to formal methods. *Computer*, 23(9):8–22. (Cited in page 34.)

[Yu, 1997] Yu, E. S. (1997). Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of ISRE'97: 3rd IEEE International Symposium on Requirements Engineering*, pages 226–235. IEEE. (Cited in page 19.)

[Zhu, 2005] Zhu, H. (2005). *Software design methodology: From principles to architectural styles*. Elsevier. (Cited in page 62.)

Une approche formelle pour la modélisation d'architectures de haut niveau de systèmes complexes alignées avec les modèles d'exigences

Résumé : Les systèmes complexes sont un ensemble de sous-systèmes reliés entre eux pour représenter un tout intégré. La conception de ces systèmes devrait représenter les interactions entre leurs sous-systèmes. Pour y parvenir, des modèles graphiques sont généralement recommandés pour spécifier, visualiser, comprendre et documenter le système de manière simple. Cependant, lorsqu'on considère des systèmes critiques pour la sureté où les conséquences d'une défaillance entraînent des pertes de vie, des dommages matériels ou environnementaux importants, les langages graphiques ne sont pas suffisants car ils ne sont que semi-formels et ne permettent pas un raisonnement formel et rigoureux nécessaires pour vérifier les propriétés de sûreté et de sécurité. Enfin, la qualité d'un système dépend de la mesure dans laquelle il répond à ses exigences. La traçabilité des exigences est largement reconnue comme un élément crucial de tout processus de développement de système rigoureux, en particulier pour la conception de systèmes complexes critiques.

Pour répondre à ces enjeux, la thèse vise à définir une approche d'alignement entre les modèles d'exigences et les modèles d'architecture de haut niveau pour les systèmes complexes critiques, permettant ainsi de spécifier des liens de traçabilité entre ces deux entités et de garantir que les modèles d'architecture de haut niveau satisfont les besoins requis des parties prenantes. Ceci est réalisé en utilisant des spécifications formelles pour vérifier d'une part l'exactitude et la cohérence des modèles d'architecture de haut niveau et d'autre part la cohérence des liens d'alignement établis.

Mots clés : Architecture de haut niveau, Exigence, Alignement, SysML/KAOS, SysML, Transformation de modèles, Spécification formelle, Méthode Event-B.

A formal approach for modeling high-level architectures of complex systems aligned with requirement models

Abstract : Complex systems are a collection of sub-systems linked together to represent an integrated whole. The design of such systems should represent the interactions between their sub-systems. To achieve this, graphical models are generally recommended to specify, view, understand, and document the system in a simple way. However, when considering safety-critical systems where the consequences of a failure result in loss of life, significant property or environmental damage, graphical languages are not sufficient since they are only semi-formal and do not allow formal and rigorous reasoning necessary for verifying safety and security properties. Lastly, the quality of a system depends on the degree to which it fulfills its requirements. Requirements traceability is broadly recognized as a crucial element of any rigorous system development process, especially for the design of critical complex systems.

To cope with these issues, the thesis aims to define an approach of alignment between requirements models and high-level architecture models for safety-critical complex systems, thus allowing to specify traceability links between these two entities and to guarantee that high-architecture models fulfills required stakeholders needs. This is achieved by using formal specifications to verify firstly the correctness and consistency of high-level architecture models and secondly the consistency of the established alignment links.

Keywords : High level architecture, Requirement, Alignment, SysML/KAOS, SysML, Model transformation, Formal specification, Event-B Method.