THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 2 - Jean Jaurès

Présentée et soutenue par

Ronan BADUEL

Le 30 septembre 2019

Une approche intégrée d'ingénierie des systèmes ferroviaires basée sur les modèles et prenant en charge leur validation

---

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Jean-michel BRUEL et Iulian OBER

Jury

M. Vincent CHAPURLAT, Examinateur
M. Ludovic APVRILLE, Examinateur
M. Marius BOZGA, Examinateur
Mme Isabelle PERSEIL, Examinatrice
M. Jean-michel BRUEL, Directeur de thèse
M. Iulian OBER, Co-directeur de thèse
M. Drira KHALIL, Président

An integrated model-based early validation approach for railway systems

Ronan Baduel, PhD Student at Bombardier Transport

Acknowledgments

Though a personal work by nature, many people were involved in this thesis, and deserve to be cited. I would like to thank first Jean-Michel Bruel, Iulian Ober, Mohammad Chami and Eddy Doba, which have overseen my activities these past three years and helped me achieved my goals. I would then like to thank Vincent Chapurlat, who helped me find this thesis, reviewed it as a rapporteur and has been the professor that taught me system engineering before that. A special mention to the functional architecture team in BT. They welcomed me in the company, helped me with my work and made my period in Crespin very enjoyable. I hence thank Jerome, Samy, Laurent, Lucile, Ali, Mouhammadou, Maria, Carine, Aurelien, Emmanuel, Carlos, Delphine, Pierre and all the others in BT that helped during my time there.

I thank my family, which greatly supported me in my studies up to this Phd. A special thank to my brother, who helped proof-reading this work without counting the hours.

I finally thank the jury, who agreed to evaluate this work and provided both a serious scrutiny during the defense of the thesis and welcomed advices afterwards. I am liable to omit people, so I would like to thank the readers, be they contributors or persons interested in this work, for the value they bring to this work.

Abstract

Systems engineering is a field of study multidisciplinary by nature, using knowledge and techniques from different fields, from mathematics and computer science to organizational theory. As such, it encompasses hundreds of different jobs and practices. It is a discipline used to develop complex systems, meaning systems being composed of different elements linked by relationships. While specifying, developing and validating individual components may not be an issue, doing the same for the whole system is often difficult.

A complex system such as a train is developed by several teams of engineers with different viewpoints. We consider here the practice in Bombardier Transport (BT), a train manufacturing company. In BT, the functional architecture alone necessitates to be divided among different engineers, using different scopes of study. It is expressed at system level, meaning the level of abstraction of a vehicle (consist) forming a train. The system is not defined as a global element, it is induced by different pieces of information. This information is specified by several engineers or even teams of engineers, even when considering the system only as a whole without the elements composing it.

The requirements, meaning the expectations expressed regarding the system, often

include information relative to sub-systems or components. In order to conceive the system, the information characterizing it should be expressed at system level or be linked to global properties or constraints. It means abstracting the information when possible. Considering the amount of information used to characterize a system, abstracting it helps to make it more manageable while overlooking unnecessary details which are not relevant to a global perspective. It can then be integrated into a coherent whole.

Achieving the abstraction and the integration of the information requires having traceability, so that every relevant piece of information is considered when studying the whole system. Relevant information about the whole system is taken into account when studying each of its parts. The notion of relevance mentioned here means that it has an impact on the part that is currently studied. For example, a train, which is a whole system, will generally not move if one of its doors, which is a component, is open. On the other hand, a door component will not open if the whole train is moving. In this example, with the train being considered as a complex system, information regarding each individual door will not be considered when studying the train as a whole, just as a door will not receive information regarding characterizing the train as a whole. It is however necessary to express, correlate and trace information between different levels of abstraction to know and/or specify a system and its behavior.

The goal pursued in this thesis is to provide a method to integrate information regarding a train system during the design phase, enabling the specification, representation and validation of its behavior. To this end, several solutions are developed: concepts characterizing the system and its behaviors are developed and applied in models. The way the models are created and the information expressed through them is automatically checked using verification rules. The information and the models are then integrated. The integration is specified and checked using constraints and properties based on the system concepts developed.

Résumé

L'ingénierie système est par nature un domaine d'étude multidisciplinaire. Elle fait appel à des connaissances et des techniques d'origines variées, allant des mathématiques et de l'informatique à la théorie de l'organisation. Ainsi, elle couvre des centaines de métiers et de pratiques différents. C'est une discipline employée pour développer des systèmes, c'est-à-dire des ensembles composés de différents éléments liés par des relations. Bien que spécifier, développer et valider chacun de ces éléments séparément soit réalisable, faire de même pour le système dans sa globalité est souvent difficile.

Un système complexe tel qu'un train est développé par différentes équipes d'ingénieurs, chacune adoptant un point de vue différent. On s'intéresse ici aux pratiques au sein de Bombardier Transport (BT), une entreprise fabriquant des trains. A lui seul, le développement d'une architecture fonctionnelle d'un train au sein de BT nécessite de répartir le travail entre différents ingénieurs, chacun s'intéressant à un cadre d'étude particulier. Une telle architecture est considérée au niveau d'abstraction du système étudié, qui est un véhicule formant un train ou une partie de train. Le système n'est alors pas défini comme un ensemble global, il est induit par différentes informations. Celles-ci sont fournies par différents ingénieurs ou équipes d'ingénieurs, quand bien même on s'intéresse au système et non pas aux éléments qui le composent.

Les exigences, c'est-à-dire les attentes exprimées vis-à-vis du système, comprennent généralement des informations relatives à ses sous-systèmes ou ses composants. Pour concevoir un système, les informations qui le caractérisent doivent être exprimées à son propre niveau d'abstraction ou bien être liées à des propriétés ou contraintes globales qui lui sont propres. Cela demande d'abstraire les éléments d'information chaque fois que cela est possible. Etant donné la quantité d'informations relatives au système, les abstraire facilite ainsi leur gestion tout en occultant les détails superflus. On peut alors les intégrer au sein d'un tout cohérent.

Abstraire et intégrer les informations relatives au système nécessite de pouvoir les tracer, de façon à ce que toute information pertinente soit prise en compte lors de l'étude du système global, et que toute information pertinente issue du système global soit prise en compte lors de l'étude d'un des éléments qui le compose. La notion de pertinence exprimée ici se rapporte à l'impact que cela peut avoir sur l'objet étudié.

Un train, en tant que système global, ne pourra généralement pas bouger si l'une de ses portes, étant donc un de ses composants, est ouverte. De la même façon, une porte, en tant que composant, ne pourra pas s'ouvrir si le train est en mouvement. On voit ainsi que l'étude du train dans son ensemble ne tient pas compte des informations spécifiques à chaque porte, tout comme chaque porte ne considère pas l'ensemble des informations caractérisant le train. Il est en revanche nécessaire d'exprimer, corréler et tracer les informations entre les différents niveaux d'abstractions pour connaître et/ou spécifier un système et son comportement.

Le but de la thèse présentée ici est de fournir une méthode permettant d'intégrer les informations caractérisant un système de train au fur et à mesure de sa conception, permettant ainsi la spécification, la représentation et la validation de son comportement. Pour ce faire, différentes solutions sont développées : des concepts caractérisant le système et son comportement sont développés et appliqués au sein de modèles. La façon dont les modèles sont créés et dont les informations dont exprimées à travers eux est vérifiée automatiquement à l'aide de règles de vérification. Les informations et les modèles sont alors intégrés. L'intégration elle-même est spécifiée et vérifiée à l'aide de contraintes et de propriétés basées sur les concepts développés vis-à-vis du système.

Abbreviations

BDD: Block Definition Diagram

BT: Bombardier Transport

BT SysMM: Bombardier Transport Modeling Method

DITLOTT: Day In The Life-cycle Of The Train

DSML: Domain Specific Modeling Language

FA: Functional Analysis

FB: Functional Block

FBS: Functional Breakdown Structure

FC: Functional Context

FMI: Functional Mock-up Interface

FMU: Functional Mock-up Unit

HIL: Hardware In the Loop

HMI: Human Machine Interface

HW: Hardware

IBD: Internal Block Diagram

OA: Operational Analysis

OB: Operability Analysis

OBS: Operational Breakdown Structure

OMG: Object Management Group

MIL: Model In the Loop

SHL: System Hierarchical Level

PI: Product Introduction

RBS: Requirement Breakdown Structure

SIL: Software In the Loop

SOI: System Of Interest

SysMM: System Modeling Method

SW: Software

TA: Technical Analysis

TCMS: Train Control Management System

T0: Train 0

VB: Virtual Bird

V&V: Verification and Validation

WBS: Work Breakdown Structure

Contents

1 Introduction **12**

1.1 Context 12

1.2 Needs 14

1.2.1 Specify the train behavior 14

1.2.2 Check the specifications for errors 15

1.2.3 Enable communication of information between engineering teams . 17

1.2.4 Summary of needs 19

1.3 Challenges 20

1.4 Target 23

1.5 Contributions 26

1.6 Organization of the thesis 27

I Problem analysis **28**

2 Context **29**

2.1 Bombardier Transport Modeling Method 29

2.1.1 BT SysMM overview 29

2.1.2 Operability 35

2.1.3 System development at consist level 46
2.2 Verification and validation process in BT 50
2.2.1 Organization 51
2.2.2 MIL: Cameo SysML 52
2.2.3 SIL: Virtual Bird 56
2.2.4 HIL: TRAIN0 58
2.3 Needs analysis 59
2.3.1 List of issues 59
2.3.2 Derived needs 62
2.4 State of the art 63
2.4.1 Specifying the train behavior 63
2.4.2 Checking the specifications for errors 65
2.4.3 Enabling communication of information between engineering teams 67
2.5 Contributions 68
2.5.1 Concepts of states and modes 68
2.5.2 Model verification method 69
2.5.3 Behavior verification method and model 69

3 Background: system theory and engineering 70

3.1 System theory and definition 71
3.1.1 System concept 71
3.1.2 System representation 75
3.1.3 Method 77
3.2 Systems engineering 78
3.2.1 System concept 79

3.2.2 System representation 81
3.2.3 Method 84
3.3 Synthesis 85

II Contributions **87**

4 Concepts of states and modes **88**
4.1 Concept of State 90
4.1.1 State of the art 90
4.1.2 Analysis 91
4.1.3 Definition 93
4.1.4 Example 95
4.2 Concept of Mode 95
4.2.1 State of the art 95
4.2.2 Analysis 96
4.2.3 Definition 98
4.2.4 Example 99
4.3 Application 99
4.3.1 Definition of train states 99
4.3.2 Definition of train modes 103
4.3.3 Verification of the behavior 106

5 Model verification method **107**
5.1 Models V&V 107
5.2 Background on BT SysMM 108

5.3 State of the art 109
5.4 BT SysMM V&V 110
5.4.1 Method Stakeholders 110
5.4.2 V&V Method Overview 114
5.4.3 Benefits 116
5.5 Use case example 117

6 Behavior verification method and model 120

6.1 Presentation 120
6.1.1 Context 120
6.1.2 Issues 121
6.1.3 Related works 122
6.1.4 Method 123
6.1.5 Case study 123
6.2 Behavior description through states 123
6.2.1 States in the case study 123
6.2.2 State constraints 124
6.2.3 State constraints in the case study 126
6.2.4 Use case pre-conditions 127
6.2.5 Use case pre-conditions in the case study 128
6.3 Verification method 128
6.3.1 State constraints verification 129
6.3.2 Use case preconditions verification 130
6.3.3 Results 130
6.4 Execution model 132

6.4.1 Holonic structure for states . . . 132
6.4.2 Structure of the behavior . . . 135
6.4.3 Structure of modes in the case study . . . 137
6.5 Synthesis . . . 138
6.5.1 Method . . . 138
6.5.2 Traceability . . . 138

7 Conclusion 140

7.1 Contributions synthesis . . . 140
7.1.1 Concepts of states and modes . . . 140
7.1.2 Model verification method . . . 141
7.1.3 Behavior verification method and model . . . 143
7.2 Impact of the global solution . . . 144
7.2.1 Progress toward the overall objective of BT . . . 144
7.2.2 Consequences on BT . . . 145
7.2.3 Consequences on MBSE . . . 145
7.3 Future work . . . 146
7.3.1 Remaining work to be done in the company . . . 146
7.3.2 Research challenge follow-up and perspectives . . . 148

Chapter 1

Introduction

This PhD provides solutions to specify, integrate and verify a train system behavior. In this introduction, the context of the work is explained, as well as the problematic that led to the development of the solutions presented in the next part.

1.1 Context

The work presented here results from a research project conducted in the industry and require some context. Bombardier Transport (BT), a train manufacturer, has been developing its systems engineering practice for several years, increasing the use of models, tools and general methods. BT is a big company of more than thirty thousand people that focuses its activity on six main implementations all around the world. Albeit being one of the leading company in train manufacturing, it faces a strong competition, and has an increasing need to save time and money while improving the quality of its products to satisfy the clients requirements. In order to get a contract, BT often has to be the quickest to deliver a product. This requires to build a satisfactory product the first time, or to face delay penalties otherwise.

We can summarize the current goals of BT as follows:

- Avoiding delay penalties.
- Limiting the number of iterations during the development of a train product.
- Providing higher quality products that meet the clients' expectations.

The problem considered here does not deal with the product itself, but the way it is developed. The Train technologies and components have remained roughly the same for the last decade, so the main part in developing or adapting a product relates to the specification of *functions*. A function is something performed by a train, taking inputs and providing outputs. A *functional architecture* is a hierarchical arrangement of functions [1]. Its definition leads to the choice and integration of adequate, often pre-existing components. The issue for BT is hence to specify what the train must do, when, and how. The object of study is hence the *behavior* resulting from the functional architecture being developed. Interviewing BT's engineers, it appeared that the concept of behavior could be understood as "the peculiar reaction of a thing under given circumstances", which is one of the definition given in the standard ISO/IEC/IEEE 24765 [1].

The current development process has an issue: the concept of behavior is understood but not formally defined in BT. The behavior of a train system as a whole is hence not specified, it is first induced by requirements and individual specifications, and later on by the design. One could say that the system and its behavior only exist in the engineers' minds, using their experience and knowledge.

Information regarding the train behavior exists but needs to be captured. Since the development of a train is based on its functions and behaviors, engineers have to manipulate abstract information. Be it communicating specifications or searching for errors, the information contained must be correctly represented and understood by all. This is an issue in BT, as a same train project can involves development teams from different departments, countries or companies. Information actually represented can be expressed differently depending on the tool, the method, the language or the point of view adopted. This can lead to loss of information, misunderstanding or inconsistencies.

Clients and company requirements regarding the solution have to be taken into account all along the development process. The current tendency for the development of a design is to favor the reutilization of requirements, specifications and existing solutions whenever possible. BT is currently developing product families from which existing solutions can be chosen and adapted depending on the project. This once again requires a way to formalize and represent knowledge regarding the train and its behavior.

The kind of issues presented here lead BT to adopt, develop and improve Model-Based Systems Engineering (MBSE) [2] solutions, using SysML [3]. MBSE is the current and favored approach to the development of complex system [4], supported by research work and opportunities, which is why it is considered as a solution to BT needs and problems.

BT has developed a modeling method regarding the functional specification of a train system, the BT System Modeling Method (BT SysMM) [5], which is implemented through a SysML profile. This method is based on the standard ISO/IEC/IEEE 15288-2015 [6] as well as studies done to develop functional architecture with SysML [7]. BT SysMM is presented in further details in the chapter 2.

To reach the goals previously listed, BT needs to improve its development process. This requires to satisfy the following needs:

N_1. Specify the train behavior.

N_2. Check the specifications for errors.

N_3. Enable communication of information between engineering teams.

Those needs will be satisfied by solutions developed in an MBSE context, compatible with SysML and improving or completing the BT SysMM.

1.2 Needs

We analyze in this section the global needs of BT and decompose them into a list of detailed needs to be satisfied by the solutions developed.

1.2.1 Specify the train behavior

We first consider the specifications relating to the train behavior. Elements of the behavior are specified using *scenarios* and *use cases*. Scenarios are *"step-by-step description of a series of events that occur concurrently or sequentially"* [1]. In BT, they are expressed using activity diagrams and sequence diagrams, or even drawn without an official modeling language. They correspond to sequences of operations performed in interactions with users and external systems. Those operations are called *use cases*. They correlate with services or capabilities performed by the train from the users' point of view. Use cases are specified using requirements, activities or the actual use case SysML element. They are defined independently from the scenarios: a same use case can be mentioned in different scenarios.

A scenario corresponds to an intended utilization of a train system and is used to model sequence of use cases performed in a given set of circumstances, meaning the situation the system is in. Scenarios do not cover all possible utilizations of the train, and the circumstances they express are neither formal nor exhaustive.

A scenario can be considered as a piece of the train potential behavior. Depending on how the execution of the use cases is constrained and the design satisfies the behavior specified in the scenarios, we can obtain different train behaviors: integrating different behavior results in new ones, through the phenomenon of *emergence* [8].

Information on the circumstances of realization of the use cases and their integration into a global behavior is either incomplete, induced or “hidden” in textual requirements or specific models such as scenarios, which have limited scope. For example, a use case can indicate how to change the speed of windscreen wipers, and another to activate windscreen wipers for a duration after triggering a washing mechanism. In the model and documentation, there is nothing that says at which speed the wipers should be set at after washing, if there is a need for a timed activation if the wipers were already activated, etc. The example is trivial, and may be contained in a specific scope, but it is symptomatic of a lack of integration of the train capabilities.

The integration itself leads to the specification, comparison and prioritization of capabilities depending on the circumstances. We need to complete the existing modeling method with models enabling to specify and integrate the whole system behavior. Based on current specifications regarding the behavior, there is a need to:

N_1.1. Specify dynamic aspects between use cases.

N_1.2. Correlate the information expressed in different scenarios and use cases.

N_1.3. Formalize the circumstances enabling the execution of each use case.

1.2.2 Check the specifications for errors

We now consider the second objective, which relates to detecting errors in the specifications. In order to make a good product on the first try, errors have to be avoided or detected early. A general observation in the industry is that the later an error is detected, the more the cost [9]. This is why current solutions and improvements focus on the early stage of a train's design. There are two aspects to consider when checking if the specifications are correct or not: it is necessary to check (i) whether the specifications are consistent with a train product; and (ii) whether the obtained product satisfies the initial expectations. This 'checking' process corresponds to two kinds of activities called *verification* and *validation*.

Verification and *validation* are two concepts frequently used in systems engineering, and need to be defined. By default, the terms used in this thesis correspond to the definitions provided in the x ISO/IEC/IEEE 24765 standard on systems engineering vocabulary [1]. We define here the terms of V&V using the common definition and understanding of these concepts [10]:

- Verification: answering the question "are we building the system right?"
- Validation: answering the question "are we building the right system?"

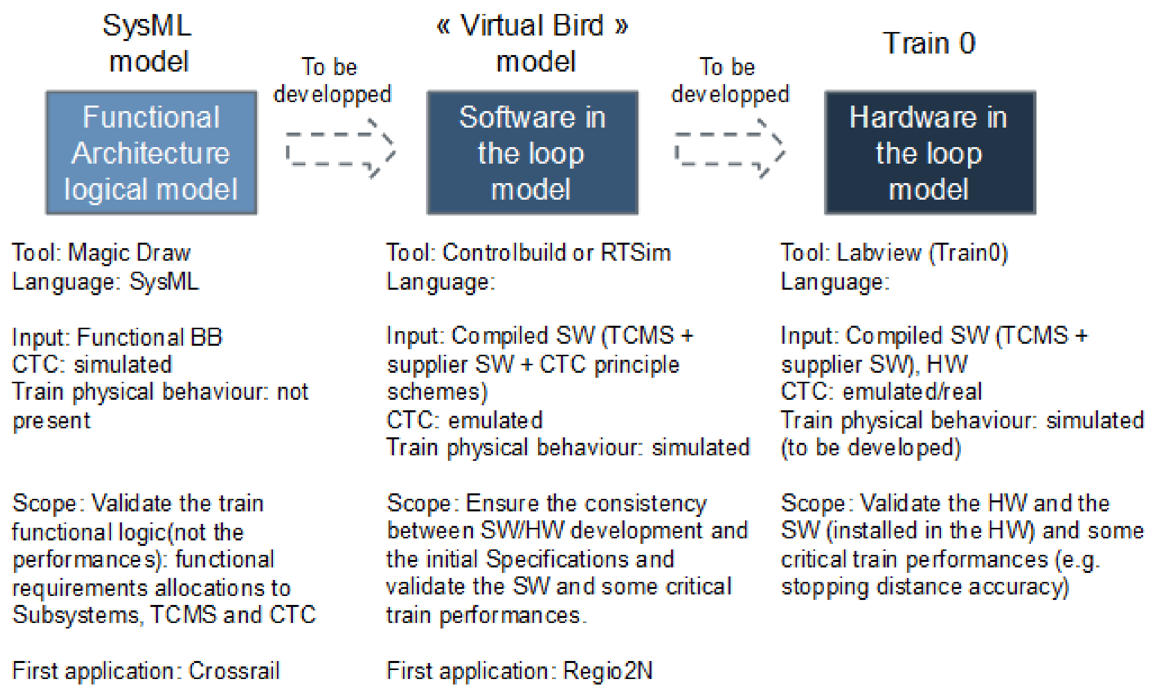

Figure 1.1: BT current validation process

We consider now BT's current validation process, illustrated in Figure 1.1. It follows three steps: Model In the Loop (MIL), Software In the Loop (SIL) and Hardware in the Loop (HIL). MIL is a validation conducted on a co-simulation through SysML models. SIL is performed on the developed software and emulated hardware through co-simulation. HIL is performed either on a bench test or a real train.

BT aims for a tool chained method that provided V&V solutions for MIL, SIL and HIL steps of the validation process. The method should focus on filling the gap between each of these steps so as to ensure traceability and continuity of the validation. SIL and HIL already having V&V solutions working, though not fully developed yet, the second objective was to develop a way to verify and validate the behavior based on SysML models, which are themselves developed over several steps following BT SysMM.

BT currently checks whether a train system performs the expected functions or not, using tests. Checking that the train *behaves* as expected, meaning that it can perform the right functions in the right circumstances is the responsibility of the functional engineers. They check behaviors that are part of the expected utilization of the system, expressed using scenarios. This can be done by means of execution or simulation on a co-simulation or a bench test, that other teams are in charge of developing.

Understanding the system as a whole and knowing how it should *behave* is something that is checked by engineers through their knowledge and experience, as well as their understanding of the requirements. There is no formal validation method regarding the behavior, engineers rely on manual execution or tests created from scenarios and their experience to evaluate whether the system is well-made and satisfactory or not. Just as all possible dynamics between different capabilities may not have been specified, they may not be verified and validated either. There are no formal requirements on what should be validated regarding the behavior.

We can summarize the needs for the system behavior V&V as follows:

N_2.1. Define V&V requirements for the integrated system behavior

N_2.2. Provide V&V solutions of the behavior based on SysML models

N_2.3. Ensure traceability and continuity of specifications and validation results

N_2.4. Fill the gaps between tools, models and development steps

1.2.3 Enable communication of information between engineering teams

The definitions provided earlier are only relevant when talking about a system, here the train. As there a need to properly represent and communicate information regarding the system, there is a need to ensure that those representations are well made and correspond to expectations of the people receiving and using them. In an MBSE context, those representations are models. There is hence the need to verify and validate models, not the system itself. We will hence talk of system V&V, and model V&V. In the case of model V&V, we use the definitions of the IEEE 1012-2012 standard [11]:

- Verification: ensuring that the models created have been correctly built.
- Validation: ensuring that the system represented by the models matches the requirements traced to the information displayed or induced.

As there is a need for the exchange and reuse of information, there is a need to check that any kind of model created as part of the development process will be created and understood in the same way by any engineer. This implies to verify the model. A modeling language comes with its own rules which force the model to be constrained and avoid ambiguity in the way it is built. Beyond the fact that the models are correct, the chosen modeling language has to carry a unique semantics regarding the real system characterized by the models. This is not validation, this is verification, *i.e.,* the semantic is not checked but rather the models are verified with respect to the semantic rules of the chosen modeling language.

We come back to the need to formalize information in models. Concepts have to be expressed to characterize the real system and its behavior. Defining all concepts to be used means creating an *ontology.* An ontology is a *"logical structure of the terms used to describe a domain of knowledge"* [1]. Van Ruijven explains the needs of ontology in MBSE [12]. Supposing the creation of a formal ontology is achieved, there is a need to verify the model accordingly, to see if its elements properly express the corresponding concepts.

Any verification solution will have to be deployed and used in all BT implementations. It shall be used to standardize the way models are created and understood. It hence has to be part of BT SysMM.

The needs toward enabling proper communication of a train specifications in BT can be listed as follows:

N_3.1. Define an ontology that enables us to express system concepts in all models

N_3.2. Develop a model verification solution according to the ontology and modeling method provided

N_3.3. Implement this solution in a method supporting the system development

1.2.4 Summary of needs

We obtain the following list of needs:

N_1. Specifying the train behavior.

- **N_1.1.** Specify dynamic aspects between use cases.
- **N_1.2.** Correlate the information expressed in different scenarios and use cases.
- **N_1.3.** Formalize the circumstances enabling the execution of each use case.

N_2. Checking the specifications for errors.

- **N_2.1.** Define V&V requirements for the integrated system behavior
- **N_2.2.** Provide V&V solutions of the behavior based on SySML models
- **N_2.3.** Ensure traceability and continuity of specifications and validation results
- **N_2.4.** Fill the gap between tools, models and development steps

N_3. Enabling communication of information between engineering teams.

- **N_3.1.** Define an ontology that enables us to express system concepts in all models
- **N_3.2.** Develop a model verification solution according to the ontology and modeling method provided
- **N_3.3.** Implement the solution in a method supporting the system development

Before addressing those needs, it is necessary to consider the scope in which solutions must be developed and what are the constraints and limitations to consider. More importantly, we have to identify the challenges toward solving BT issues. Such challenges will be linked to current research work in MBSE.

1.3 Challenges

In this section, we link the needs of BT to scientific challenges in MBSE. We reference several sources to sum up the challenges faced in an MBSE approach.

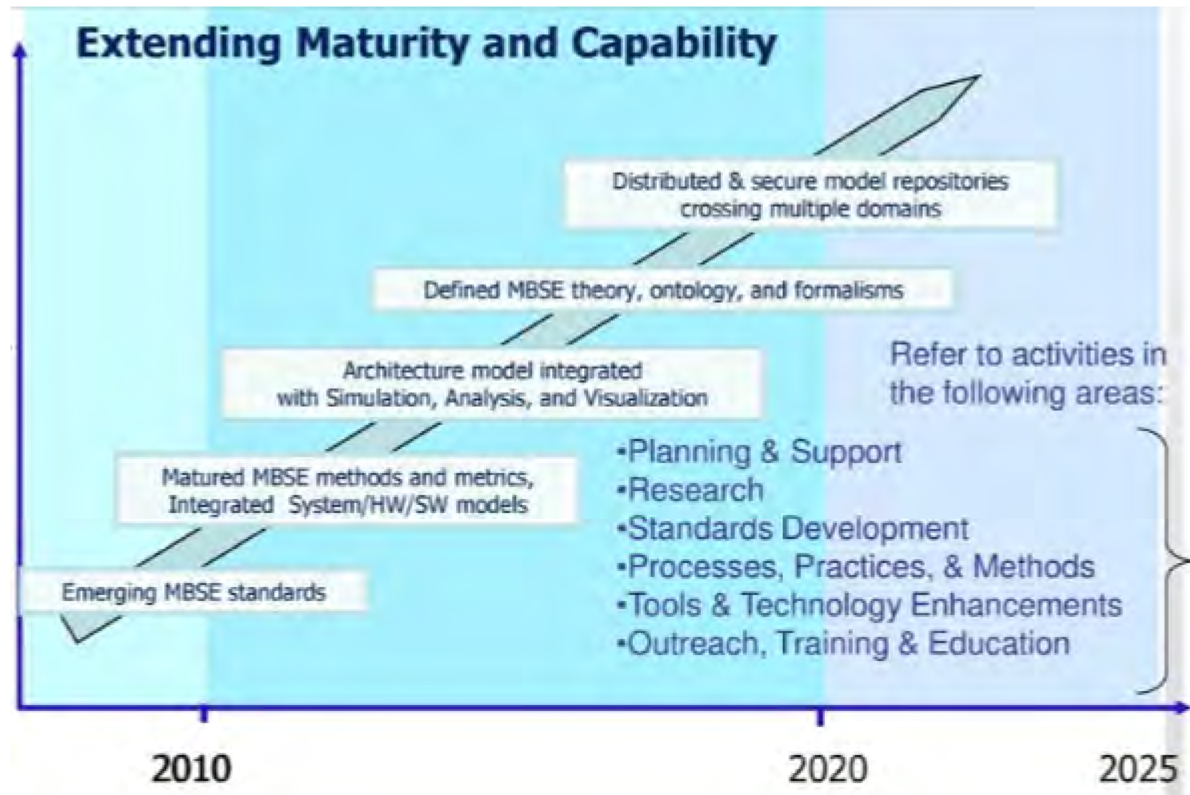

Figure 1.2: INCOSE MBSE roadmap [4]

We mainly refer to the MBSE roadmap, shown in Figure 1.2. This roadmap shows that MBSE is evolving and has yet to reach maturity. Developing and adapting MBSE solutions for a company is in itself a challenge [13]. BT needs correspond to the need for a *well-defined MBSE*, as described in the roadmap.

The issues encountered in BT can be compared to those addressed by Ingham [14]:

- Sub-system-level functional decomposition fails to express the whole system behavior.
- There is a gap between the requirements and their implementation.
- The system behavior is not always explicitly specified.

The first challenge presented by Ingham can be linked to the notion of emergence we presented earlier. The second can be linked to the needs in BT **N_2.3.** and **N_3.**, as the proper information must be traced and communicated. The third correspond to the need **N_1.**

BT needs to verify and validate train systems behavior. Works [15, 16, 17] show that one should specify, and if possible validate, the *expected* behavior of the system *as a whole* using requirements and scenarios, before any design or implementation. The main issue encountered to achieve such a task is that a model of the system is required to define and support the behavior. The lack of a formal description of the system at specification level prevents its automatic verification [18]. Rather than just specifying what the system-to-be does, we have to specify "what" system we want to obtain [18]. Having an integrated model of the system at the early step of design is a challenge.

We can see in [19] that requirements tend to be textual-based. Using models to express requirements is not a common practice, as explained in [20]. Developing an MBSE approach for requirements engineering is a challenge [21]. Furthermore, SysML, which is the language we want to use here, manipulates requirements as blocks with textual properties. New ways to use SysML elements have to be created in order to specify requirements in models, as presented in [20].

How to perform validation on system behavior is a field of research in systems engineering [22]. Studies that show that the issues considered in BT are parts of different challenges that are rarely or ill-resolved in common solutions. For example, emergent behavior, differences in granularity of the information or the fact that implementation details can pollute the model are discussed in [23]. The issue regarding various levels of detail regarding the specifications is also highlighted in [16].

Another study [24] explains that validation is seldom attempted regarding the dynamics of a system, and mentions incomplete information as a source of uncertainty when modeling and validating complex systems. Information can be found partially in the requirements, but it is often incomplete. Supposing the information is available, it needs to be correctly integrated in models to be useful. Having an MBSE solution enabling to transmit information from requirements to a functional architecture is a challenge, as explained in [25].

The difficulties encountered can be linked to a current research topic that is *early validation* [26, **?**]. The integration and application of different V&V solutions in the overall development and modeling process is also an issue [27]. The same study highlights the issue of expressing and continuously checking system properties through different or evolving models, which can be linked to BT needs in terms of V&V requirements and traceability.

As mentioned in the need **N_3.1.**, an ontology is necessary to enable communication and consistency of the specifications in the models. Van Ruijven explains the needs of ontology in MBSE [12], arguing that a standard ontology for MBSE has yet to be defined and the ones used in practice lack consistency. This is supported by the MBSE roadmap defined by the INCOSE, presenting the need for an MBSE theory and ontology to be considered by the year 2020.

We summarize the challenges as follows:

C_1. Having an integrated model of the system at the early stage of design.

C_2. Making use of SysML elements to specify requirements regarding the system behavior.

C_3. Anticipating and/or managing the emergence phenomenon in the behavior.

C_4. Formalizing, completing and standardizing information in the specifications.

C_5. Integrating and applying V&V solutions in the overall development and modeling process.

C_6. Expressing and continuously checking system properties through different or evolving models.

C_7. Supporting the MBSE approach by a formal ontology

The way the needs previously expressed are linked to the challenges identified is given in the Figure 1.3.

	C 1. Create an Integrated Model	**C 2.** Use SysML To represent Specified Behaviour	**C 3.** Handle Emergence In behaviour	**C 4.** Specification Standardization	**C 5.** Enable model Verification	**C 6.** Enable Continous Validation Of the model	**C 7.** Support the MBSE Approach
N 1.1. Specify dynamic aspects between use cases.		X					
N 1.2. Correlate the information expressed in different scenarios and use cases.	X	X		X			
N 1.3. Formalize the circumstances enabling the execution of each use case.			X	X			
N 2.1. Define V&V requirements for the integrated system behavior				X	X		
N 2.2. Provide V&V solutions of the behavior based on SySML models				X	X	X	
N 2.3. Ensure traceability and continuity of specifications and validation results	X			X	X		
N 2.4. Fill the gap between tools, models and development steps		X					X
N 3.1. Define an ontology that enables us to express system concepts in all models		X					X
N 3.2. Develop a model verification solution according to the ontology and modeling Method provided					X		X
N 3.3. Implement the solution in a method Supporting the system development	X	X	X	X	X	X	X

Figure 1.3: Links between BT needs and identified research challenges

1.4 Target

All the challenges presented in the previous section will not be solved in this thesis, in part because there are independent teams in BT that are responsible for some of them. The rest is due to the limitations on the scope of the thesis to keep the issues manageable and relevant to the company. We provide the scope in which solutions are to be developed as well as limitations and constraints to take into account.

We consider issues and limitations from different points of view, resulting in the definition of seven domains that we will discuss in this thesis:

- Ontology.
- Semantic.
- Integration.
- Specification.
- Verification & Validation.
- Traceability.
- Modeling.

"Method" and "MBSE" are not mentioned in these domains as it overlaps with the method that we are currently trying to define, as part of an MBSE solution to the issues identified. As such, the domains are used to characterizes the different solutions, goals and benefits of the developed method.

The solutions to be developed will support the BT SysMM. BT SysMM is deployed and used by the functional architecture department, responsible for functional specifications of the whole train. Engineers belonging to this team are working closely with requirements engineers. The functional architecture department provides specifications to the teams or external suppliers working on the sub-systems of the train. They also work with teams working on verification and validation of the whole train system, taking part in the process. This study is hence centered on the role and work of the functional architecture department. Specifications and models will be provided to the validation department.

According to the Figure 1.1 and according to the need **N_2.4.**, there is a need to interface V&V tools and enable the exchange of models. However, the SIL and HIL solutions are outside the scope of this study. Furthermore, the methods and models used in SIL are still in development, while the use of SysML models in MIL has been confirmed. The first step in bridging the gap between a tool and V&V steps of the validation process is to define and formalize information regarding the behavior and the criteria used to check it. It is why the solutions developed focus on the SysML models, specifications and requirements to be transmitted and not the technical aspects, which are already studied by the validation team. The need **N_2.4.** will be covered by specifying the specifications and validation requirements to be transmitted.

Each functional architect is responsible for the specification, design, verification and validation of the system regarding his own scope. They are responsible for the global functional design along the development process. This is why the focus is put here on the definition of the upper layer of the system representation, which is the more abstract way to represent a train. The information manipulated can however be defined with various *levels of abstraction*.

A level of abstraction is a *"view of an object at a specific level of detail"* [1]. We consider four levels of abstraction: train, system, sub-system, component. Expressing information at system level means the information will not mention sub-systems or components. For example, the energy supply of a train characterizes the train, but it could mention the status of the batteries of the train, which are components. At train level, it would mention the train internal energy supply, without specifying any element at a lower level of abstraction, such as batteries.

Developing sub-systems is often the responsibility of external suppliers. As functional architects are responsible for specifying the behavior, not the physical architecture, they should manipulate information at train or system level, not lower. Sub-systems are defined as abstract entities and treated as black boxes. The solutions developed will focus on the system level or higher regarding the object of study and the information manipulated.

BT SysMM is supported by a SysML profile. Concepts have already been defined regarding the semantics associated to the modeling elements. Rather than developing a full ontology as specified by the need **N_3.1.**, only concepts needed to represent a system and its behavior will be defined.

We summarize the scope of the solutions as follows:

- Methods, solutions and models are developed for the functional architecture department.
- Specifications and models will be provided to the validation department.
- V&V solutions are developed for MIL only.
- Studied information will be at the system level of abstraction or higher.
- Solutions will be compatible will SysML models and BT SysMM method.
- New concepts will directly characterize the system and its behavior.

These limitations apply to both the needs and the challenges previously defined.

1.5 Contributions

This thesis is organized around three main contributions, answering the needs of BT and providing answers to the related challenges.

The first contribution relates to the definition of the state and mode concepts, with the goal to model a system and it behavior. We define the concept of state to describe a system and the circumstances it is in at a given time. The concept of mode is defined in order to specify the behavior by linking actions to circumstances enabling to perform them. All of this is addressed in chapter 4.

The second contribution presents a solution to check SysML models for errors. A method is developed to create *rules*. These rules are short scripts that enable to check all instances of a SysML modeling element in a project. Any property, attribute and relationship can then be identified by a rule and compared to one or several criteria. Using such rules to constrain the way SysML elements are used, it is possible to automatically check models created by engineers for errors with respect to a modeling method. Based on the SysML profile of the BT SysMM, rules were created to check the semantics behind models, ensuring that engineers create the same models the same way, enabling their communication and reuse. This is presented in the chapter 5.

The third contribution uses the concepts of state and mode provided in chapter 4 to describe the system, correlate the information, precondition the use cases, perform verification activities on the behavior and generate a structure for the model of the system behavior. This is detailed in the chapter 6.

1.6 Organization of the thesis

This thesis is organized in two parts. The first part is an analysis of the issue. The chapter 2 presents the context of this work and the solutions used by BT, as well as their limitations, to conclude on a list of needs to be satisfied by the solution developed. The chapter 3 presents and comments the general system theory and current issues in systems engineering, to position this work regarding the state of the art and the challenges it addresses.

The second part presents the contributions. The chapter 4 proposes an innovative definition of the concepts of *states* and *modes*, used to characterize and model a system. The chapter 5 presents a method to verify SysML models to enforce semantics specific to BT company. The chapter 6 presents a behavior verification method which leverages the concepts of states and modes to integrate and verify train system behavior at a high level of abstraction. The chapter 7 summarizes the contributions and their deployment in the company, and discusses research and industrial opportunities to explore.

The annexes contain references and examples of some of the solutions deployed in the company.

Part I

Problem analysis

Chapter 2

Context

This thesis has been conducted inside a team of functional engineers, in collaboration with experts from different sites of the company (France, Germany, England, Canada) in charge of developing the BT system modeling method for the train functional specifications. This modeling method is applied on full scale projects but is still being worked on, as it requires improvements.

This chapter presents the Bombardier Transport System Modeling Method (BT SysMM) the validation process and solutions in BT regarding the train behavior.

2.1 Bombardier Transport Modeling Method

We presents in this section the BT SysMM. We first provide an overview of the method and the context in which it is deployed. We then present in more details the operability analysis, performed at train level, and the operational, functional and technical analysis performed at system level.

2.1.1 BT SysMM overview

Work Breakdown Structure

BT projects are conducted based on the general *Work Breakdown Structure* (WBS) [28], shown in Figure 2.1. A WBS is a structure that describes the work to do in the project. All nodes of this structure are linked to work packages containing the details of inputs, outputs, work activities and milestones in each work domain.

LEVEL					WORK BREAKDOWN STRUCTURE (WBS)
1	2	3	4	5	2w 3w 4w 5w
1					Milestones
2					Project Wide Management
3					Vehicle Performance, Functions and Architecture
4					Sub systems Development and Integration
5					Vehicle Assemblies & Integration
6					Manufacturing and Methods
7					Train Testing, Debugging and Acceptance
8					Train Production
9					Product Introduction and ILS
10					Operate and Maintain

Figure 2.1: The BT General Breakdown Structure [28]

All projects are different, and so are their WBS. Nevertheless, in order to ensure consistency, compatibility and communication, each WBS is built around the same base, which is the general WBS. The focus is put here on the design method, meaning only a few work packages are considered. They are highlighted in Figure 2.2. Our goal is to study how the behavior is specified through use cases, scenarios and functions.

3				**Vehicle Performance, Functions and Architecture**
3	1			Railway
3	1	1		Operability Context
3	1	2		Environmental Conditions and Durability
3	1	3		Electromagnetic Compatibility and Earthing
3	1	3	1	Electromagnetic Compatability
3	1	3	2	Earthing
3	1	4		Railway Integration
3	2			Vehicle Performance
3	3			Functional Design
3	4			Architecture Layout Design
3	5			Digital MockUp Set Up
3	6			Testing Authorisation and Homologation Plan
3	7			Manufacturing Concept

Figure 2.2: Elements of interest in the general WBS [28]

Global modeling approach

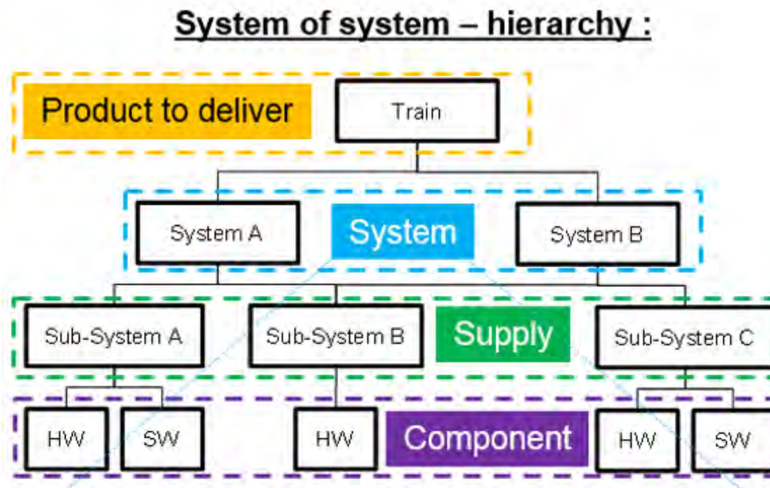

Figure 2.3: System Hierarchy Level (SHL) for the *train* design [29]

Each project in BT is developed starting from the highest level of abstraction to the lowest. The current working level is called the System Hierarchy Level (SHL). The *train* is first considered as a whole, described as a black box, then it is divided into systems corresponding to *consists*. A consist is an independent element of a train, also called vehicle. The *train* can be constituted of one or several *consists*, each divided into subsystems, themselves divided into components.

The Figure 2.3 describes the hierarchy of the train system elements. This hierarchy allows to decompose the development process of the train into more manageable steps, lowering the level of abstraction of the information considered and restricting the scope of study as we descend into its levels. The element considered at each step of the process, for example a *consist* or one of the subsystems, is called the System Of Interest. This SOI is the target of the modeling effort and define the scope of the element to consider.

We are interested here in the development of the system behavior, from the definition of operational scenarios to the definition of a functional architecture. This is done first in the operability analysis (OB) performed at train level, and then at the system (consist) level through three consecutive steps: the operational analysis (OA), the function analysis (FA) and the technical analysis (TA).

BT SysMM follows the ISO 15288 standard [6], where a technical process is defined around successive steps. Specifying the system behavior and functional architecture would be done at the following steps:

- Stakeholder needs & requirements definition process.
- System requirements process.

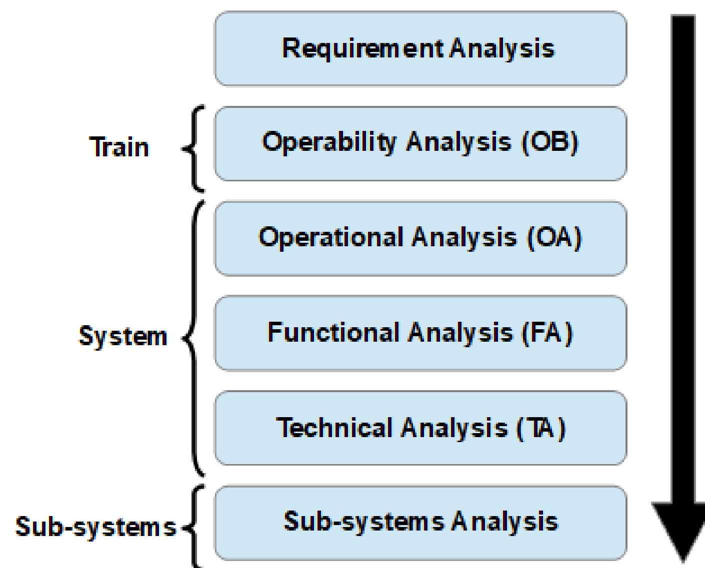

Figure 2.4: Train development process

Requirements analysis in BT corresponds to the first step. The train operability analysis is the second steps, used as a junction between requirements analysis and consist analysis, which comes next. The corresponding development process is shown in Figure 2.4.

ID	Category	SOI
0R	Operability	Train
1P	Performance	System
2F	Functions	System
3A	Architecture	System
4S	Sub-systems	Sub-systems

Figure 2.5: Requirement Breakdown Structure (RBS)

The train development process is driven by requirements. All development steps receive requirements as inputs and generate requirements as outputs. Requirements are classified according to categories and the SOI they qualify. This classification is called the Requirement Breakdown Structure (RBS), shown in Figure 2.5.

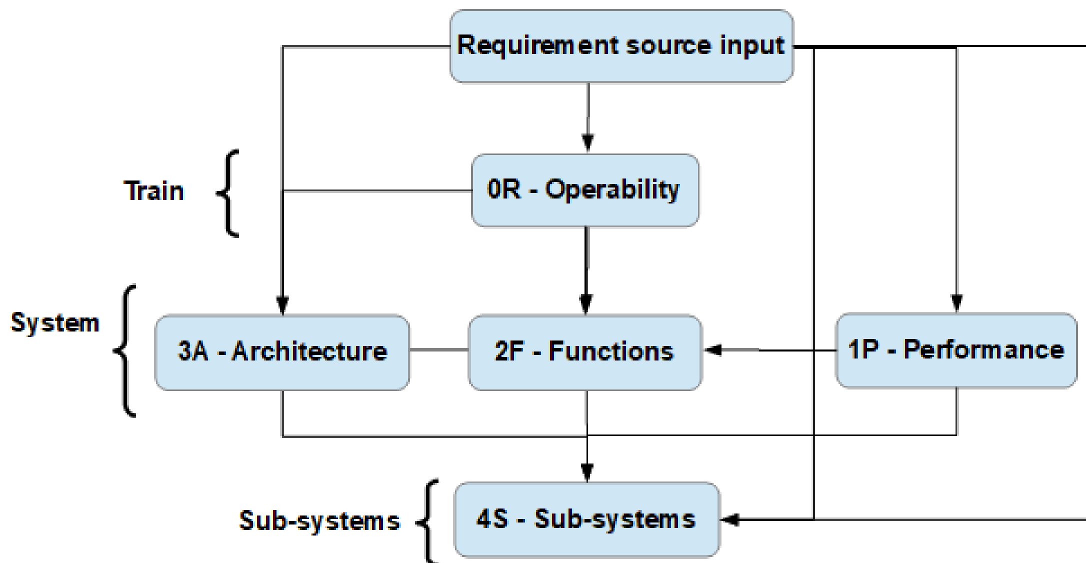

Figure 2.6: Requirement driven design process [29]

The traceability and refinement of requirements is shown in Figure 2.6. The terms 0R, 1P, 2F, 3A do not correspond to development steps but to the classification of the requirements associated to or specified in it. For example, functional requirements will be expressed or traced in OA, FA and TA while being classified in 2F. The arrows correspond to traceability links between requirements.

Traceability is keeping track of information as it is refined or used in different modeling elements or documents. Refined requirements are traced to the initial ones. Architecture and functional elements can be traced to performance requirements linked to them, supposing they have some.

The traceability should be enforced between requirements, but tends to be lost through the modeling activities, as the requirements are traced to models. New requirements are obtained after several steps of modeling, making it hard to keep track of the initial requirements. We identify the following issue:

I_1. Requirements traceability is lost between modeling steps.

The Operability analysis (OB) step concerns the analysis of the utilization of the *train*, according to the requirements. Scenarios regarding the operations performed with the train are specified, then detailed as sequences of train activities which will correspond to use cases later in the process.

The functions are defined on the *consist* level, then linked to the subsystems in an architecture. The subsystems are then developed. Definition of the physical architecture and development of the subsystems are separate activities that are not covered in this study. At each level, a three-step development process is applied. Those three steps are presented in the next section.

The three-step modeling approach: OA, FA, TA

The system development in BT is done around a three-step approach at different hierarchical levels (Figure 2.4): Operational analysis (OA), Functional Analysis (FA) and Technical Analysis (TA). In practice, these steps are not assigned to a specific team.

The definition of functions, however, is performed by a team of functional architects and is supported by existing modeling methods. This analysis is conducted at the system (consist) level. The modeling method associated can be decomposed into the Operation Analysis (OA) and the Functional Analysis (FA). The Technical Analysis (TA) is in majority performed by another team dealing with the system architecture. While the same modeling method can in theory be applied to subsystems or even components, those are developed by either dedicated teams or external providers which often possess their own methods or models. They work more independently and are not considered in the scope of this study.

Regarding the specifications of functions at system level, the three steps can be defined as such:

- The OA aims to describing the behavior of the SOI from an external point of view. Its interactions with its environment are modeled, as well as the tasks it performs for the users or linked elements, which translate as use cases or functions.
- The FA aims to match uses cases and interactions to global functions, in order to build a functional architecture. Exchanges between those global functions are modeled, defining the internal mechanisms of the system.
- The TA corresponds for the main part to others activities such as defining the logical and/or physical architecture, as well as studying the variability in the available components (*eg.* different brands of batteries). It relates to the work of the functional architects when allocating functions to structural elements.

Development process and validation activities

The validation activities are generally performed along the modeling process. BT wishes to improve those activities and group them as a global tooled method. The Figure 1.1 presented in the introduction shows the validation process currently used by BT, as well as the solutions to be developed or integrated.

2.1.2 Operability

The goal of operability is to specify the behavior of the *train* at its own level of granularity. We define granularity as the level of details of the information specified, following the same decomposition as the SHL: train, consist, subsystem, components. Informations should be set at a suitable level, generally the one of the SOI. In current practice in BT, the whole train can be set as a SOI, and be used to attach low level informations. This is something we try to avoid:

I_2. The specified information is too detailed for high-level SOIs.

Operability analysis (OB) is the most crucial step in our study, as it is where the expected system and its behavior can and should be specified through the integration of the expectations into a coherent whole. The OB is a step between requirements analysis and operational analysis (OA), as shown in Figure 2.4. Following this process, the requirements from 0R are divided into three categories: architecture (3A), functions (2F) and performances (1P).

The focus given on the functional aspect of the operability analysis, which enables the transition from the functional requirements captured, analyzed and formatted in the requirements analysis and classified as 0R, to those specified in the functional analysis, refined and classified as 2F.

Concept of Operability

Operability was a concept first developed to express the analysis done at train level when performing the requirements analysis [30]. The objectives of the operability analysis can be summarized as follows:

- Analyze and complete 0R functional requirements to derive them as 2F requirements.
- Define all planned utilization of the system through scenarios and use cases.
- Describe the train situation and capabilities through its life-cycle.
- Integrate use cases and scenarios into a global behavior.

Operability is initially a requirement driven process, with requirements as inputs and outputs. It was originally performed on DOORS [31], the tool used to manage the requirements.

The operability concept in BT is inspired by the ANSI/AIAA G-043 A-2012 standard [32]. According to this standard, there is a need to develop an Operational Concept Document (OCD) that, among others things:

- Provides a clear vision of the intended uses and the resulting benefits of the system.
- Provides the basis for system validation.
- Describes how the system will be used.

Method

There is no current official method in BT for operability analysis, as it is a work in progress. Several attempted versions of a modeling method have been made [33, 34, 35], and constitute iterations toward a working method. The main issue is that there is a lack of system concepts, preventing a formalization of the models and of the information they should contain. A fourth method is currently under development.

I_3. There is no modeling method for the specification of the expected behavior.

I_4. There is a lack of system concepts to formalize and model the behavior.

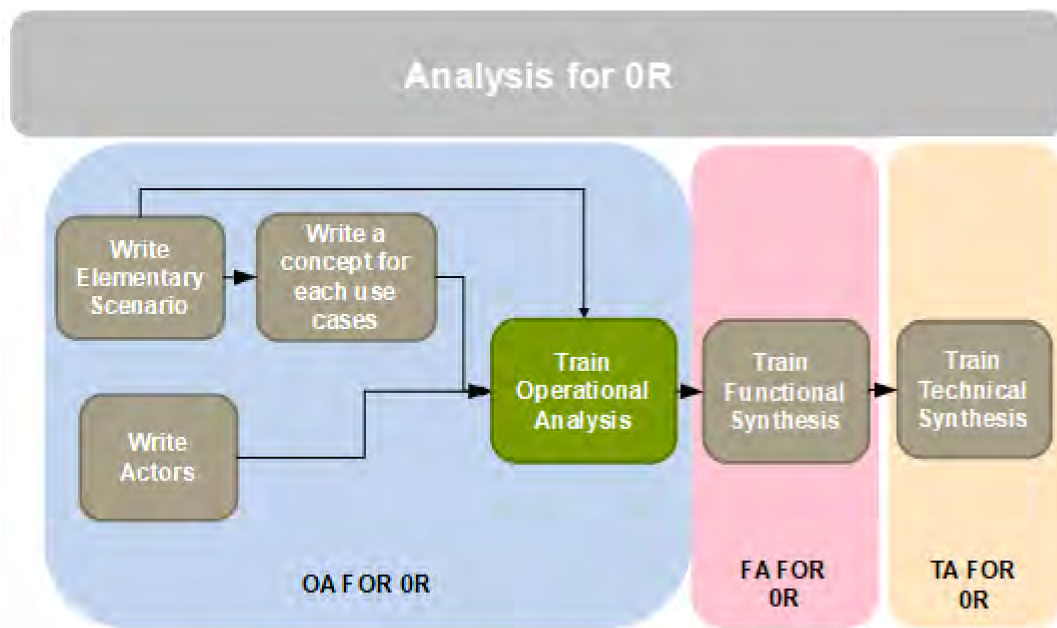

Figure 2.7: Global operability analysis workflow [33]

As a modeling method for operability is developed, it has to be considered as part of BT SysMM. This is why the full operability analysis is divided into the OA, FA and TA steps, as those three steps are applied at each level of development. The operability analysis corresponds to the development step at the train level, meaning it has the train as a SOI. The Figure 2.7 is an early proposal workflow for operability analysis. The gray boxes correspond to steps performed in DOORS.

The focus is put on the specification of the system behavior and the modeling activities. As such, we are not interested in the TA. The actual specification and modeling effort regarding functions and behavior in BT is done in the OB, OA and FA.

Inputs and outputs of the operability analysis, as it is the case for most development steps in BT SysMM, are essentially requirements. There is an intent for outputs to include models. It is then necessary for models to all be built the same way.

I_5. The models created must be consistent with the modeling method.

Focusing only on the modeling part of operability, a more detailed workflow can be defined, as illustrated in Figure 2.8 and Figure 2.10, taken from the most recent method [35].

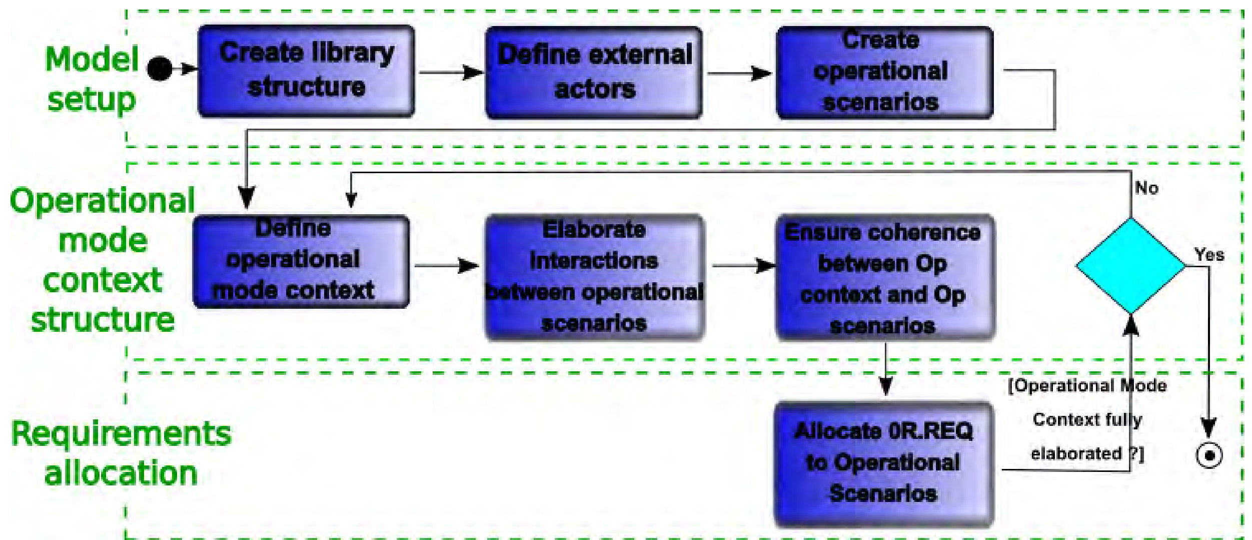

Figure 2.8: Operability analysis workflow (part 1) [35]

In Figure 2.8, we see that after actors and scenarios have been defined, the scenarios are linked together through *operational modes* (which are explained in the next section), in an attempt to have an integrated behavior resulting from the possible actors interactions with the system. Each scenario corresponds to a complex operation, meaning an operation that cannot be easily detailed as basic interactions between actors and the train. All of this is done from the user(s) point of view, based on 0R requirements.

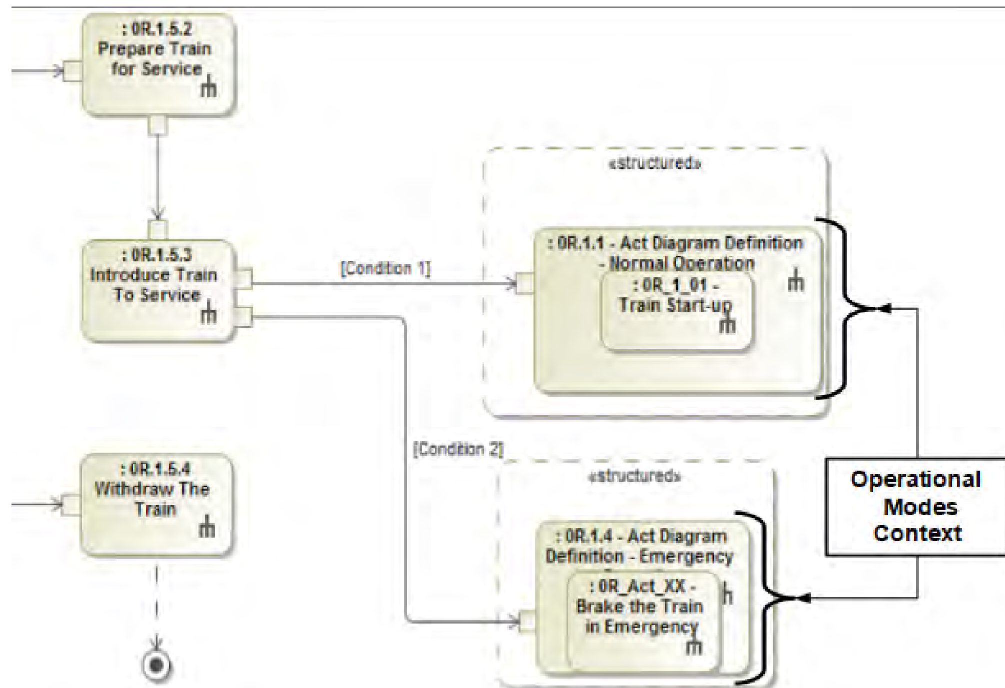

Figure 2.9: Example of scenarios integrated through operational mode contexts[35]

An example of scenarios allocated to operational mode contexts is given in Figure 2.9. The modeling elements used are activities and activity diagram. While a train scenario/operation can correspond to a SysML activity, the notion of operational mode cannot, as it is not an action performed but rather a classification object or a description of the context/situation of the train. Operational modes and scenarios are not supposed to be executed sequentially, they are part of a dynamic behavior. There is no proper semantic in the way the models are defined here, as they do not respect SysML semantics, with activities used the same way as blocks. This originates from the fact that requirements engineers had the habit to draw informal scenario to be used in documents. The information has to be shared on the modeling tool but the method enabling it has yet to be developed. We note the following issue:

I_6. Concepts of modes are not correctly expressed in SysML models

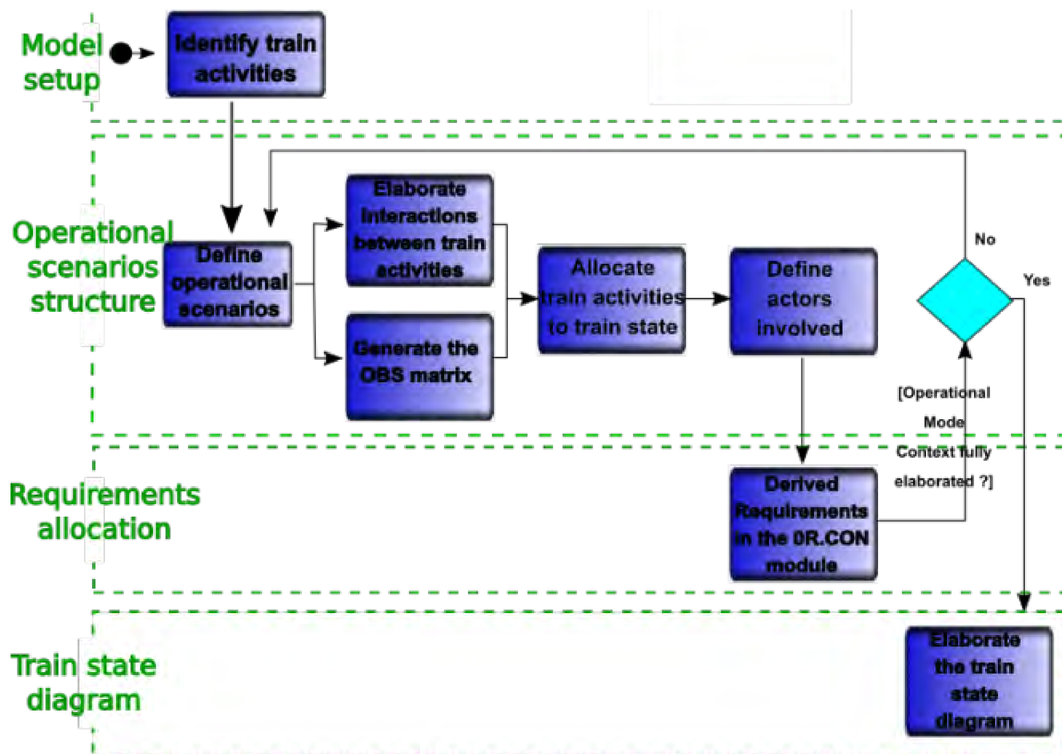

Figure 2.10: Operability analysis workflow (part 2)[35]

In Figure 2.10, we see that train activities are defined. They will be used to detail the scenarios/operations defined earlier. Those train activities correspond to use cases that can be detailed as basic interactions between the actors and the system, something done later in the process at consist level. Train activities are supposed to be associated to *train states*. In this context, those "states" express the conditions under which activities are (can be) executed. The train states are to be integrated in one SysML state diagram specifying the evolution of the train and its behavior. However, there is an issue as there is no definition or method explaining what a state is and how it should be defined. Engineers know they need an integrated behavior of the train, and have made several attempts to define state diagrams describing either the evolution of the train states or its behavior. There is a need to make a distinction between the evolution of system and the evolution of its behavior. This is expressed in the following issues:

I_7. There is no definition or method explaining what a state is and how it should be defined.

I_8. There is no clear distinction between the evolution of system and the evolution of its behavior.

The Figure 2.10 includes a step where activities are supposed to be associated to states. The goal is to specify the condition under which activities can be realized.

However, there is a difference between the conditions for a given scenario and the conditions under which each activity can be performed. The engineer is left to wonder whether a given state qualifies a scenario (under which activities are grouped) or the activities themselves, that may be performed under a larger set of conditions. As activities are associated to scenarios and may be present in more than one of them, states defined around activities inside a scenario qualify the scenario and not the activities. Consequently, it specifies particular uses of the train but not all of its potential behavior, preventing a full specification and integration. The method expressing this workflow was hence not successfully applied, as it failed to answer the issues regarding the specification and integration of both the system states and its behavior. We note the following issue:

I_9. The way the states are used and associated to SysML elements is not clear.

Integration

We present here the means used by BT to express integration. The current practice is to provide an operational life-cycle indicating the sequence of use cases and operations a system is supposed to perform in the context of a normal execution.

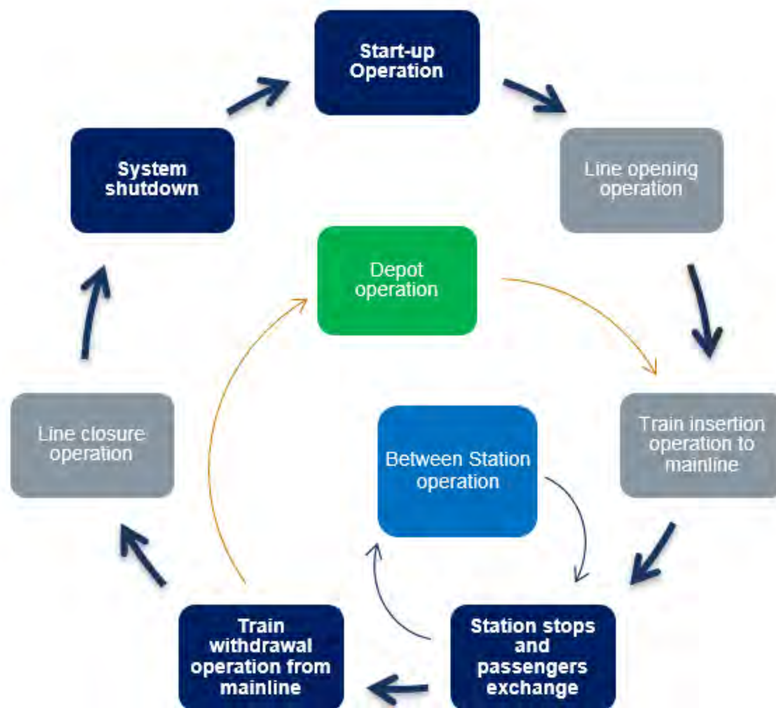

Figure 2.11: Day In The Life-cycle Of The Train (DITLOTT) [30]

The environment, actors, interfaces, scenarios and use cases are defined in the context of a train daily life-cycle once delivered and in use. This lead to the definition of the Day In the Life-cycle Of The Train, or DITLOTT, shown in Figure 2.11.

Scenarios are classified among the following categories [30], and constitute the *operational modes* presented earlier:

- Normal Operation.
- Restricted Operation.
- Degraded Operation.
- Emergency Operation.
- Maintenance Operation.

Those categories can be divided into several other levels of classification. It forms what is called the Operability Breakdown Structure (OBS). The OBS was supposed to be used to classify use cases but failed to do so in practice. While the OBS is a good classification for scenarios, it is not a good one for the use cases called in them. Many use cases are performed in more than one scenario and across the different categories. Use cases are in fact classified in the Functional Breakdown Structure (FBS) as an input for the first step of the functional analysis that comes after the Operability analysis.

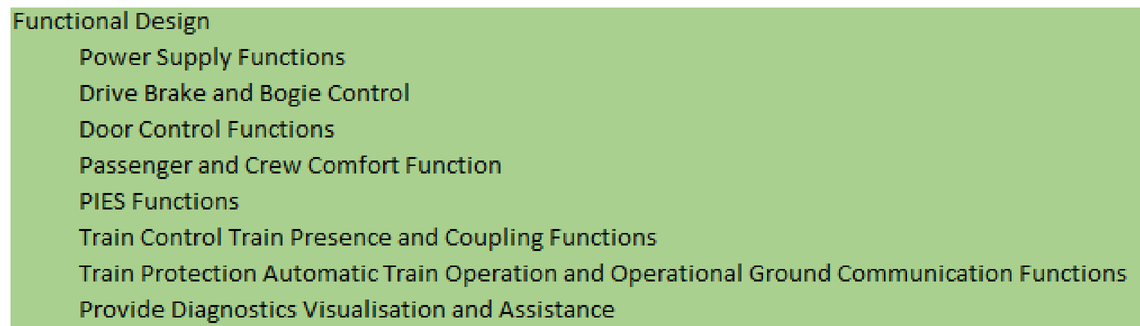

Figure 2.12: Functional domains defined in the first level of the FBS [28]

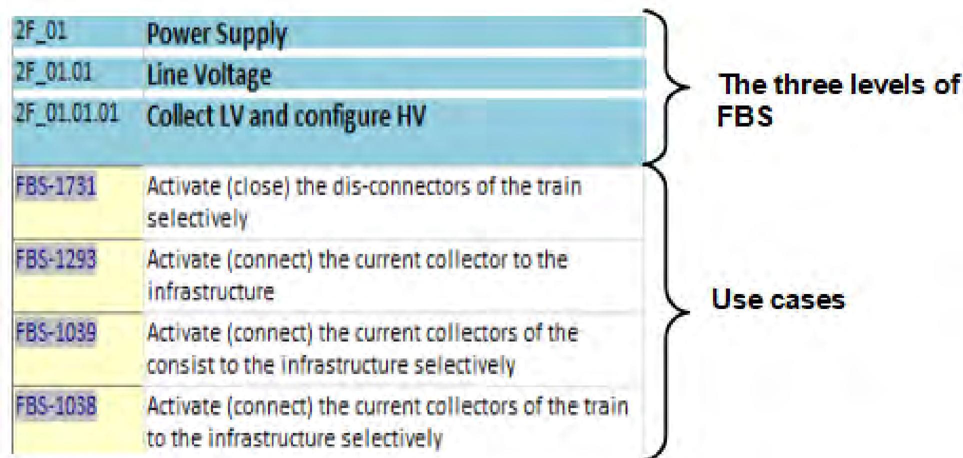

Figure 2.13: The three levels of the FBS [28]

The FBS is separated in 8 domains, as shown in Figure 2.12. Each domain is structured on three levels (Figure 2.13). The different levels of the FBS can change from one project to another, and are defined or adjusted as part of the operability analysis. The use cases are then allocated to the third and last level of the FBS.

Operational modes, which are modes of utilization expressed from the user(s) point of view, tend to be mixed with *operation modes*, which characterize the train capabilities linked to a situation from its own perspective. Operation modes are shown in Figure 7.1 in the annexes. Operation modes have been defined in a previous method and are referred to in BT documents as a standard. They are often reused or adapted in the specifications, but their definition, adaptation or utilization is not formalized and do not appear in the workflow. The Figure 7.1 indicates that operation modes are supposed to be organized in separate hierarchical levels, but often characterize different pieces of information that are more or less correlated. For example, the energy supply modes (such as “battery power supply”) are contained under the higher level mode “In service mode”. However, those modes of energy supplied could be considered for other modes, as it is supplied in energy whether it is in service or not. Operation modes are in fact built by conditioning different pieces of information put in correlation, meaning conditions are set on the values of different pieces of information, for example the current energy supply while being in service.

The Figure 7.1 illustrates pieces of information used in actual projects. The ”modes” depicted in it are part of a standard in development and still possess flaws. The different types of information they express ought to be separately defined: modes characterizing the energy supply are grouped under one mode called ”service”, the reason being that they are only considered when the train enters service according to the scenarios, even though the energy supply modes can change in other circumstances.

Operation modes, such as "normal" or "emergency" can also be confused with the corresponding operational modes. The distinction between these two kinds of modes is not explicit in current practice and is made here as part of the analysis. The issues around modes can be summarized as follows:

I_10. There is a confusion on what a mode is and how it is used. **I_11.** The hierarchy between modes is not clearly defined. **I_12.** The way the different types of information are characterized and correlated is not clear.

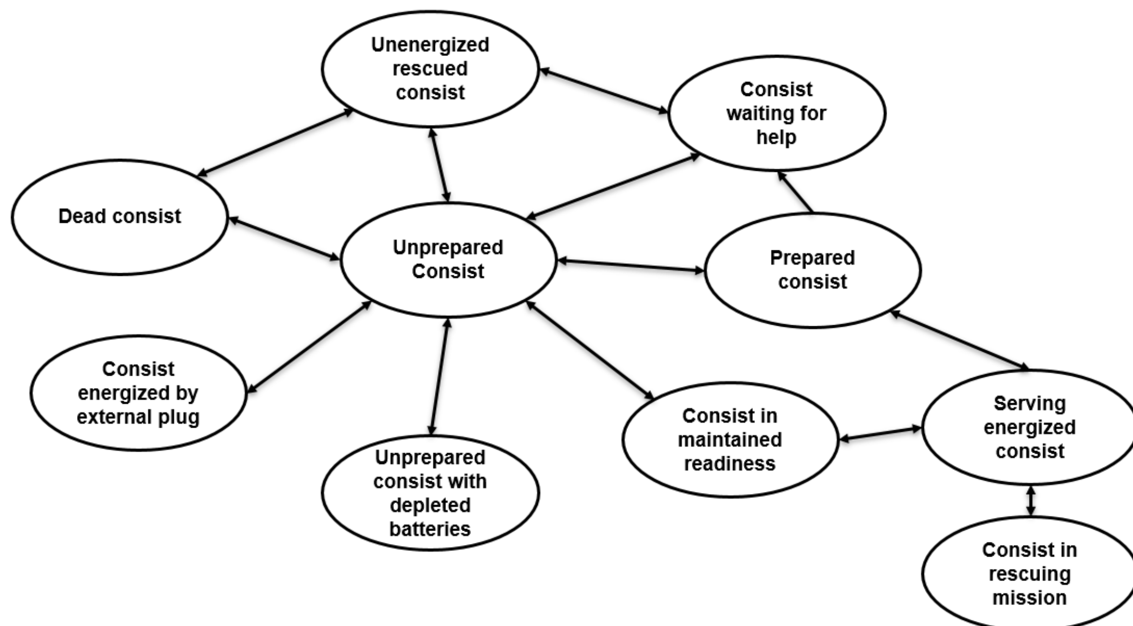

Figure 2.14: Example of a state diagram of the operability modes

One of the goals of operability is to deliver a diagram summarizing the evolution of the train behavior. It sometimes represents the evolution of its driving modes, but it more generally exposes the evolution of its capability to perform operations, also called operability. The term carries the same meaning concerning the development step or this diagram: it is the way the system can be operated.

An example of the operability diagram is shown in Figure 2.14. The SOI in this diagram is a consist and not a train. The reason is that the train operability corresponds to the operability of the *master* consist, meaning the one controlling the others, if there are any. As the train is an abstract entity, engineers tend to study the consists it is composed of. This leads to an ambiguity as to what the SOI is under study at a given step of development.

Each train has a specific operability diagram. The elements of these diagrams are either called states or modes. They characterize the train and relate to the operability, hence to the evolution of the train readiness and capabilities. Operation modes or operability modes do not always characterize use cases or functions directly. They are often used to describe the situation in the scenarios. The realization of a use case may be possible in a broader scope than the scenario where it is called.

Train states describe the situation the train is in. It directly preconditions train activities. Train states are also defined in documentation to express a different kind of information, such as the train situation in its environment or its energy supply [29]. The information qualified by the states are not always separated. Train states tend to characterize different types of information at the same time, without concern about consistency. Even though each type of information has a finite set of values, they are not always individually or formally defined: in a same situation, the energy supply of the train can be said to be internal or being provided by its batteries. This difference in definition for the same information is ambiguous, given values of different types of information may not always be compatible. For example, a train cannot be moving with its parking brake engaged, proving that two different kind of information can constrain the values of one another. As such, there is an issue:

I_13. Constraints between possible simultaneous states are not formally defined.

States are often mixed with operation modes or operability modes. Modes in BT, compared to states, are understood to be linked to the system functions, meaning functions are enabled or not depending on the modes currently activated. Those relations are not clearly defined. Coming back to Figure 7.1 and the step where activities are supposed to be allocated to train states, we see that train states fulfill the same role as modes, demonstrating that those two concepts are mixed. Modes are used to both classify scenarios and use cases and to represent the system behavior. There is an ambiguity in their meaning and application, just as there is for the states. We note:

I_14. The differences and relationship between states and modes is not clear.

We have seen that scenarios and use cases are classified separately. This is a key point and issue to address in operability: switching between the user's point of view previously adopted in the requirements analysis to the one of the system adopted in the functional analysis. Scenarios and use cases are classified according to utilization, situations and skill domains, while functions of the train are to be organized in system elements (logical and/or physical).

2.1.3 System development at consist level

BT SysMM currently specifies how to model the system (a consist), though it could in theory be also applied to subsystems. It applies a three-step development process for a given hierarchical level.

Each step of the process follows the same pattern: first defining the context with all interacting elements in a Block Definition Diagram (BDD), then specifying the exchanges between these elements in an Internal Block Diagram (IBD), and finally defining the other elements specific to these steps (use cases or functional blocks for example).

It should be noted that although the method is presented sequentially, the modeling work is in fact done iteratively. It is for example doubtful that all the information exchange will be specified in the first version of the IBDs. They will be completed after modeling sequence diagrams or other elements.

Operational Analysis

The goal of OA step is to express the system behavior from an external point of view, as well as what it is expected to perform. The elements of the BT SysML profile for the OA are the operational block, the operation context and some variation of the requirements and relations that will not be specified here.

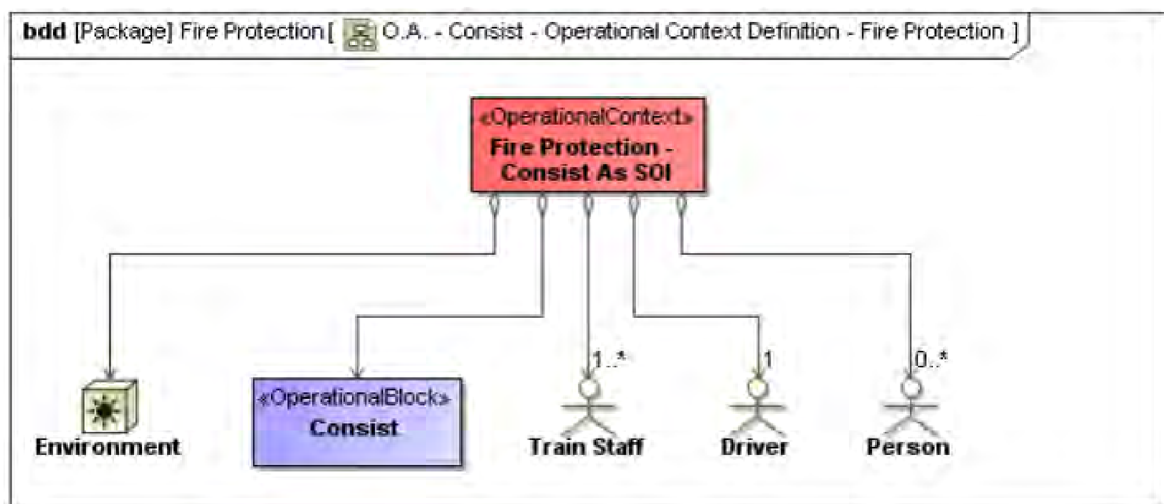

Figure 2.15: The operational context and its related elements [36]

The operational block (named *consist* as it is the SOI) is defined for all elements in the OA. For each scope, an operational context is defined with all actors and environment elements interacting with the SOI.

Actors and environment elements are parts of a library that can be completed or adjusted following the requirements and the specifications in operability analysis. An example of an operational context definition is shown in Figure 2.15.

Once the operational context has been established, an IBD with all the interacting elements in the operational context is modeled.

The elements specific to the operational analysis and used as outputs of this step are then defined. Use cases are created, linked to sequence diagrams and activities. BT SysMM suggests creating the sequence diagrams before the use cases. Each sequence diagram specifies the interactions between the SOI and the elements of the context relating to a use case.

Another issue in the use cases definition is the expression of the preconditions, be it for the use case as a whole or for the operations described in the sequence diagrams. This information is taken from the requirements, and as such is not formal, nor complete, shared or known by all engineers working on different use cases and requirements. This corresponds to the need **N_1.3**. Part of the issue would be solved if we had a solution in operability analysis, where the same difficulty was encountered. However, the information would not be exactly the same, as the SOI has changed from the train to a consist. The states and modes expressed here would characterize a consist, not the train. We note one issue:

I_15. States and modes do not follow the evolution of the SOI.

In order to specify a given behavior, it is necessary to define the preconditions of each operation and action taken by or through the train. Preconditions are expressed informally, their specifications and meaning being based on the engineers knowledge and competence. The information is for the most part contained in the engineers minds, not in the model. Another concern is that the preconditions are not easily accessed, traced or manipulated. A solution to this issue has been to define spreadsheet files to put together the information in front a list of use cases, but ultimately failed as it had to be filled manually and the information was too detailed, redundant or unclear.

Once defined, the use cases are put in use case diagrams for more visibility. The use cases are linked to actors or relevant context elements. Traceability can be ensured at this step by allocating requirements to the use cases. Use cases would normally be linked to requirements in operability, where they were expressed as train activities. The traceability between train activities and use cases is not ensured. Ideally, use cases would be traced to the train activities elements, themselves linked to requirements. Requirements can be imported from DOORS as SysML elements.

In order to define the tasks (functions) and operational behaviors expected in each use cases, activities should be defined for all of them, according to the method. An activity can be used in several use cases. In theory, such elements are created in a separate folder and allocated to the relevant elements through the diagrams.

In practice, creating activity diagrams require a large modeling effort, and there are no library of activity elements as they are to be defined along the use cases specification. This restriction makes it difficult for engineers to cooperate and use the same activity elements when needed. Operations in sequence diagrams would correspond to those activities, hence leading to some redundancy. Those activities elements could eventually be allocated to function blocks (defined hereafter) to support their definition, but they are not defined with such an intent and functional blocks have no formal definition method for them.

The reason activities should be defined is to be shared among use cases, enabling to specify a unique group of functions and the circumstances under which they are performed. Sequence diagrams have a beginning and an end, they are not adapted to express dynamic aspects, as loop and simultaneous actions are not formalized. Alternative choice of actions can be hard to model. Integrating activities from all use cases, on the other hand, would enable to specify the dynamic between them: which one can be executed at the same time, are there any loop, how can use cases be sequenced, etc. This would be facilitated by the integration done in the operability analysis, should it be achieved (we saw it was not the case yet). Instead of specifying separated expectations regarding the system behavior, it would then be possible to integrate them into one, full, consistent model of the expected system behavior. However, there is currently no method nor models achieving this purpose. This issue has been covered in the introduction in the need **N_1.**.

Once all operational blocks, use cases and corresponding sequence diagrams have been defined, the Operational Analysis is finished. The OA can be used as a communication mean with all persons of interest, whether engineers or clients. The activities defined in the OA are not considered as official functions regarding the deliverable. Functions are officially defined in the FA. As a consequence, there is a lack of traceability between the behavior specified in OA and the one developed in FA. This illustrates the need **N_2.3.**

Functional Analysis

The functional analysis aims to define the main functions the system considered must fulfill. The elements of the BT SysML profile for the FA are the *functional context* (FC) and the *functional block* (FB). Each FC correspond to a use case, and regroup functional blocks. The FBs are global functions elements, meaning they each contain different functions that are not detailed. They are used to classify and organize functions, so that they may be allocated to sub-systems. They are grouped in FC. A FB can be used in several FC.

First, the FC is defined in a BDD. Functional contexts aggregate or are composed of FBs. FBs are often reused from other projects or from a product family. There is not an official library for them. Engineers can create their own FB and refine them later so that they correspond to preexisting ones in order to be able to allocate them to subsystems later in the process.

Each FC is defined in the third level of the FBS. FBs are defined in the first scope where they are called by a FC. As said before, there is no official library for them, so they are not contained in a separate folder. This can be an issue as they can be called in several scopes of the FBS, and as such cannot be properly contained or classified in it. Each FB is supposed to be defined only once. A FB is contained by the FC where it is defined first. A FC including a FB already defined in another FB will aggregate the existing FB.

The FBs are used to refine the sequence diagram associated to the use case from which the FC has been defined. Each FB corresponds to a line in the sequence diagrams. Exchange between those lines show the inner working of the train. The activities allocated to the use case defined in the OA can be used to build the sequence diagram.

An IBD is then created, modeling communication and exchange between the FBs of a given FC.

This time, the system behavior is partially integrated as all of its functions and behaviors are expressed through a same set of FBs. However, the operations performed by each FB in the different functional contexts and sequence diagrams it is involved with are not detailed. As each FB can be responsible for different operations, the way those operations are integrated (without being detailed further) should be specified.

The FA is then finished as specified in the BT SysMM. Requirements and FBs are expected as outputs. FBs are defined at a same level of abstraction and detail, or are derived until it is the case. They are all defined in a same level, without any hierarchy between them. This means that the function architecture is ensured by the FBS and the FCs it contains. Each FBs will be allocated to one subsystem in the TA.

A dysfunctional analysis is done at this point, as part of the functional design process. Each function gets a context and an analysis of the possible failures and their potential consequences. This work is done by a separate team on a separate tool, so it is not considered as part of BT SysMM and the process described here. This separation is enforced for organizational reasons, but also to ensure an independent analysis of the system by people with specific responsibilities and skills.

Providing that the different, internal operations are properly identified and modeled, the previous issues identified around the specifications of the system behavior remain the same: specifying the states and modes, the preconditions of the operations, correlating the information, etc.

Technical Analysis

In the TA, the functional blocks are associated to subsystems. A logical architecture is then established. IBDs are made to show the exchange of signals between subsystems. Interfaces are defined for each subsystem.

The allocation of FBs is decided based on architectural requirements and after an analysis on the variability of subsystems and components that can be chosen as a solution. This is hence a task performed by another team in charge of the system architecture, and is not in the scope of the present method, where we focus on the system behavior independently of the system logical or physical structure.

2.2 Verification and validation process in BT

The validation process of the system behavior at system level is best summarized by the Figure 1.1, already presented in the introduction.

BT engineers are fully capable of building a train with current methods and models, but such a train may not be a working or satisfying solution for the client, at least not on the first try. Conducting an iteration on the development of a finished product can take months or years and is very costly. To ensure that the right product is developed in optimal conditions, we need to perform some early validation and check the system continuously as it is developed, as opposed to a pure Specify-Design-Build-Test-Fix (SDBTF) approach which is inefficient and limited [37].

This section aims to analyze the current validation solutions of the system along the development process. We will consider the constraints and needs regarding their evolution. We want to capture the needs regarding an early validation solution of the train behavior at system level, using the requirements provided by the functional architects.

This early validation would be performed on integrated executable models at the MIL step.

2.2.1 Organization

The definition, management, planning and realization or the train validation is done through several departments and roles:

- The project management team orders, manages and plans the validation. It is constituted around people from management and engineering departments. For example, a functional architect can be given a project manager role.
- The validation requirements are provided by the engineering departments involved in the train development and analysis, such as the functional architecture department, the application TCMS (Train Control Management System, a subsystem of the train) linked to a site, etc. The Safety and reliability department as well as the authorization department also provide validation requirements, after doing their own analysis.
- The validation department performs the validation in internal or external laboratories.
- Product introduction (PI) takes care of maintenance and management of test trains, and as such should not take part in any early validation activity and won't be considered here.
- Train0, the team managing the bench tests and hence the HIL, depends on the validation department and performance measurement. It can be considered as a laboratory.

The task performed by each department are here given as an indication, the work process being more complex than that. For example, the validation department takes part of the planning and discusses the budget, engineers often take part in the validation realization, etc.

The departments involved in the validation process have been identified. We will now define the stakeholders involved in or benefiting from the solution pursued, whether they depend on these departments or not.

We are considering the teams working on the system and subsystem level regarding specifications, designs development and validation activities. The following roles can be defined:

- System specification suppliers: The people producing the functional design of the train at system level, giving inputs to suppliers.
- Subsystem suppliers: the people developing, verifying and validating the different subsystems and providing resulting system validation specifications.
- Validation teams: people already working on validation solutions.
- Clients and providers: people influencing or benefiting from the solution results.

2.2.2 MIL: Cameo SysML

We present here previous work done in BT to create models for a validation during the MIL phase, based on system specifications by the functional architects. An executable model was created as part of a project, using the tool Magicdraw and its plug-in Cameo.

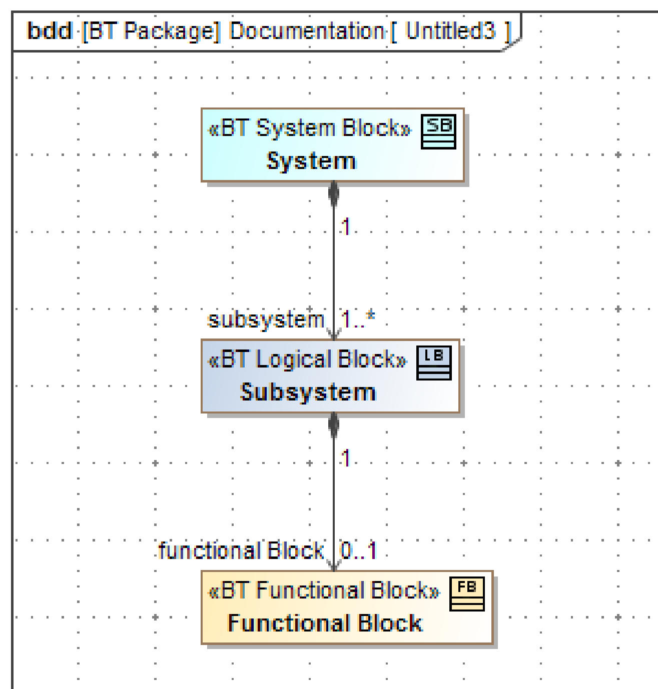

Figure 2.16: System architecture

The integrated system behavior is to be specified using SysML state machines to coordinate and mix individual sequences of operations.

Those sequences of operations can be represented using activity diagrams, called in transition between state elements in the state machines. Each state machine expresses the behavior associated to a block, for example a FB. Those blocks need to be part of a structure and exchange signals through ports. The Figure 2.16 represents the structure obtained at the end of the system functional specification.

A first issue is that the allocation of FBs to subsystems is done at the TA, meaning it is the conclusion of all the work of the functional architects. Due to the impact of an error in the specification, the length of the development process and the amount of work needed to develop a subsystem architecture, there is a need to validate the behavior and the FBs before their allocation to subsystems. This means having no logical (subsystems) nor physical (components) structure to which we could allocate the behavior. The solution used in previous attempts to model the system behavior was to structure the behavior around the functional contexts. This proved impractical, as there were 74 functional contexts in the project considered, all at the same level of abstraction and with many connections between them. Moreover, they each aggregated between 3 and 10 FBs out of a poll of 121, and the same FBs could be contained in different functional contexts, expressing different pieces of behavior. It means that the behavior of each FB was not unique and integrated. Even limiting the scope of study, it still resulted in big models difficult to explore and analyze. The goal was to study the behavior, not the mechanism inducing it which was too complex to consider in details. This raises the following issue:

I_16. There are no structure upon which building and organizing the system behavior before considering a design.

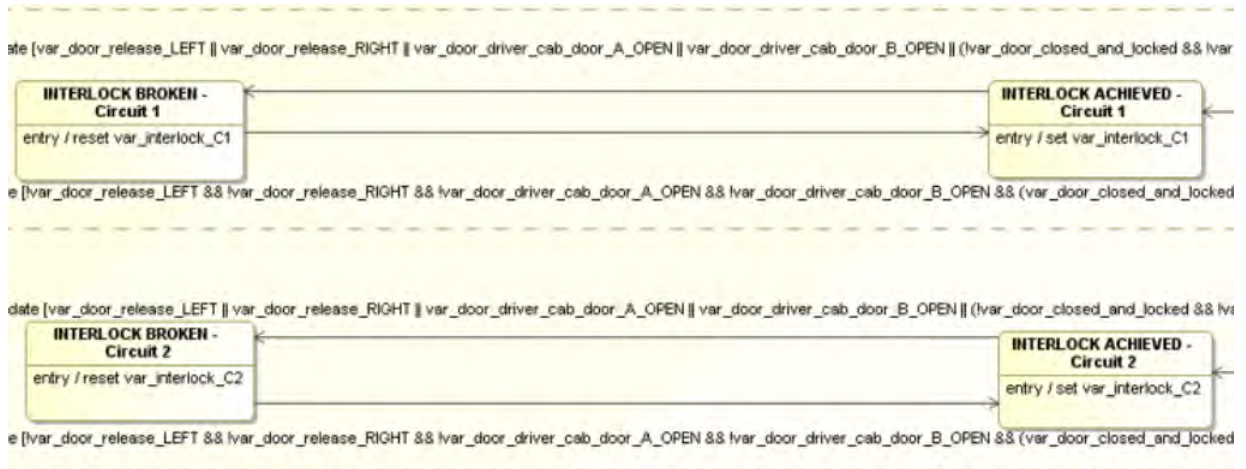

Figure 2.17: Extract of a FB behavior modeled with a state machine

Without an insight on the current actions available, one has to manually explore the state machines of all FBs. In our example, there are 121 FBs with even more state machines as the FBs can be called in different FCs. It is not practical or even possible to know if and how the system will react by looking at all the state machines each time there is an interaction. An example of executable state machine associated to a FB is shown in Figure 2.17. We can see that the guards on the transition are very long, cannot be read and tend to be repeated. State elements can both correspond to the state of different elements, such as a component or a subsystem. Such states express an information that could be used or put in the guard in the transitions. We identify the following issues:

I_17. Possible actions during the execution of the executable models are not available. **I_18.** Guards on transitions are redundant, hard to specify, not readable and not linked to states.

We consider that the role of the SysML state elements in a state machine is to express the internal evolution of the system behavior. This behavior is non-deterministic from an external point of view. There can be different responses from the system to a same input, even if all variables and parameters are the same, due to the evolution of the system behavior through time. The order in which actions are performed can also have an impact. Going back to Figure 2.17, we see that state elements are used to see the consequences and evolution of some information regarding train elements, such as the fact that a door is open or not. This information could be accessed using a variable. Looking at this state machine, which is typically hidden in a large and complex executable model, an engineer has the knowledge of some information regarding system elements but has to read every single guard to know what can be done at this point in time.

It would be useful to integrate similarities in guards into operational situations where a group of actions is enabled. To better apprehend the behavior specified here, one could reduce the redundancy in the specification of the guards. To do that, it would be relevant to express separately the current "state" of the system and the one of each of its elements. By state, we mean here the circumstances the system elements are in relative to the environment and internal variables. Those states would then be used to express the conditions put on the guards. Conditions would be attached to the "state" modeling elements of the SysML state machine, which we do not consider to correspond to the concept of state. All sub-states elements and transitions linked to a given state element would then all share the conditions associated to the state they are contained in. A guard on a transition would then only contain the conditions that are specific to it.

The issue of having the behavior only specified in state machines allocated to FBs is that it is not explicit. Indeed, FBs can be used in several functional contexts for different purposes, and as such do not provide direct information on the train capabilities. Capabilities can be captured by specifying the use cases available or currently performed. The functional decomposition and specification describes the internal working of the system that will implement the expected behavior, expressed from the user(s) point of view. The expected behavior is induced by the functional decomposition, but is not explicitly described by it. The state machines specify *how* the train works but not *what* it can do. As the desired interactions and their effects with the train are known and limited, the goal is to integrate them and constrain the resulting behavior so that only the desired reactions are possible. It is currently possible to specify the mechanism providing an *induced* behavior. What we lack is a way to specify, integrate and constrain such a behavior at a higher level of abstraction, so that we know what should be induced. This relates to the need **N_N1.** and the challenge **C_1.**.

What the user(s) does has been expressed through requirements and scenarios in the operability analysis and the OA, how the train works is specified in FA using sequence diagrams, state machines and activity diagrams. What is lacking is a model of what the system *can* do. It should be possible to evaluate the capabilities available or performed by the system at any given point in time, which is the objective of the integration desired at the end of the operability analysis. The details of such a behavior can be specified using the specifications in OA. Once a model of the system behavior has been created, it should be used to check designs and implementations of the system. To do that, the model should use information specified in FA to determine available capabilities.

2.2.3 SIL: Virtual Bird

Virtual Bird (VB) is a solution for integrating and simulating models and software. It is based on Controlbuild, a tool developed by Dassault. Controlbuild respect the IEC 61131 standard [38], created by PLCOpen, and as such is able to integrate all models and SW that are compliant with it. It is also compatible with several tools which handle the FMI/FMU standard [39]. FMU/FMI is an open standard for models and software exchange and is used by a growing number of tools such as Matlab, OpenModelica, Catia or Magicdraw.

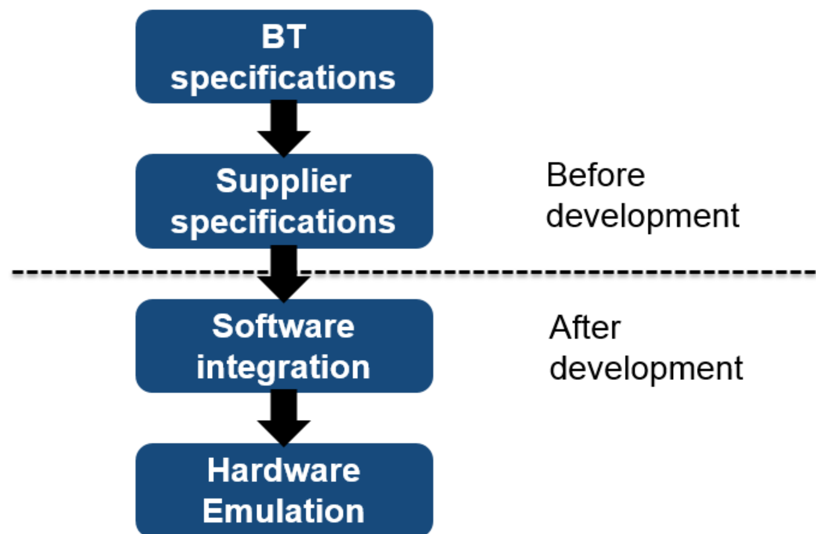

Figure 2.18: The four levels of abstraction for models integrated in VB

VB proposes an environment for every supplier to test their models and software. The team working around VB develops or adapts executable models, performs V&V on them before integrating them with others models and SW provided by the suppliers. Then, a validation regarding the functional specifications is done. The VB team differentiates four levels of abstraction regarding the sub-systems they integrate, as shown in Figure 2.18.

The activities performed using VB are the following:

- Acquisition, adaptation or development of executable models and software.
- Building the integration structure/environment and managing the implemented configurations.
- Individual verification and validation of models adapted or created by the team.
- Functional validation on integrated components through their abstraction levels, depending on availability and needs.
- Other analysis and support depending on the validation needs.

As of now, the VB team performs two kinds of validation: one through tests under the form of scripts that are automatically executed, and one through manual simulations, sending signals from a Human Machine Interface (HMI) according to a set of scenarios taken from the specifications. It is also possible to generate random scenarios and to check properties, but it is not used since it is in low demand and takes a lot of time to analyze scenarios that are often not relevant. Performance analysis could be possible depending on the models integration but it is not in the scope for the moment. Some work was done in VB to test the performance of the brake system, so it can and has been done already. We can keep it in mind for future uses or improvements of the solution.

In order for the VB team to perform other V&V activities than tests or execution, it needs a list of properties or constraints to check the integrated system behavior. There is a need to know the limitations that should be put and respected in the train's system evolution. Considering the need **N_2.1** and the issue **I_8.**, we can see that even though V&V activities in SIL are outside the scope of the solutions to be developed in this thesis, they are impacted by the issues we consider.

Checking currently known constraints and properties is done by generation of random inputs, which is only pertinent if the cases tested are plausible. Having an integrated, constrained model of the system behavior would enable to identify which use case could (should) be performed at a given point in time. This way, random inputs can be limited to what the system is supposed to react to according to the specifications.

The VB activities start once the system level specification is done. As this time, the signals and interfaces are issued from the technical analysis and hence are the same as the ones that will be developed. There is no difference between the levels of abstraction of integrated sub-system from the point of view of the integration environment, as it should be since they are considered as black boxes.

Executable models obtained from BT or supplier specifications are currently used to complete SIL co-simulation but could be used for MIL.

BT specifications that come before the development of sub-systems are not expressed into integrated models, they are functional blocks allocated to sub-systems, and their behavior is described in separated sequence diagrams. Those sequence diagrams are currently used to specify tests for the co-simulation in SIL and HIL. There are no models integrating these behaviors into one with dynamical aspects that could be built into a MIL co-simulation. More than an executable model, which they could develop, the Virtual Bird team is in need of integrated specifications and requirements for validation .

2.2.4 HIL: TRAIN0

The team Train0 (T0) depends on the verification and performance measurement department. It takes care of the integration of HW and SW on test benches and simulation bays. The scope of T0 V&V activities is around sub-systems considered as black boxes. Such activities cover the whole train development process, as they participate in the validation plan initialized during the response to clients' call for offers to develop a train. They discuss with system engineers and the Product Introduction team to see which sub-systems should be integrated, and then they prepare the T0 base. There is a T0 for each train family, and even if the SW change, most components remain untouched, and the test bench is modular. The bay can be connected to others pieces of HW and SW. Executable models can be put in units of the simulation bay, and any number of simulation bays can be added as needed.

T0 has a broad range of scopes and activities regarding the use of their solution, among which the validation of functional specifications: the same validation done in MIL/SIL is performed on the real components.

Even if we suppose that the validation done in the desired solution combining MIL and SIL is deemed satisfying regarding the specifications, T0 solution integrates or interfaces with HW, which has its own benefits. It possesses a lot more of calculation power while adding information and capabilities such as time response through the cable network.

T0 also has needs for integrated model of the system behavior as well as constraints and properties to check it. The VB team and T0 team are independent but there is in fact some redundancy in their tasks, responsibilities and skills. This can be justified by the fact that the two solutions are not always deployed on a same project. However, it can happen that they build similar co-simulations and models separately. There is a need to track the system behavior through its models and the V&V activities performed on it. The solution to this issue is less technical than methodological and implies to solve the same issues as the one identified for VB. This relates to the needs **N_2.3.** and **N_2.4.**, as well as the challenges **C_5.** and **C_6.**.

2.3 Needs analysis

This section presents the needs of BT regarding the solution(s) to be developed as part of an integration and validation process using models. The needs presented here result from the analysis of current solutions in BT.

2.3.1 List of issues

We present here the list of issues analyzed in the previous section:

I_1. Requirements traceability is lost between modeling steps.

I_2. The specified information is too detailed for high-level SOIs.

I_3. There is no modeling method for the specification of the expected behavior.

I_4. There is a lack of system concepts to formalize and model the behavior.

I_5. The models created must be consistent with the modeling method.

I_6. Concepts of modes are not correctly expressed in SysML models

I_7. There is no definition or method explaining what a state is and how it should be defined.

I_8. There is no clear distinction between the evolution of system and the evolution of its behavior.

I_9. The way the states are used and associated to SysML elements is not clear.

I_10. There is a confusion on what a mode is and how it is used.

I_11. The hierarchy between modes is not clearly defined.

I_12. The way the different types of information are characterized and correlated is not clear.

I_13. Constraints between possible simultaneous states are not formally defined.

I_14. The differences and relationship between states and modes is not clear.

I_15. States and modes do not follow the evolution of the SOI.

I_16. There are no structure upon which building and organizing the system behavior before considering a design.

I_17. Possible actions during the execution of the executable models are not available.

I_18. Guards on transitions are redundant, hard to specify, not readable and not linked to states.

We analyze these issues one by one, and express corresponding needs to be answered by the solutions developed.

I_1. Requirements traceability is lost between modeling steps.

This needs imply that information should be traced in the models, so that a new requirement linked to a model can be traced to previous requirements associated to the model. Traceability links hence have to be ensured between modeling elements. There is a need to enforce such a traceability. This can be linked to the needs **N_2.3.** and **N_3.2.**, as enforcing traceability link can be performed by a model verification solution.

I_2. The specified information is too detailed for high-level SOIs.

It is implied in BT SysMM that each step targets a single SOI. We consider that the level of abstraction of the information manipulated should be the same as the one of the SOI. If each piece of information is expressed at a different level of abstraction, then there can be no complete, integrated representation of the system information for a given level. It also enable engineers to use and share the same information, so it relates to the need **N_3.**. We define the need:

N_3.4. Express information at the SOI's level of abstraction.

I_3. There is no modeling method for the specification of the expected behavior.

This issue can be linked to the needs **N_1.** and **N_3.3.**.

I_4. There is a lack of system concepts to formalize and model the behavior.
I_6. Concepts of modes are not correctly expressed in SysML models.
I_7. There is no definition or method explaining what a state is and how it should be defined.
I_8. There is no clear distinction between the evolution of system and the evolution of its behavior.
I_9. The way the states are used and associated to SysML elements is not clear.
I_10. There is a confusion on what a mode is and how it is used.
I_14. The differences and relationship between states and modes is not clear.
I_15. States and modes do not follow the evolution of the SOI.

All of those issues relate to the definition of the concepts of states and modes, which are used to define the system and its behavior. We can define corresponding needs, that relate to the need **N_3.1.**:

N_3.1.1. Define the concepts of states and modes.

N_3.1.2. Define the link between states and modes

N_3.1.3. Use state and mode concepts to model the system and its behavior

N_3.1.4. Manage information expressed in states and modes

I_5. The models created must be consistent with the modeling method.

This issue relates to the need **N_3.2.**.

I_11. The hierarchy between modes is not clearly defined.
I_16. There are no structure upon which building and organizing the system behavior before considering a design.

Modes will be used to structure the behavior of the system, which relate to the need **N_1.**. We hence define the need:

N_1.4. Define a structure around which organizing the model of the behavior.

I_12. The way the different types of information are characterized and correlated is not clear.
I_13. Constraints between possible simultaneous states are not formally defined.

Correlating information relates to the need **N_1.2.**. Defining constraints on states enable to correlate information.

I_17. Possible actions during the execution of the executable models are not available.
I_18. Guards on transitions are redundant, hard to specify, not readable and not linked to states.

These issues relate to the need **N_1.3.**

2.3.2 Derived needs

We obtain a new list of needs:

N_1. Specifying the train behavior.

- **N_1.1.** Specify dynamic aspects between use cases.
- **N_1.2.** Correlate the information expressed in different scenarios and use cases.
- **N_1.3.** Formalize the circumstances enabling the execution of each use case.
- **N_1.4.** Define a structure around which organizing the model of the behavior.

N_2. Checking the specifications for errors.

- **N_2.1.** Define V&V requirements for the integrated system behavior
- **N_2.2.** Provide V&V solutions of the behavior based on SySML models
- **N_2.3.** Ensure traceability and continuity of specifications and validation results
- **N_2.4.** Fill the gap between tools, models and development steps

N_3. Enabling communication of information between engineering teams.

- **N_3.1.** Define an ontology that enables us to express system concepts in all models
  - **N_3.1.1.** Define the concepts of states and modes.
  - **N_3.1.2.** Define the link between states and modes
  - **N_3.1.3.** Use state and mode concepts to model the system and its behavior
  - **N_3.1.4.** Manage information expressed in states and modes
- **N_3.2.** Develop a model verification solution according to the ontology and modeling method provided
- **N_3.3.** Implement the solution in a method supporting the system development
- **N_3.4.** Express information at the SOI's level of abstraction.

2.4 State of the art

We present here existing solutions or leads in the research fields regarding the needs identified and the challenges to address.

2.4.1 Specifying the train behavior

We first consider the need **N_1.** and explore existing ways of specifying a system behavior. The goal is here to specify it independently from a design. This means that in terms of development, this task is related to the *requirement analysis*. We use here the term requirement analysis as defined in [1]: a "systematic investigation of user requirements to arrive at a definition of a system". While requirement analysis in BT is outside of the scope of this work, operability is not. In operability, requirements are expressed through scenarios and use cases expressed using SysML elements and diagrams. The goal is here to formalize, integrate and represent information in SysML models.

To consider both the standards and the current leading effort in MBSE research, we refer to the working groups on MBSE methodologies related to the Object Management Group (OMG) and the organizations associated [40, 41, 42]. .

Referring to a system engineering book published by INCOSE [41], we see that in the proposed technical process for system development, the behavior is first specified by operational scenarios. The behavior as a whole is described by life-cycle concepts, which are document-based requirements of the evolution of a system, according to different contexts and points of view. The operability state machine created in BT can be considered as a model of one such life-cycle concept.

The Systems Modeling Toolbox (SYSMOD) presented by Weilkiens [40] describes a development process based on the definition of actors and related use cases. Use cases appear in scenarios described using activity diagrams. Each use case is later described by a sequence diagram. We see that this approach is similar to BT SysMM. A global behavior is not considered or modeled before the definition of an architecture and associated functions. Another book by Weilkiens [43] describes the same idea. The behavior specified is the one of the users, not of the system. The expected system behavior will be induced in part by these specifications, but not defined. This can be explained by the fact that the user's point of view is adopted and not the one of the system.

As a whole, these methodologies answer parts of the need **N_1.**, but show limitations regarding the needs **N_1.1-4.** associated to it.

Having considered the early definition of a system behavior in existing system devel-

opment methodologies, we now analyze the use of models in requirement analysis.

The book [44], which is based on the IREB standard, explains that models can be used to represent specification requirements. It also indicates that SysML state machines are used to express requirements regarding a system behavior, with use cases and scenarios being used as support or context for the information displayed. A *dynamic view* is presented, describing a point of view where different types of diagrams are used to characterize the system behavior. Once again, activity and sequence diagrams are used to represent scenarios and interactions with the system, but state machines are explicitly cited in being used to represent the system evolution and reactions. The following benefit of using models for requirements is presented:

"Diagram types are defined for a specific purpose and, through the available notation elements (semantics) and the way the language allows these notation elements to be combined (syntax), force the modeler to focus on a situation. For example, state machine diagrams should be used to model the necessary reactive behavior of the system under development as part of requirements modeling and not to model processes or information structures."

We see the necessity of formalizing the way the behavior is expressed and how. In addition, the idea of constraining the use of a modeling language elements and semantics to better express information can be related to the needs **N_3.**, **N_3.2.** and **N_3.3.**. In particular, the need to define and restrict what a state represent and how to use it to specify the system behavior can be linked to the needs **N_3.1.1.** and **N_3.1.3.**.

[44] defines the preconditions of each use case using the states defined in the state machines. Added to the other views and diagrams, it address the needs **N_1.2** and **N_1.3.**. It does not, however, present a way to correlate the information or the use cases themselves. Use cases are correlated by operational contexts, just as in the methodologies studied earlier. The need **N_1.1.** is not addressed. Neither is the need **N_1.4.**, as there is no indication on how to structure and correlate the different state machines. The book describes how to use SysML for various aspects of system modeling but does not include them in a modeling method for a consistent development process. This hence does not cover the need **N_3.3.**

We consider solutions using other means than SysML. In [25], a behavioral view is defined, in which a DSML is used to specify the behavior. The behavior is expressed through the specifications of inputs and outputs of the system, considered as a black box. The way the system reacts is determined by modes specified in state machines. This is a work in progress, for which a structure for the model of the system behavior and a method describing the implementation of the solution are needed.

In [45], the Event-B language is used to specify a train system behavior. In this language, the behavior is specified as the interactions of the system with its environment.

The system itself is viewed as a black box. While it is possible to build a model of the behavior this way, it is a formal method. Functional architects in BT lack the skills to use it, as it is not their responsibility to build such a model. The issue here is to capture and formalize the necessary information and then provide it to another team than can build a model. The same information, separated from a formal model, has to be communicated to suppliers, clients, etc. Solutions using event B or others formal methods are limited by the need **N_3.**.

According to [19], the current efforts to specify and model a system behavior at the requirements analysis step lack a theoretical framework to ensure that the models properly express the desired information. The same work indicates how to use SysML elements, in particular state machines, to integrate simultaneous or concurrent behaviors. It hence answers part of the needs **N_1.** and **N_1.1.**. However, there is no actual structure upon which defining the behavior around the different state machines, and the information used to specify the preconditions of the operations is not formally defined. It hence does not cover the needs **N_1.2**, **N_1.3.** and **N_1.4**.

2.4.2 Checking the specifications for errors

Verification and validation (V&V) activities are based on MBSE, as we want them to be conducted continuously, starting at the design of the system. We consider here the state of the art for the need **N_2.** and those classified under it.

The way the specifications are checked in BT is done using common solutions such as testing, simulation and execution. They are currently performed on SIL and HIL, but not on MIL, as there are no models. Though new verification and validation solutions can be developed and deployed, the first issue is to create executable models and integrate them in a MIL solution, enabling to deploy known V&V techniques. Previous solutions presented to specify the system behavior aim at creating an executable model.

In [46], a work regarding co-simulation with SysML models is presented. It is a work of reference cited in OMG working groups. However, it is based on a physical architecture and is hence dependent on a design. In [47], an Executable System Engineering Method (ESEM), related to INCOSE's works, is proposed to create executable SysML models to perform a verification of the system requirements. Properties to be verified are expressed from requirements and implemented in constraints blocks. Preconditions of use cases are expressed using states elements. However, the information does not appear to be directly correlated. Only one state value can be specified at a time as a precondition of a given operation. It means only one state is defined for a given component, lacking a formalization of what the state is, what it means and what is the information associated. Again, the behavior is structured around a physical architecture. This solution covers the same kind of need as **N_2.1.**, **N_2.2.** and **N_2.3.**. The way the semantic and the fidelity to the system concepts is enforced is by construction: in order for the model to work,

the information expressed has to be consistent. This is an issue in BT. Specifications have to be made by functional engineers, who are not responsible nor are qualified for creating executable models, and work separately. The need **N_3.** is not satisfied.

[48] presents a tooled-process for the early validation of SysML Models. It highlights the difference between models used for explicit specifications, here SysML models, and others used for simulation, such as models created using SystemC or Modelica. The proposed solution is to specify the system using SysML models, and then generate code using those models to create Modelica models that can be executed. However, this solution requires to use Modelica concepts in SysML, which are not compatible with BT modeling process.

Supposing the specifications of the behavior can be put into a model, there is still a need to integrate and check them, as per the needs **N_1.2** and **N_2.**. We saw that there was a need to structure and correlate different behavior models, such as state machines. We saw that SysML models themselves could be made executable, though there may be a need to integrate other kinds of models. To perform such an integration, there exists a standard, the FMI/FMU [49]. It enables to use a common format to interface different kinds of model, and integrate them into a co-simulation. Methods [50] have been developed to design complex systems through the creation, management and integration of models using such a standard. There are works such as [51] to use the standard with executable SysML models. As we consider state machines, there is also a standard dedicated to the exchange of such models: SCXML [52]. Although both standards could be useful in BT case, there is still a need to create specification models and have a theoretical framework to formalize, integrate and manage the information in them.

In [16], a solution is proposed for specifying and testing user requirements using a behavior-driven approach. Tests are generated using the scenarios specified. The evolution of the requirements is taken in account and traced. This is an answer to the needs **N_2.1.** and **N_2.3.**, but not **N_2.2.**. The dynamic aspects are not covered. Tests, while useful, can detect errors but not their absence.

There is a difference between checking the expected behavior and the behavior as a whole. Assuming it has been specified, the unexpected aspects of the behavior, induced from emergent phenomenons, have to be identified and/or prevented. [53] express the issues in using formal V&V techniques at system level, as it may not be practical or even possible to generate all possible interactions with a system or explore all of its state-space. The proposed solution relies on tests and automatic properties generation. It covers the needs **N_2.1-3.**. Testing is already the favored validation approach in BT for system level analysis. Being able to properly generate properties to be checked or relevant scenarios is an issue, which originates from the needs **N_1.1-4.** which are not covered here.

2.4.3 Enabling communication of information between engineering teams

In order to enable or improve communication between engineers and stakeholders, it is necessary to use the same vocabulary. In MBSE, the models created characterize real aspects of the system developed. The information they contain has to correspond to identified concepts enabling to capture and understand the information. This mainly relates to the need **N_3.1.** for an ontology.

[54] presents the current state (as of 2012) of ontology in system engineering. His work is referenced by the Ontology Action Team (OAT), which is part of the INCOSE Model-Based System Engineering (MBSE) initiative. Graves highlights the needs for concepts that can describe "big systems" globaly and independently from the modeling languages used, so that the information can be transmitted across the development process. The definition of a proper ontology hence relates to the need **N_2.3.**. Graves also indicates that the ontology used and the way it is expressed in the models have to be checked, which relates to the need **N_3.2.**.

[12] presents an ontology developed for system design. The solution proposed relates to the formal definition of an ontology, by defining concepts and their relationships. In BT, the issue regarding the ontology is related to missing concepts and ways to apply and check the ontology in models. The ontology used is to be deployed on a modeling process, the information expressed in the models is not intended to be formalized in documents and textual requirements yet. The issue of defining and using an ontology in a modeling process is presented in [55]. Models express information regarding a given domain independently from the system represented. SysML enables to represent part of the behavior or the structure of a system, independently of the system itself, its abstraction level, or the domain to be characterized at a corresponding scope or step of the development process.

Conceptual modeling is something that is evolving and is increasingly needed in modeling and simulation [22], as it enables to characterize aspects of the real system while integrating considerations from different domains and scopes. System concepts associated to abstract models can serve as a foundation to less abstracted information and models. Mastering the expression of concepts in models allows the verification of their semantic and ensure that everyone creates and understands models the same way. The needs **N_3.** and **N_3.1.** are related to the need **N_2.** and those defined under it.

Regarding the need **N_3.2.**, we already saw that some solutions were to build executable models, counting on the fact that the information expressed had to be consistent to enable the execution. This is neither suited to BT, who needs to separate specification from execution, nor it is reliable, as the information is not formally expressed. This prevents a proper analysis and communication of the information in the models or their reutilization. [56] proposes to use the *semantic web*[57] technologies to support the development of systems using SysML models. Such technologies allow to share data linked

to formal ontologies. However, in BT, the ontology is not formalized in the first place, and there is no structure or process upon which sharing data. In [58], OCL constraints are used to verify UML models. It is applied to check that a specified software, or at least the code generated from the related diagrams is consistent. It does not relate to an ontology expressing system concepts through elements of representation, which would need to be supported by a method. A more recent work [59] shows that the use of OCL constraint to check UML models is still a work in progress. Other works such as [60], still in progress at the date of the thesis presented here, show the use of such verification mechanism coupled with ontologies to check the application of system concepts in models. This has been independently considered in this work, as shown in chapter 5.

The state of the art regarding existing definitions of states and modes and related to the needs **N_3.1.1-4.** will be studied in chapter 4. An example of similar needs and applications can be found in [61]. This work present the definition of an ontology related to system states and applied to SysML models.

The needs **N_3.3.** and **N_3.4.** appear in the projects already presented. In the case of BT, answering them require to answer the other needs and hence are not considered separately for the state of the art.

2.5 Contributions

The needs presented here are answered in the three contributions of this thesis. A short presentation of the contributions was given in the introduction. We present here the needs to be addressed by each contributions.

2.5.1 Concepts of states and modes

N_1. Specifying the train behavior.

N_3.1. Define an ontology that enables us to express system concepts in all models

- **N_3.1.1.** Define the concepts of states and modes.
- **N_3.1.2.** Define the link between states and modes
- **N_3.1.3.** Use state and mode concepts to model the system and its behavior
- **N_3.1.4.** Manage information expressed in states and modes

N_3.4. Express information at the SOI's level of abstraction.

2.5.2 Model verification method

N_2. Checking the specifications for errors

- **N_2.3.** Ensure traceability and continuity of specifications and validation results

N_3. Enabling communication of information between engineering teams.

2.5.3 Behavior verification method and model

N_1. Specifying the train behavior.

- **N_1.1.** Specify dynamic aspects between use cases.
- **N_1.2.** Correlate the information expressed in different scenarios and use cases.
- **N_1.3.** Formalize the circumstances enabling the execution of each use case.
- **N_1.4.** Define a structure around which organizing the model of the behavior.
- **N_2.** Checking the specifications for errors

- **N_2.1.** Define V&V requirements for the integrated system behavior
- **N_2.2.** Provide V&V solutions of the behavior based on SySML models
- **N_2.3.** Ensure traceability and continuity of specifications and validation results

N_3. Enabling communication of information between engineering teams.

- **N_3.2.** Develop a model verification solution according to the ontology and modeling method provided
- **N_3.3.** Implement the solution in a method supporting the system development

Chapter 3

Background: system theory and engineering

The study presented here revolves around systems, their behavior and the development process of both. Those should be properly defined. One of the goal of this PhD is to represent a system and its behavior. In order to do so, we analyze the theory around system.

The notion of system itself is something that goes beyond the scope of systems engineering, where it can already change depending on the sources or solutions developed. The intent here is to model the system and its behavior as part of an engineering process, falling under the scope of MBSE. As shown in the INCOSE roadmap in Figure 1.2, there is no definitive MBSE theory, concepts or formalism, and defining them constitutes a research topic. We consider here scientific studies, established theories and concepts to capture the context and needs of our study.

To have a better idea of the concept of system in general, we turn to the *general systems theory*, developed by Bertalanffy [62]. The general system theory forms the basis of this field of study [63]. It contributed to the development of systems engineering [64, 63, 65], which in return improves the scientific approach and definition. Theory improves practice and practice improves theory, a dynamic shown in the *System Praxis Framework* developed by INCOSE and ISSS [66].

We explore in this chapter the general system theory and current systems engineering approaches on which we can base our solution.

3.1 System theory and definition

Systems engineering can be considered as one of the applications of system theory. We are interested here in the general system theory, and will consider in particular the work of Le Moigne [67]. Le Moigne [67] explains the general system theory, its history and context, and proposes a system modeling approach based on it, which relates to our goal. We will compare it with more recent works, such as [68].

3.1.1 System concept

Le Moigne's definition of a system [67] has been intended as a description, in order to be compatible with most definitions of the concept. It is defined as : *"an active and stable object evolving in an environment in relation to a few goals"*. An object that is "active" conduct one or several *activities*, which are defined as ongoing functions. "Stable", regarding the other form of the definition, refers to the structure. A more detailed and decomposed definition of a system is provided, and can be roughly translated as follows:

- Something (meaning anything we assume can be identified)
- in something (an environment)
- for something (purpose or project)
- does something (activity = operation)
- through something (structure = stable form)
- that is transforming over time (evolution).

The terms in the definition are intentionally vague, expressing the concept in the natural language with the key terms linked to concepts detailed in the theory and put in parenthesis. A system as presented in the theory can correspond to any object, entity or phenomenon in the universe that we want to consider and study. It corresponds to a scientific method and an epistemology.

Many definitions of a system, and in particular a complex system, define its concept as a group of elements linked by relationships. That is what the term "structure" refers to: "Structure defines components and their relationships"[68]. Referring to the *structuralism* paradigm, Le Moigne explains that there are properties induced by the integration of all elements of a structure that go beyond the sum of the properties of each individual element.

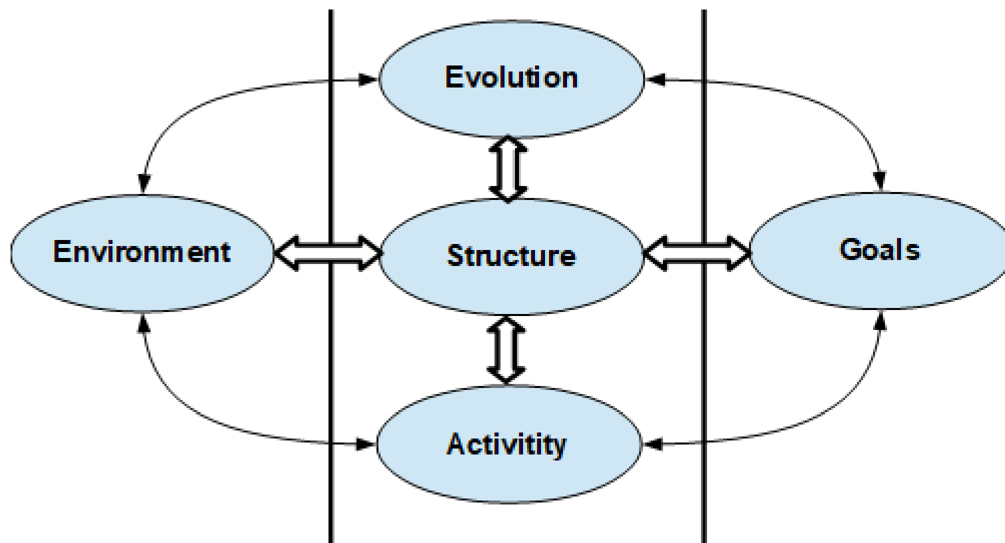

Figure 3.1: Systemic paradigm [67]

A first comment regarding this theory is: even if we do not define nor consider the different elements forming the system, meaning the structure, we will have to consider the properties induced by them. A property, as defined in [1], is a *"responsibility that is an inherent or distinctive characteristic or trait that manifests some aspect of an object's knowledge or behavior"*. We assume here that such a definition is suitable to the meaning intended by Le Moigne and Bertalanffy.

The induced properties are specific to the system, as they result from the integration of different elements that do not possess them individually. We hence consider that we always study a structure, even if none of its elements (except for the system containing them) are represented.

Le Moigne bases his definition on the system paradigm, shown in Figure 3.1.

The general system theory is the theory of object modeling [67]. To conceive or study a system is to represent it, by the way of models. The definition of a system proposed by Le Moigne provides five concepts (activity, structure, evolution, environment, goal) intended to support the creation of models. Environment and goals are external to the system, which is characterized by the notion of activity, structure and evolution. Those are found in recent approaches, such as *system thinking*, developed from the general system theory:

"The most basic relations in systems thinking are function, structure, and process. Briefly, function is contribution of a part to the whole; structure is an arrangement in space; and process is an arrangement in time. These three relations as essential to systems thinking. ... a design approach dealing iteratively with structure, function, and

process is the "enabling light" of systems methodology. Structure defines components and their relationships, which in this context is synonymous with input, means and cause. Function defines the outcome, or results produced, which is also synonymous with outputs, ends, and effect. Process explicitly defines the sequence of activities and the know-how required to produce the outcomes. Structure, function, and process, along with their containing environment, form the interdependent set of variables that define the whole." [68]

Functions, according to Ing, are objects with a goal, an intent, a *usefulness*. Their outputs are used for one or several purposes. But an input does not necessarily come from a function, hence why there are inputs associated to the structure: they come from properties that are pieces of information which can be useful but have no goal or usefulness on their own. They require to be processed and/or are correlated with others. What the system does or produces depends on both the functions performed and the properties known or perceived.

The concept of activity is complementary of the structure. An activity, rather than defining a function, defines an-on going function, meaning a change conducted over a period of time. A structure is considered at a point in time. The activities describe what the system does over time, the structure describes what the system is (in term of element composition and properties) at a given point in time.

A structure, through its properties, is used to define the functions that can be executed. The activities performed hence depend on the way the structure changes over time. This is characterized by the concept of evolution. Both the structure and the activities evolve over time. The structure changes over time, enabling new functions or preventing others to be performed. The activities themselves can change the structure as they are performed. The evolution of both the activities and the structure are related but distinct.

In BT, the evolution of the system activities is specified in scenarios, be it in train scenario using activity diagrams or use case scenario using sequence diagrams. We saw that the issue in these representations was that the execution of activities was conditioned by the circumstances in which the system is, which were not formally specified.

To study the evolution of the structure is to study the evolution of its properties, hence of the system properties. We argue that in the case where we do not define a structure for the system, we still need a way to represent its properties. Such a representation could be used to express the conditions for which the activities can be performed.

We have defined the concept of behavior as "the peculiar reaction of a thing under given circumstances". To specify the system behavior is to specify what functions it can perform at a given point in time. Considering the definition of a system by Le Moigne,

we see that defining the behavior requires for us to link the evolution of the system structure and the evolution of the system activities. In order to do that, we would need a representation of both.

As activities are performed over periods of time, they are described inside processes. The evolution could be seen here as replaced by the concept of process, but it is specified in the theory that it is the activities that evolve through processes, not the structure. Le Moigne defines the concept of process as a physical whole that can change or transform over time, referring to the structure, as shown in the relation in Figure 3.1. He references another definition, according to which any change of matter, energy or information over time is a process. Those two concepts of a process may seem incompatible, though it is not the case. A process has to be understood as an evolution over time, be it in the system activities or its structure.

In system thinking, functions are highlighted instead of activities. Rather than considering ongoing functions, the focus is put on what happens when they are performed. We shall adopt the same point of view. As we want to analyze the system behavior, we are interested in what the system *can* do instead of describing what its does, something that has already been described in train scenarios.

Le Moigne presents as a fundamental hypothesis of System Theory the idea that any model of an object behavior can be conceptualized as a process. However, this can lead to some confusion, as we saw that we need to express the evolution of both the system properties and of the realization of its functions to represent its behavior. Though related, those evolutions are distinct. We need to specify two processes that influence each other. Those two processes need to be part of a same model.

3.1.2 System representation

Our goal is to represent the system and its behavior. We need to create two processes that describe the evolution of the structure and the evolution of the activities. Those two processes influence each other.

A system is never completely represented, explained or understood. System theory is based on the assumption that an exhaustive, true, predictable representation of the system is not possible. What can be done is a description fitting the needs and purposes of the people conceiving a system. Le Moigne considers that what matters for the people considering a system is what the system does. While it may not be possible to capture everything the system does, the system activity and outputs are described and associated to goals. They are defined by the fact they are given purpose. They correspond to what the system is perceived to do in the vision of the one defining the system.

Regarding the structure, we saw that a representation of the elements composing the system was not always available or desired. What matters is in fact the properties of the system. Those properties are expressed through information characterizing the system.

Assuming a system is not predictable, or at least dependent from the environment, assessing the current situation and information of the system requires to consider it at a point in time. This is why the notion of *state* is developed.

The concept of state is not clearly defined. It is presented by Le Moigne as a "photography" of the system behavior at a point in time, obtained by considering inputs and outputs. It has been established that knowing the behavior of the system requires knowing its state. The proposed definition of state is empiric, the states are observed or deduced, not defined. A state is a perception of what the system *is* at a given time by the one defining it. As the system functions are defined with a goal, we define states which have a utility toward the realization of the system functions.

The system is defined in relation to its environment, so we can define state regarding the situation of the train in this environment. However, this is not enough for describing its behavior. The system has its own internal state resulting from its evolution through time. Without defining the elements composing the system, it is necessary to consider an internal state of the system.

There is no intent toward making the system evolve. The intent is to model the evolution of the system capabilities. States are a description of what enables the system functions and capabilities (the choice of action available) to evolve and be performed. The process describing the evolution of a system structure, expressed using states, supports the process describing the evolution of the system capabilities. The process describing the evolution of state is a description of the system situation. The process showing the evolution of capabilities should specify the relationship between conditions on the states and functions available. This last process is the one that correspond to a description of the system behavior.

We already know that the system and the evolution of its properties can be described by the evolution of its states. However, we did not establish how to represent the evolution of its capabilities. We need to define a modeling element that links the realization of functions to conditions on the current state of the system.

A system is something assumed to exist and susceptible to be identified. To define it is to describe it, as the notion of system is associated with the notion of representation. It means that by considering a system, one already has knowledge, or at least an idea, of what it is, if only in relation to its environment. If we are to conceive a new system, describing, meaning specifying its structure, state and properties, is not enough. We will have to specify its inner mechanisms, even at a high level of abstraction, for it to be consistent. This changes the goal from describing to specifying a system. We have to consider the modeling of a system from an engineering purpose and point of view.

3.1.3 Method

A method can be understood as providing a way to achieve a goal. Here we consider a method as a way of representing a system. The general system theory is a scientific method that is presented as a modeling method. Coming back to the issues considered in this work and presented in the previous chapter, we need a way to represent an integrated system.

System theory and systems engineering are both linked by a same need: representing a system. We present here the modeling method described in the general system theory and developed by Le Moigne.

Le Moigne defines his method using four principles that he opposes to those of the scientific method defined by Descartes [69]. Those four principles are:

- Relevance: any object considered is defined depending on the user intent. If the intent changes, so do the definition of the object.
- Globalism: the object studied should be considered as an active, incorporated part of a bigger whole. It should first be perceived globally.
- Teleology: study and understand an object behavior through the goals associated to the object.
- Aggregation: instead of an exhaustive representation, aggregate elements from an object that are deemed relevant and sufficient to the study.

Our need in integration can be found through these four principles: considering the system as a whole by aggregating relevant information and elements with a specific goal in mind (behavior representation and validation).

According to Le Moigne, a model is obtained using a representation system. We already described the elements used by Le Moigne: processes, states, activities, structure, functions. Those can be found in system thinking.

According to Ing, a SOI is composed of properties and a behavior, and is considered inside a bigger whole [68]:

"In the systems approach there are also three steps: 1. Identify a containing whole (system) of which the thing to be explained is a part. 2. Explain the behavior or properties of the containing whole. 3. Then explain the behavior or properties of the thing to be explained in terms of its role(s) or function(s) within its containing whole."

Following this approach, a system is first to be considered in an environment. It is then considered itself as a whole with properties and behavior. Elements composing the systems are then considered in the context of the system containing it and the other elements at the same level of abstraction. Each element is to be considered an independent whole with a behavior and properties. This is similar to a holonic structure [70]. A holon is an element that can be considered as both an independent object and as a part of a bigger whole. In this case, elements (holons) of a given level of abstraction are contained by a bigger whole with properties and a behavior induced by their union and bigger than their sum.

From a systems engineering point of view, information regarding the environment is either known or provided, as it is the system that is to be developed. In BT, external systems are either known or are the responsibility of external providers. Their information and models are either received or to be provided later. The behavior of the user(s) is specified in scenario but cannot be specified or predicted as a whole. Engineers in BT have to provide a system that satisfies given expectations. This is why there is a need to know what the system can do, to ensure that the expectations are correctly integrated and satisfied, without creating unknown or unwanted behaviors. Ideally, only the expected interactions should generate a response from the system, which should be constrained accordingly.

3.2 Systems engineering

"Science seeks to understand and describe properties and relationships of things in the world while engineering strives to understand these properties and relationships in order to apply them to solutions to engineering problems. Engineering then will create new properties and relationships in their designed artifacts, properties including such things as behavior, functionality, performance, structure, economy, practicality, and so on." [65]

Scientific theory does not pursue the same goal (understanding, explaining) as engineering (solving problems, creating solutions). As such, merging theory and engineering is an issue in itself. It is illustrated by the *System Praxis Framework* developed by INCOSE and ISSS [66] to address the problem. As systems engineering is based on the general system theory, the analysis made in the previous section can be applied here. However, aspects specific to engineering activities have to be considered.

3.2.1 System concept

Conceiving a system in systems engineering is done with a different mindset than the one in system theory. The concept of system in systems engineering is more specific that in system theory, as the goal is to conceive and develop systems for human purposes. The definition used here is the one provided by INCOSE [41], which is supported by the standards [1] and derived from the general system theory. According to this definition, as system is:

"...an integrated set of elements, subsystems, or assemblies that accomplish a defined objective. These elements include products (hardware, software, firmware), processes, people, information, techniques, facilities, services, and other support elements."

Considering this definition, elements constituting a system can be traced to different disciplines required to develop them. Developing individual parts of a complex system may imply different types of engineering activities specific to these elements. It means that it is possible to have information, and even requirements, on what may constitute the system. A system is not born from nothing. Engineers generally have an idea of the system they are to develop. Capturing the right information and expectations regarding a system-to-be is the first step in order to develop it.

Each engineer working on the system expresses an information about it, based on requirements, specifying something desired, relevant or necessary regarding the system. From then, it can be assumed that the information regarding what is *expected* from the system is given. Knowing if these expectations are right or complete is an issue, before worrying if these expectations are satisfied. In both cases, it requires having an idea of the system resulting from the expectations. Expressing expectations is one thing, expressing the *system expected* is another.

The earlier an error is made (and not detected), the more the cost [9]. Error in the very design of the system are hence the more damaging. From there comes the need to integrate the information expressed regarding the system, in order to obtain a representation of the whole system, at least from a given perspective such at its functional behavior.

Integration is not something that is just performed, it is something that is specified. Let's consider two different expectations regarding the system that can affect a same aspect of the system and can be realized in the same time. There are different ways to integrate them: letting them happen at the same time, making one overruling the other, etc. For example, a project in BT had two use cases that related to the behavior of screen wipers: one enables to activate and select the speed of the wipers, while the other washes the screen and activates the wipers for a short cycle to remove the water. No speed was specified for the wiping cycle during wash.

There were no specification on how those two use cases worked together. If the wipers are already active, is a cycle launched? Does it deactivate the wiper afterward? Can the cycle be interrupted by reducing the speed of the wipers to zero? Answering these questions is the responsibility of the functional architects, who lack a way to identify such conflicts. There is a need to know which use cases can happen at the same time in given circumstances.

Parallelism and concurrency between functions is one example of specifying integration. To apprehend what the system is and how it behaves, it is necessary to specify how to put in relation the functions, information, parts and properties regarding the system. Another form of integration corresponds to the definition and restriction of the system states. The system behavior depends on the circumstance the system is in. The circumstances are expressed by the system states. To restrict the system behavior to what is expected and/or possible, the circumstances, hence the possible states, should be constrained to only allow the behavior specified. Those constraints correspond to properties of the system.

System properties are more than the sum of the properties of its elements. It is also true for its behavior. It is the principle of *emergence* [8]. According to Abbott, *"the properties and behaviors of complex system are not describable as a simple, closed-form, mathematical function of the properties and behaviors of the system's components. Typically, the best one can do is to propagate the descriptions of the component interactions and see what happens at the system level."* [8]. As said earlier, the system is studied before its elements. If emergence is to be analyzed after developing the group of elements causing it, we see that in practice there is a need to anticipate it. Specifying the system behavior is the same as specifying the emergent behavior of the elements it is composed of.

A group of elements with their own behavior and properties produce emergent properties and behavior as a group that have to correspond to the ones specified for the system. What is to be avoided when developing the system is missing or adding properties or behaviors. There is a need for traceability between elements of the system and the system itself:

"Early pioneers of SE and software engineering, such as Yourdon (1989) and Wymore (1993), long sought to bring discipline and precision to the understanding and management of the dynamic behavior of a system by seeking relations between the external and internal representations of the system. Simply stated, they believed that if the flow of dynamic behavior (the system state evolution) could be mapped coherently into the flow of states of the constituent elements of the system, then emergent behaviors could be better understood and managed." [41]

This can be linked to system theory, Le Moigne using this notion of state evolution and flow. States of the system elements shall be captured and traced to the system state to check that the same properties and behavior are enforced.

3.2.2 System representation

There is a need of state processes, behavior modeling element, integration and traceability in the representation of the system. This all relates to the use of models. A model is a representation of the system that is created for the purpose listed in the definition. The standards [1] gives several definitions of model:

- "A representation of a real world process, device, or concept."
- "A representation of something that suppresses certain aspects of the modeled subject."
- "A semantically closed abstraction of a system or a complete description of a system from a particular perspective."

The need to represent system to define them is the reason systems engineering evolved toward a model-based approach, the MBSE. It is defined by INCOSE as:

"Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases." [4]

MBSE is there to support engineering activities in the development of system through the use of models and methods. As engineers have knowledge regarding the system they develop, they need to express it. The information and description of the whole system often only exist in the mind of engineers, as there is a lack of solution for describing and integrating the system in models. The issue is that engineers do not work alone and need to exchange and communicate. More importantly, information and specifications regarding the system have to be formally defined, otherwise they are overlooked or subject to interpretation and cannot be part of a modeling method. Describing the system state and behavior has to be done through dedicated models.

The information expressed in a model should correspond to a vision and concepts that characterize the expected system:

"A system concept should be regarded as a shared "mental representation" of the actual system. The systems engineer must continually distinguish between systems in the real world and system representations." [41]

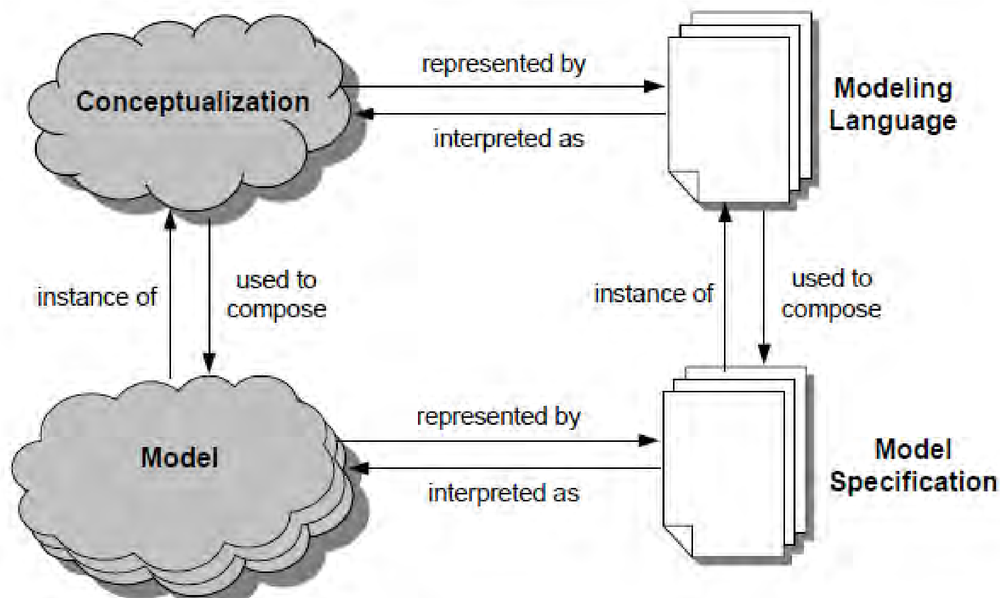

Figure 3.2: Relations between conceptualization, Model, Modeling Language and Specification [71]

Producing a model implies using one or several *modeling languages*. A modeling language possess a syntax, supported by a semantic, illustrating concepts. Those concepts related to the system to be represented. The same concepts can be expressed in different modeling languages, but the models obtained won't contain the same information or semantic. This is expressed in Figure 3.2.

Engineers following a same modeling method can produce different models, interpreted differently. As the development of the system is done by different teams, it involves people from different backgrounds and skills, using sometimes different modeling languages. There is a need to manipulate the same concepts and express them the same way in a given model and language.

Following the steps of the system approach defined by Ing, representing the system and its properties would be done using states, which then enable to define the behavior. The behavior is often described using processes and evolution of the states and activities, but what we need is a way to *specify* it. Description and specification are distinct activities, as we want to describe the system expected without taking decisions on the design and only then start to specify it. However, those two activities are not clearly separated:

"As complexity grows, the line between specifying behavior and designing behavior is blurring. To the extent the software design reflects the systems engineer's understanding, the software will perform as the systems engineers desire." [14]

By describing what is expected from the system and its behavior, we want to describe the whole system itself. By doing so, we are sure to know what is the desired system, something necessary to specify and validate a design. Integrating the information of the expectations require to specify the logic by which they constitute a coherent whole. The integration itself is hence both a description and a specification. Coming back to what is done in BT, we see that such an integration is first realized in the operability analysis, which is a step that constitutes the transition between requirement and design activities.

A key concept to system modeling is the notion of state, which is as critical in engineering as it is in theory:

"Understanding state is fundamental to successful modeling. Everything we need to know and everything we want to do can be expressed in terms of the state of the system under control." [14]

The concept of state has to be clearly defined and correctly expressed in any model, with the information traced.

Modeling the evolution of the system states is not enough. It has to be linked the evolution of the system capabilities. The behavior is not just what the system *does*, it is what the system does *in which circumstances*. To describe and/or specify the system behavior, we need an element that links the system structure with the evolution of its actions. We will now refers to this element as *mode*.

3.2.3 Method

As said previously, the difference between the scientific modeling method and an engineering modeling method is that the scientific method only needs to describe a system, whereas an engineering method needs also to *specify* a system. In addition to of providing intents and context, we present here systems engineering methods as a mean to complete what is lacking in the scientific one for our purpose. We do not pretend to make a full analysis of how a system is developed, but we want to show an overview of the design process of the system behavior.

As BT SysMM is based on the ISO/IEC/IEEE 152882015 standard, we consider here the different processes it specifies as part of its development:

- Business or mission analysis process.
- Stakeholder needs & requirements definition process.
- System requirements process.
- Architecture definition process.
- Design definition process.
- System analysis process.
- Implementation process.

Our focus is put on the *Stakeholder needs & requirements definition process* and the *System requirements process.* Those steps are described but not associated to a modeling method. The reason for that is that there is no official modeling method in BT. Such a method depends on a modeling language and concepts associated to it. SysML is a modeling language that can be considered as a standard, but it does not have an official modeling method. The reason being that SysML is made to be customized, allocating specific system concepts to its modeling elements so that it correspond to the user's needs.

There is no transition steps between requirements specification and design. We saw that to integrate the system expectations meant to specify the integration itself. This is why the operability analysis in BT is in this context an innovation of BT and constitute a crucial point in our approach.

A system goes through different stage in its life-cycle. Such a life-cycle is specified by the ISO/IEC/IEEE 24748_1_2016 standard:

- Concept stage.
- Development stage.
- Production stage.
- Utilization stage.
- Support stage.
- Retirement stage.

We specify specification solutions at the concept stage, where the information specified cover the utilization and support stage of its life-cycle.

3.3 Synthesis

In this chapter, we positioned ourselves regarding the system theory and practice. We provide the context and the scope of our work regarding system development. We also justified and detailed some issues and challenges presented in the previous chapter, with hints to the solution needed. In particular, we saw that in order to represent, specify and integrate a system and behavior, we need to:

- Define the concept of state applied to systems engineering activities.
- Define a concept for behavior modeling, linked to the system states.

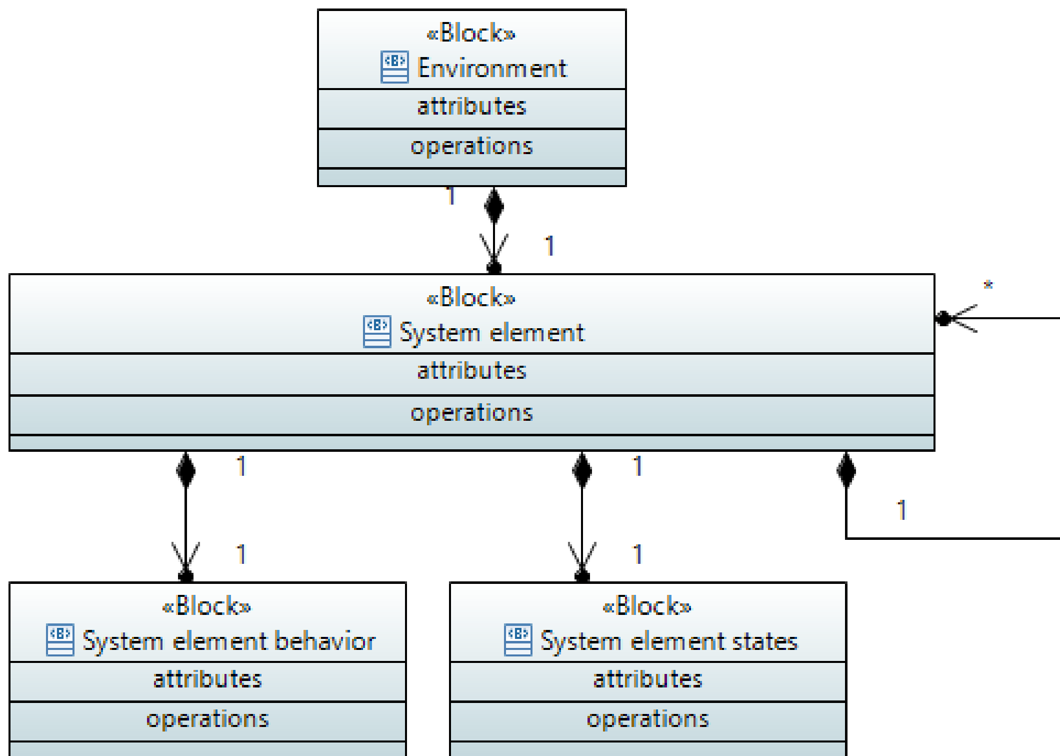

Figure 3.3: Composition of a model of the system and its behavior

According to our analysis, a representation of a system and its behavior in an environment would be decomposed into a structure of system elements, with the global system as a root of the structure. Each element would be composed of a behavior and of evolution states. This representation is shown in Figure 3.3 in a simplified way, not representing properties, constraints, relationships between system elements of a same level of abstraction or dependencies between state and behavior. Those details will be addressed later as part of the solution developed. A definition of the concepts of states and modes is given in chapter 4.

Part II

Contributions

Chapter 4

Concepts of states and modes

We saw in the previous chapters that the concept of *state* is essential when modeling a system and its behavior. The issue is that it is a concept that is not clearly defined, be it in the theory, modeling language or in methods. BT makes an extended use of *states* in its train development process, but the term can correspond to different kinds of information, modeling elements or utilization.

We already highlighted the notion of state in system theory. We now want to further explore this concept in systems engineering. We first consider its use in the standard modeling language SysML, which comes from UML [72]. What a state represents regarding the system or its behavior in SysML is not specified. It is an issue raised by Naumenko in his thesis [73] regarding UML: "*existing UML semantics are very ambiguous in presenting relations between models constructed using the language on the one hand and the subject that is being modeled on the other hand*". While the representation and semantics of states regarding the model are specified in the standard, it is not clear what a state actually represents regarding the system to be modeled. This is the gap discussed in the previous chapter and shown in Figure 3.2.

Aside from UML, there are "state" elements or concepts used in different languages, tools and methods with different semantics, such as UPPAAL [74] or in Abstract State Machines (ASM) [75]. There is no consensus on what a state is [76, 77], and it is often confused with the notion of *mode*. The definition of mode is also debated. When referring to the ISO/IEC/IEEE 24765 standard, which references other standards, we see that there are a variety of definitions of what a state is.

There is a confusion between states and modes but both are widely used in systems engineering. It is necessary to define both concepts in the scope of our work. Bombardier Transport (BT) uses states and modes in its models but lacks a definition of the corresponding concepts.

This leads to a difficulty in capturing and integrating the right information (such as preconditions of functions) in executable models, which are themselves state machines.

Beyond the definitions of states and modes, we are interested in their utilization. While in SysML the states are used to model a behavior, we see in the definitions proposed by Wasson [76] that we can define different kinds of states and modes, capturing different information. Such information can describe or impact the behavior of the system without being able to fully describe, condition or model it.

When modeling the system behavior using a SysML state machine, the information used for the guards of the transitions has its own evolution. This evolution is done through transitions with their own guards. This leads to the creation of other state diagrams or constraints on variables characterizing information about the system but not its behavior. Referring to the general system theory presented in the chapter 3, the need to consider two different evolution relates to the issue of modeling the evolution of the system activities and the evolution of its structure. Those are two different issues, and we should use two different modeling elements to express them, keeping in mind that they are correlated. We established that the states are used to describe the system properties associated to its structure. We need a different kind of element to specify the behavior: the modes.

The goal of this chapter is to define the concepts of *states* and *modes* with the goal of modeling a system and its behavior in an integrated way.

4.1 Concept of State

4.1.1 State of the art

We consider several definitions:

- According to the ISO/IEC/IEEE 24765 standard, a state can be a characterization of the system at a given time, the value of the variables defining the system, a condition to a behavior or a function, something that determines the set of functions that can be performed, and other meanings.
- According to a systems engineering handbook supported by INCOSE [41]: "*A system is in a state when the values assigned to its attributes remain constant or steady for a meaningful period of time*". It is otherwise specified that the handbook refers to the ISO/IEC/IEEE 24765 standard for vocabulary.
- According to Naumenko [73], a state is one of two concepts at the basis of the semantic he defines, and its definition is: "*an information about a thing (object) at a given time (point in continuum) inside a context (time continuum)*".
- According to [76], a state is "*An attribute used to characterize the current logistical employment, status, or performance-based condition of a system*".
- According to common definitions in English and French dictionaries, a state is the "way of being" of a thing, in our case a system, his environment, etc. It is also referred as a condition at a given time.

Not all definitions available or considered are given here, only a representative group issued from formal or common sources. Some sources cited in this document, such as [77], have already done similar work and reference different definitions, and can be consulted as a complement.

Besides the definition of states, we consider different kinds of elements used to represent them. In many finite state-machines such as UPPAAL, we have one state active at a time inside a state space without hierarchy or concurrency, at least in one diagram. In UML and SysML state machines or in Harel statecharts [78], we have complex state structures with concurrent states in one diagram, meaning we can have several states active at the same time. In Abstract State Machines, or ASM [75], the state of the system is represented by a set of variables, called states variables, and a state is the valuation of these variable at a point in time. This can be linked to one of the definitions found in ISO/IEC/IEEE 24765. As in state diagrams, ASM enable to define transitions, conditions and constraints.

The state definition in UML can not be used as a definition of the concept. As per the analysis made by Naumenko, these elements are not directly associated to the system of study. In addition, UML defines states to model a behavior in what is called “behavioral state machines”, but we can define states that affect the behavior without modeling it. Taking the example of a train, we can define a state that indicates the current energy supply, be it internal or external sources. It will affect its operability and hence its behavior but does not condition capabilities directly. States can capture information used to condition capabilities. This information cannot be used to directly model the behavior, as each piece of information could be used as part of the preconditions of any capability. To represent the behavior, we have to define elements characterizing the execution of capabilities and not the information used to allow it, which seems to be the goal of the state machines in UML. This is supported by the definition found in the UML 2.5.1 standard [72]: speaking of the State model element,“*a state models a situation during which some (usually implicit) invariant condition holds*”.

4.1.2 Analysis

Cross-referencing the definitions given above and excluding their use to express the behavior directly, we can establish a few characteristics of states:

- They characterize a thing (e.g., a system).
- They relate to a specific kind of information, knowledge domain or way of being regarding this thing (e.g., operations, readiness, energy, ...).
- They are evaluated or considered *at a given time*.

The most appropriate definition would be the one proposed by Naumenko [73], as it is high level, relates to the object of study that we want to model, and summarizes the main characteristics of states that are common in the definitions considered. In addition, it was thought as a core element of a semantic used to give a common meaning and understanding in the definition of models regarding the system they represent. It is one of our objectives regarding modes and states: to make it clear how to define them and what they represent regardless of which language or tool we use. It also presents the advantage of considering a temporal context in which states can be defined, which should be the system life-cycle. We will keep this notion of context as it can be applied to the other definitions.

Naumenko considers two elements to represent the information regarding an object (system): *state* and *action*. State is here an information characterizing the system in at a point in space and time. An action characterizes the information linked to an evolution in space and time. This is illustrated in the Figure 4.1. We can make here a parallel with

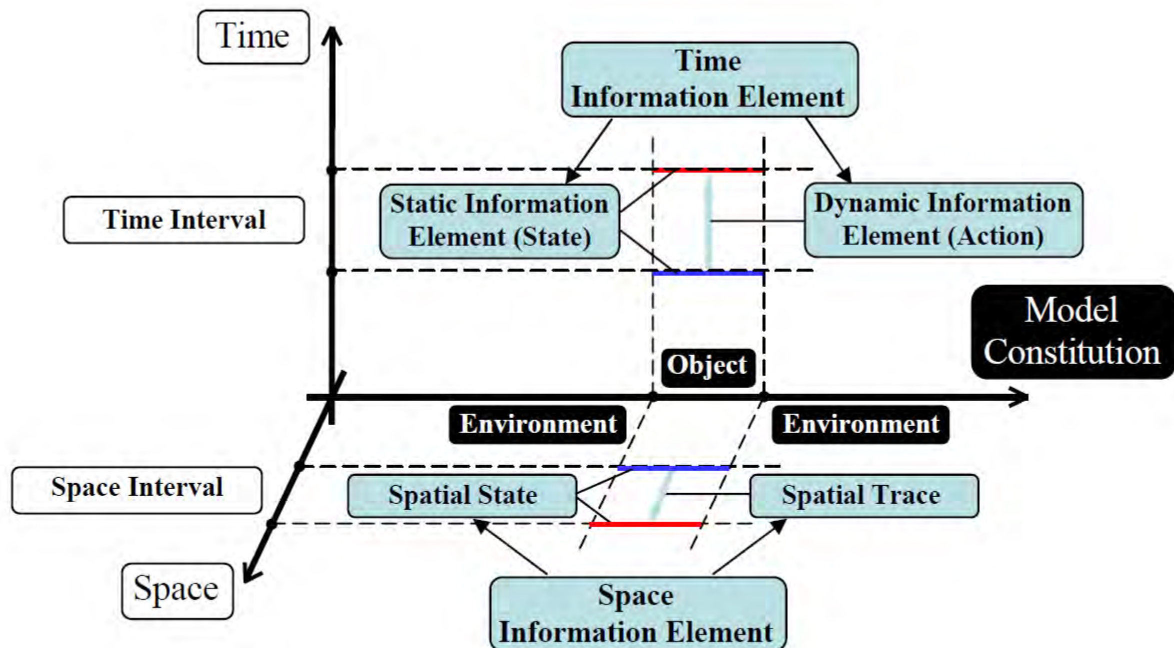

Figure 4.1: State and action concepts to characterize an object in space and time [73]

the system theory where states characterize the system at a point in time and functions or activities characterize a change. We do not consider the notion of space as the system is an abstract element in our scope.

The definition given in the INCOSE engineering handbook [41] seems to contradict the notion that a state is considered at a point in time, but we argue that it is not the case. A state is *evaluated* at a point in time. If we get the same values or few variations of them in regard of the information characterized during meaningful periods of time, then they can be abstracted as discrete values. Searching for persistent values or intervals of values can be a criterion to create state diagrams or other such models, but we consider that is it not a part of the state concept definition.

The notion of information is crucial. An information is a discriminated piece of knowledge that can be defined or acquired when considering an object. The state of the system is often considered to be a photography of a system at a point in time, regardless of the information considered. But the description of a system contains a finite amount of information. A state of a system modeled evaluate either a specific piece of information, or all of the information we associate to the system over time in our representation. We consider that a state always characterizes a finite amount of information that can be identified.

A state is also an abstraction. Even if we measure physical values to define it, those are interpreted and assimilated as an information. The point of view from which the state is defined is also important. If we consider an unmoving train and want to evaluate its operational use, then depending on the point of view, it could be considered "parked", "waiting", "in a mission", "reserved", "ready for departure", etc. When evaluating an information through the measure of physical attributes, the point of view indicates who can access the information: the train control system, the pilot or an external monitoring system. Depending on how the information is accessed and on the scope of study, there can be differences in the information received.

In SysML state machines, we can have several "states" active at the same time in a given diagram. However, an active state is either one of the higher level state, or one of the sub-states contained in a higher level state or a concurrent partition. There can be only one active state in a group of states or sub-states at a same level of abstraction and per concurrent process. We hence consider that each of these active states exist in their own context and characterize a different kind of information. It enables us to keep the definition of states as an evaluation of information in a time context, though defining several types of states characterizing different targets in a same diagram may not be relevant.

Engineers try to define a hierarchy of states. For example, Wasson [76] considers that system states (characterizing the use being made of the system) contain operational states (characterizing the mission readiness of the system). But it seems logical that a system could have the same readiness status in two different missions. Acknowledging that Wasson's scope is oriented to the definition of scenarios and mission specifications phase by phase, we consider that in general two types of state of a same object characterize different information that may have dependencies and constraints between them but exist at a same level.

4.1.3 Definition

Using the definition proposed by Naumenko, we consider the concept of state as "*an information about a thing at a given time inside a context*". In order to be independent of the representation of a state, we want to use informal attributes to characterize it.

In order to adapt our definition to the development of complex systems, we specify two attributes impacting the information characterized: the point of view and the level of abstraction. We saw that the definition of a state was different depending on who is considering the information regarding the object characterized. The level of abstraction, here divided into system, subsystem and component level, enables us to define a same information at different level of details along the development process. For example, a same state of the train energy supply can be evaluated as "internal supply depleted", "battery depleted" or a voltage measure depending on the level of abstraction.

This is also how we could ensure traceability of information and state along the development process, by refining states. Refining a state means lowering the level of abstraction which the information is expressed with. According to this analysis, the attributes necessary to define a type of state are given in Table 4.1.

Attribute	*Definition*
Target	The characterized entity
Information	The type of information considered
Context	The time scope under which the state exists and can be defined
Abstraction level	The level of abstraction of the element linked to the information considered
View	The point of view by which the state is expressed

Table 4.1: Attributes used to define a state type

We then link this definition to concepts enabling to model states using different languages and modeling objects:

- A *type of state* is a type of information linked to the object it characterizes and the time context where both the information and the object are defined. A type of state is defined using a set of states variables.
- A *state variable* is used to express the information characterized by a type of state. A state variable has values defined over the whole time context where the type of state is defined.
- A *state space* is the set of all possible values of the state variables of a type of state.
- A *state* is the evaluation of the state variables of the type of state it belongs to. Each Element of a state space corresponds to a possible state.

We do not consider transitions, initial states, guards, etc. in our definitions. Our goal is only to know what a state represents regarding the object to model and what are the kind of elements needed to model it. The representation and evolution depend on the modeling language.

4.1.4 Example

We illustrate our definition on a simple example: checking whether a common type of door is open or closed. Our target is a door, the information is the opening status and the time context is the period during which the door remains installed and in good shape. The point of view is the one of the door's user, the level of abstraction is the door as a system. We use as a *state variable* the angle made by the plane of the door regarding the one of its frame. The *state space* is any value between 0 and 90 degrees, assuming that the door can not be opened further than at a right angle. The *state* of the door is evaluated by measuring the angle at a given time. If the angle is less than 5 degree, the door is considered closed. We do no consider the lock mechanism for the definition of "closed". We can abstract the *state variable* as one having for state space only two elements, "open" and "closed", as it is the only information we are interested in. We can then represent the *state variable* in a state diagram, with two state values. We see that a same *type of state* can be represented in several ways, and that is does not necessarily keep the same information value, as we could place the door at an angle over 5 degree just for an instant, that would be considered a point in time on a measure, before closing it again.

4.2 Concept of Mode

4.2.1 State of the art

Again, we consider several definitions:

- According to the ISO/IEC/IEEE 24765 standard, a mode is a set of related features or functional capabilities of a product.
- According to the SMC Systems Engineering Handbook [79], "*The condition of a system or subsystem in a certain state when specific capabilities (or functions) are valid*".
- According to Wasson [76], a mode is "*An abstract label applied to a user (UML Actor) selectable option that enables a set of use case-based system capabilities*".
- According to common definitions in English and French dictionaries, a mode is a "way of behaving" of a thing.
- According to the Arcadia method [80], a mode is "*a behavior expected of the system [...] in some chosen conditions*".

Modes are linked to actions, functions or capabilities. As there are debates about what is a state and a mode, we also analyze states definitions that relate to capabilities:

- According to the ISO/IEC/IEEE 24765 standard, a state can be "*something that determine the set of functions that are possible or can be performed*".
- According to the SMC Systems Engineering Handbook [79], a state is "*the condition of a system or subsystem when specific modes or capabilities (or functions) are valid*".
- According to the Arcadia method [80], tooled by Capella, a state is "*a behavior undergone by the system [...] in some conditions imposed by the environment*".
- According to Jenney [81], "*States define an exact operating condition of a system, where modes define the set of capabilities or functions which are valid for the current operating condition*".

4.2.2 Analysis

Considering those definitions and other sources, we have identified several attributes and characteristics about modes:

- They characterize the behavior of a thing (e.g., the behavior of the system under this mode).
- They express a behavior regarding a set of capabilities, functions or actions (e.g., moving forward or backward, performing maneuver, etc.).
- They are defined for a set of conditions (e.g., specific states of the system).

We use the following definition found in ISO/IEC/IEEE 24765 for the concept of behavior: "*the peculiar reaction of a thing under given circumstances*". There is a distinction between the behavior characterized and the behavior expressed. The former is the whole behavior of the thing characterized, hence its reaction under any circumstances. The behavior expressed by a mode represents part or whole of the behavior of the thing: specific reactions under the conditions for which a mode is defined. For example, there are capabilities on a smartphone that are valid when the phone is connected to a Wi-Fi, and that can be characterized by a "Wi-Fi mode", but this mode does not characterize capabilities such as calls, SMS, etc. which are part of the global behavior of the phone. "Wi-Fi mode" could have sub-modes, such as "automatic updates", meaning we can characterize behaviors that are not always those of the phone, but those of its main capabilities.

Behavior under specific conditions is something that we find in UML states. State elements in UML enable to call operations and are used to model a behavior. Regarding the sub-states, we saw they could not be always be considered as state variables. But according to the criteria listed above, they can be modes. A mode is not always defined for the whole time context of the object it characterizes. Individual capabilities can be characterized by more than one mode. They can have their own conditions or events needed to execute them, which can be modeled as preconditions (guards) or with others modes. For example, most capabilities in a train are conditioned by the train operability modes, as what can be done depends on the energy supply and the level of activation. But there are then functional modes conditioning their execution even when they are technically possible, like opening doors or going forward in a train fully activated. Two modes can exist under the same conditions and characterize the same function in different ways. That may be why in the UML definition of state, which we associate to the notion of mode, the conditions are said to be often implicit.

Being in a mode or not can be an information characterized by a state. We can have a set of modes linked by transitions and defined over the whole system life-cycle. But a mode is a notion different from the one of state. A state is evaluated at a point in time, and its value(s) can possibly hold only for an instant. A mode is defined independently of time, as its conditions can hold only for an instant but characterize behaviors that last in time. As a mode is linked to conditions, or a state, it may be why states are sometime defined as values holding for a period of time. We can conclude that while modes and states can be linked, they correspond to two different notions. A state characterizes a way of being of the system at a given time, a mode characterizes the way the system behaves for a particular state value. Modes depend on states, but a state do not depend on a mode to be defined.

A mode can be abstract. It could for example characterize capabilities according to the way the system is used, meaning characterizing a performed behavior, which may not cover all potential behaviors. There are capabilities one wishes to inhibit, authorize or constrain but not as part of the system specifications. For a train, there are restrictions when you drive in a station or in an urban area, representing an information that could be included in a model of the behavior.

4.2.3 Definition

Attribute	*Definition*
Source	The source of the behavior
Capabilities	The capabilities characterized by the mode
Conditions	The conditions for which the mode is defined
Characterization	The way capabilities are characterized by the mode
Level of abstraction	The level of abstraction linked to the capabilities characterized

Table 4.2: Attributes used to define a mode

We define a mode as a characterization of a set of capabilities of a thing under a set of invariant conditions. In other words, a mode participates in the specification but does not specify the behavior by itself. A behavior can be referred to without being fully described. To characterize a mode, we need to know the source of the behavior (the thing that behaves), the kind of characterization, the capabilities and the conditions. In order to adapt this definition to system modeling, we have to consider another attribute: the level of abstraction, which characterizes the capabilities and whether they are attributed to the system, a sub-system or a component. The point of view is always internal, as we characterize the system capabilities. The conditions and characterization could originate from an external source, for example if we define modes of utilization, but the behavior characterized is still the one of the system. According to our previous analysis, the attributes necessary to define a mode are given in Table 4.2.

We then link this definition to concepts enabling to model it using different languages and modeling objects:

- A *capability* is something realized by the thing characterized (e.g., actions, operations or functions).
- A *characterization* is a constraint put on a capability (e.g., enabling, conditioning, inhibiting or calling).
- A *set of conditions* is defined by conditions on a set of states variables.
- A *mode space* is the set of all possible state configurations under which a mode is activated.
- A *mode* is a characterization of a set of capabilities under a set of conditions.

4.2.4 Example

We illustrate our definition using the door example from earlier: we qualify the behavior of a door. We characterize the *capability* of a door to be opened. The *characterization* of this *capability* is enabling it. The *condition* for which the *mode* is defined is that the door is unlocked. The *level of abstraction* is the one of the SOI. At the *condition* that the door is unlocked, we are in a *mode* where the *capability* "open the door" is enabled. Whether the door is locked or not could be evaluated with a *state*, which would not characterize the behavior.

4.3 Application

4.3.1 Definition of train states

Different types of states can be defined, each characterizing a kind of information regarding the system. For a train, we can consider:

- The *operability*: the readiness of the train.
- The *energy supply*: the source used to power the train.
- The *environment*: the place where the train is operated.
- ...

Each type of state can take the value of a corresponding set of *state values*. While they may appear as mere variables, those types of states are not necessarily measured or calculated, as they can express a "known" information, as it is the case for the operability. What truly differentiates a type of state is that its state values change depending on the target it qualifies and the adopted point of view.

Types of state express pieces of information that have been identified and separated to characterize the conditions under which the system is used and where the different use cases can be performed. As such, the information contained in a given type of state can be abstract. Functions that are physically possible for a train, such as opening doors when moving or traveling at fast speed in a train station must be constrained or controlled, using abstract information to represent internal control.

A good example of a type of state used to describe the train system is its *operability*. It is a type of state that indicates the capability of the train to pursue a mission. It mainly depends on the status of the train energy supply and the activation of internal systems. It is used to specify scenarios and represent high-level conditions of a train main functions.

Operability illustrates the fact that defining a type of state is valuable on its own. Indeed, operability is defined before making a design, and most use cases can be performed for several states of operability. It means that, on its own, operability is first an abstract type of state with no practical ways of evaluating it. It is too broad to characterize use cases but still provides a key information to the user regarding the train utilization and evolution.

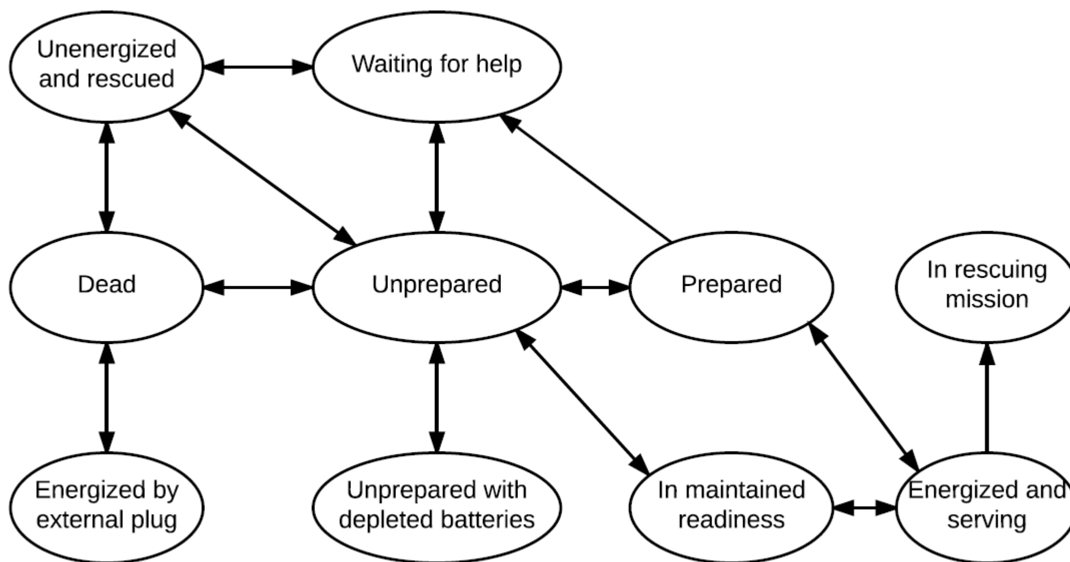

Figure 4.2: Original state diagram characterizing the train operability

We start from pre-existing train state diagram given in Figure 4.2 to highlight the issues that can arise from an unclear concept and definition of state. This diagram is part of a document describing an offer to a client, meaning BT is contractually bound to respect its specifications. The guards on the transitions have been removed for more clarity. We use simple graphical representations that are not linked to a specific tool or language, as such a diagram is specified in documents and is originally not part of BT SysMM.

Operability	
Target	Train
Information	Operability, energy supply, mission
Context	The train daily life-cycle
Abstraction level	Train or component level
View	The point of view of the train

Table 4.3: Original definition of the type of state *operability*

We now apply our definition of state to this diagram, evaluating the attributes we defined using the information in the original operability diagram. The result is shown in Table 4.3.

Operability	
Target	The train system
Information	The train operability
Context	The train daily life-cycle
Abstraction level	Train level
View	Point of view of the train

Table 4.4: Definition of the type of state *operability*

Energy supply	
Target	The train system
Information	The train current energy supply
Context	The train daily life-cycle
Abstraction level	Train level
View	Point of view of the train

Table 4.5: Definition of the type of state *Energy supply*

The states defined in the Figure 4.3 refer to three different kinds of information, as shown in Table 4.3 when trying to define a corresponding type of state. The operability is clearly defined, but this is not always the case for the energy supply. Being in a rescue mission is an unrelated piece of information. The information is not defined over the whole context, meaning that the state space is incomplete. Mentioning batteries means that the diagram characterizes information at component level, while the other information characterizes the system level. This diagram does not define a unique type of state but mixes different information. We hence define two different types of state in Table 4.4 and Table 4.5, each represented in Figure 4.3 and Figure 4.4. One type of state represents the train operability and the other the train energy supply. Dismissing the information regarding the train mission, which was not in the original scope, both states now express a complete, unique information and relate to the same level of abstraction of the SOI. The other attributes do not change.

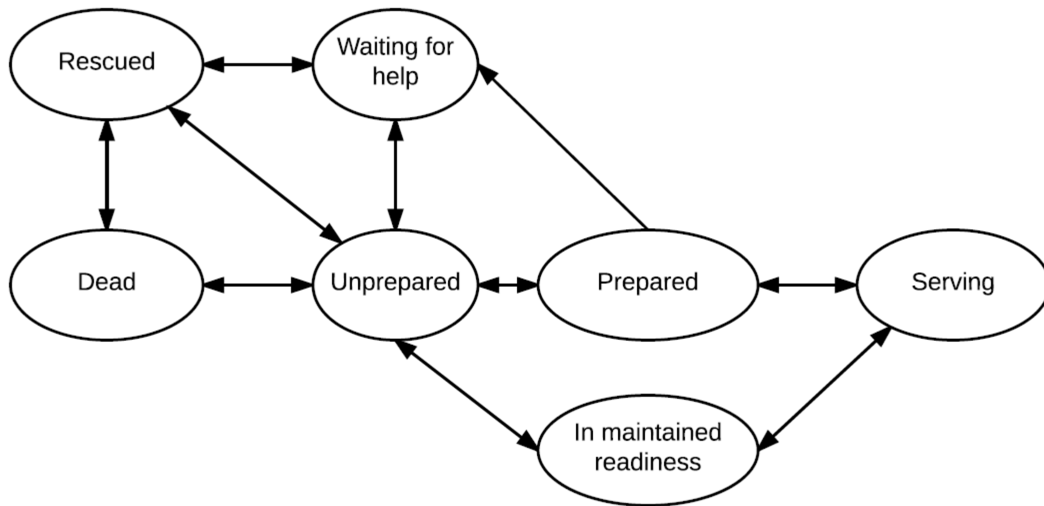

Figure 4.3: Operability state of the train

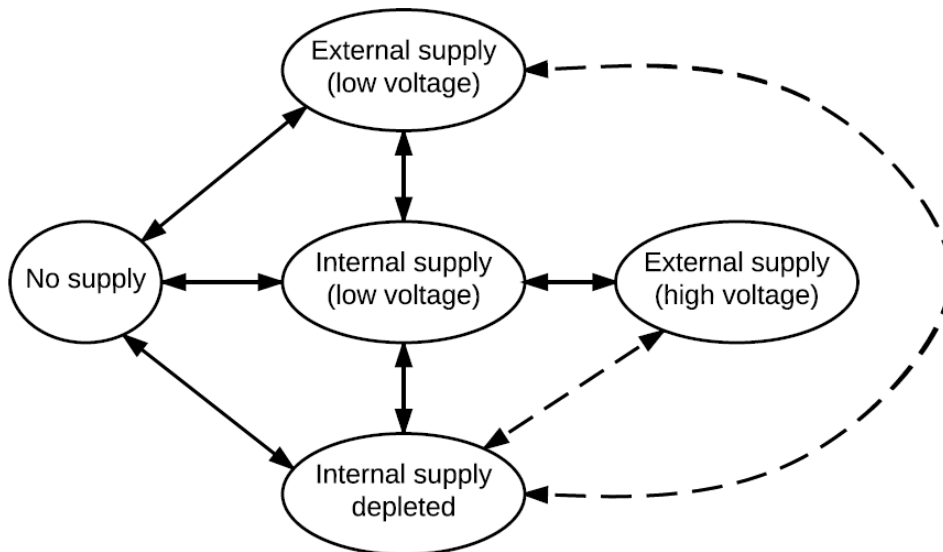

Figure 4.4: Energy supply state of the train

The dotted lines in 4.4 characterize possible transitions that were not considered in the original diagram and became evident after a separate analysis. Note that the operability is only partially determined by the energy supply. The next step would be to define the constraints between the operability and the energy supply, an activity that is covered in chapter 6.

4.3.2 Definition of train modes

The terms of operability state and operability modes are both used in BT, which is an issue. We saw how to define a type of state characterizing the operability of the train. We did not consider operability modes, as operability is an information captured by a type of state. If we were to give it a definition, an operability mode would be a mode that for a condition on the operability state, characterizes a group of use cases or functions. The operability states conditioned, the capabilities characterized, the characterization and even the source of the behavior could be different from one operability mode to another. An operability mode is either activated or not, as the conditions on the operability states are satisfied or not. There would be no natural transition between operability modes, as opposed to operability states.

Operation Mode X	
Source	The train
Capabilities	Power the train
Conditions	The train is in the operability state "dead"
Characterization	Enabling capabilities
Level of abstraction	Train level

Table 4.6: Attributes defining the operability modes

We can still characterize the train behavior depending on the train readiness using operability states. For example, we could define a mode that has for condition the state "dead" and that enables the capability "power the train". Checking the terms used in BT and presented in the chapter 2, it would correspond more to an *operation mode* than an operability mode. We name this example of a mode *Operation Mode X* and evaluate its attributes in Table 4.6.

Whereas a type of state can be identified by the type of information it characterizes, it is more complicated for a mode. A mode makes the link between the train capabilities and the conditions the train is in. Having the train as a source for the behavior and as the level of abstraction considered, both the conditions and the capabilities characterized are needed to identify the mode. Supposing that the characterization of the behavior is always enabling the capabilities considered, we can group those capabilities under the same conditions, in which case the conditions can be used to identify the mode. This is where a type of state can be defined to evaluate whether a mode is activated or not.

Under these circumstances, we can see that state and mode are indeed closely related, as in this specific case they could be identified using a same information. However, they remain different concepts and objects. To avoid this kind of confusion, it is better to identify a mode by the capabilities it characterizes, as we show in the next example.

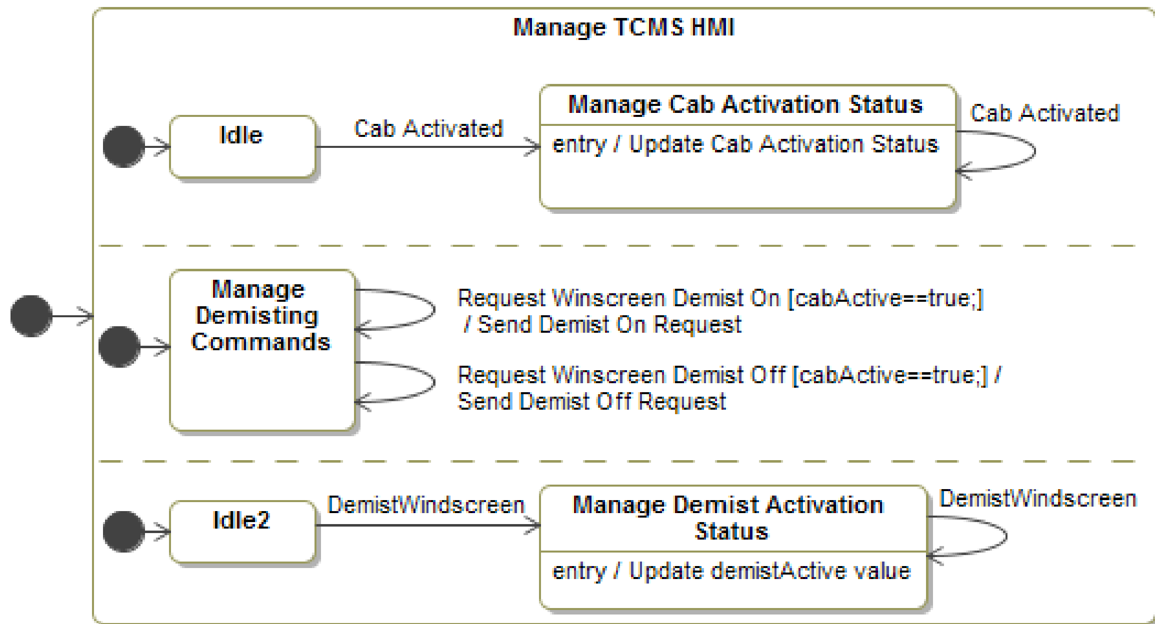

Figure 4.5: Modes of a *TCMS* function

We consider another type of mode, illustrated in Figure 4.5. This is a SysML state machine illustrating the behavior of a *TCMS* function, "Manage the TCMS HMI", the *TCMS* being a subsystem of the *consist* and our SOI here. "Manage the TCMS HMI" is a mode, and is identified by abstracting the capabilities it enables into a higher level capability grouping them. The black circle connected to "Manage TCMS HMI" represent the first of a transition from a state of the system where the conditions for the activation of the mode "Manage the TCMS HMI" have yet to be fulfilled.

Manage the TCMS HMI	
Source	A function managing the system interface
Capabilities	Managing the cab action status Managing the demisting status Managing the demisting requests
Conditions	Unknown/not specified
Characterization	Enabling the capabilities
Level of abstraction	Subsystem

Table 4.7: Attributes of the mode “Manage the TCMS HMI”

We consider the SysML state elements as modes. We define the attributes of the mode “Manage the TCMS HMI” in Table 4.7.

The conditions are not visible in this diagram, they are found in the documentation. The missing information could be represented by expressing conditions on the system's and sub-system's states using variables. The mode characterizes the function's behavior, not the SOI's. Functions are abstract notions, and we prefer to not define states for them aside from modes modeling their behavior, as they are too numerous and can exist in specific time contexts. We also see that each capability is characterized by a different behavior with their own modes. There are information about components that could be abstracted. For a same level of abstraction regarding the SOI, we can have a hierarchy of capabilities with their own behaviors, modes and time contexts. It should also be possible to model such a behavior for the *consist* itself, without referring to lower levels of abstraction regarding the system. We see that beyond describing the expected behavior of the system, modes can be used to describe how it will be implemented.

An improper definition of states and modes can have serious consequences when designing a system. There was a case where BT, as per contract, had to provide a train with troubleshooting functions executable when alimented by batteries at low voltage (under 63V). However, according to the supplier, the battery was considered at low level when under 50 volts, for which no functions could be executed. Confusing the state of energy supply and the state of the energy source supplying it, since both shared a same state value, BT implied it provided troubleshooting functions available for any voltage under 63V, even under 50V where it was no longer possible for physical reasons. Both states and modes should have been identified and specified with a specific meaning, target, etc. at a high level of abstraction, before even choosing batteries as a solution.

4.3.3 Verification of the behavior

Now that we have our concepts and a way to express them in models, we need to perform some integration: we need to correlate the different pieces of information expressed in the different types of states, and define an architecture for the mode so that they represent a single, consistent behavior. We need to define constraints around state and mode, as part of the specifications. Those constraints will define the possible state configurations, as well as the transitions between them, and the conformity to those constraints will be expressed as properties as part of the V&V requirements. We also want to ensure the traceability of the information captured to perform validation of the behavior all along the development process, based on a same high-level behavior that has been specified, modeled and validated. All of this is covered in chapter 6.

Before performing integration and V&V activities, there is a need to acquire the right information, from the right models. States, modes and all modeling element used to specify the system and its behavior must be properly defined and traced. This can be a challenge in an industrial context. The next chapter hence presents a way to enforce modeling rules to properly define and keep track of the information in models created following BT SysMM.

Chapter 5

Model verification method

The previous chapter presented some concepts needed to characterize a system and its behavior. Those concepts are meant to be linked to modeling elements, following a specific semantic. As explained in chapter 2, having an ontology and a modeling method is not enough to ensure that engineers create the same models the same way with the same meaning. There is a need to enforce a semantic regarding the expression of concepts in modeling elements while following a modeling method. In this chapter, we present a solution to express and enforce modeling rules, with the added benefits of ensuring traceability.

This chapter focuses on the *verification* and *validation* (V&V) of system models, built as part of the system development process at Bombardier Transportation (BT) for producing a broad portfolio of railway products. The models to be checked are expressed using SysML, following the BT SysMM method. The main objective is to develop a generic V&V solution based on SysML without any tool dependent criteria so that it is reusable across all BT divisions and projects.

5.1 Models V&V

From a technical point of view, the main aspect in verifying a model is ensuring that no errors were made in the specification of the system design. Creating models and having correct models are two different things, and can impact the rest of the development process. Similarly to validation, the earlier an error is detected, the less the cost.

From an organizational point of view, within large organizations, ensuring that everyone creates models under the same guidelines and constraints is a challenging task. It is crucial that the modeling team members work the same way and are able to exchange

around their delivered models with others without any misunderstanding or consistency issues. This gets more complex with teams spread across continents and/or companies. Having one defined modeling method across an organization and applying it the same way are two different things.

During the early phases of MBSE adoption at BT, the focus on models' V&V was triggered mainly by specific projects based on particular customers or countries needs. As MBSE enabled the reutilization of models specification across projects, the goals of V&V extended towards being more generic and project-independent. This however introduced the discovery of hundreds of errors and inconsistencies by the BT V&V team. It was not that the models were globally wrong, but rather that the project specific teams had their own interpretation of the method or specific modeling practice, gained from experience. What it did mean is that the models could neither be easily reused by other teams, nor could they be adapted while reproducing the same modeling approach. This is a main challenge for large organizations that are driven by project specific customers in contrast with those able to generalize their products and offer a predefined product portfolio (e.g., in automotive). Therefore, the need for reusing V&V of the delivered models is crucial to ensure proper systems models reuse. Moreover, it is crucial to implement the suitable adoption approach, similar to the D3 MBSE Adoption Toolbox [82].

5.2 Background on BT SysMM

BT SysMM tasks include V&V activities to ensure the quality of the deliverables. Those activities are performed through manual review by experts. However, through the deployment of SysMM on several projects, the implementation of V&V solutions started to get very challenging due to the many changes triggered from the various dimensions such as the applications of modeling (e.g., functional description and variant management) and the abstraction levels (e.g., train, consist and subsystems). Therefore, the need for an automatic, generic and reusable V&V solution was addressed to improve the V&V activities and hence optimize the deployment of MBSE. The targeted approach was built on the following objectives:

- Enable formal, generic and reusable V&V methods to be used across different projects and different departments.
- Ensure an early start of the V&V activities with regard to the system models development and keep it running in parallel to the SysMM tasks.
- Support V&V automation as much as possible to reduce the time consumed on V&V activities and avoid any potential errors due to manual actions.

5.3 State of the art

As explained in [83, 71], SysML on its own is not the best suited to apply a development method or build meaningful models in systems engineering. We have to ensure that we manipulate system concepts that are represented by corresponding model elements, along with proper semantic and relationships. A good example would be the lack of elements representing a function, which lead to the creation of specific methods on how to define a functional architecture based on SysML [7].

It is possible to adapt SysML to our needs through the use of profiles, constraints and additional semantic. The Arcadia method [83] is an example of an adaptation of SysML to system development using system concepts. Arcadia is not considered as a Domain Specific Modeling Language (DSML) by its creators because of the broad scope of its application and its links to modeling standards. However, Arcadia does not follow the SysML standard fully, and has fixed concepts linked to the modeling elements.

BT developed a profile that aims to give semantics to SysML elements while following a general modeling method that could be used also for other systems beside trains. From this comes the need for the verification of the models according to the semantics defined in the profile. The difference with Arcadia is that customized semantics and profile can be adapted depending on specific needs and the method used, without relying on a fixed solution and tool. We consider here an existing solution for SysML models V&V and several examples of its application.

System V&V is different from model V&V, and so are the techniques used. Instead of considering common V&V solutions such as tests or model checking, which would require for the model to be executable, we need to check if its construction is consistent and holds correct meaning compared to a real system. As shown in [84, 85], it is possible to have an implementation and verification of a SysML profile through the use of the Object Constraint Language (OCL) [86].

OCL enables to define constraints on a model, which we refer to as *rules*. We speak of verification rules and validation rules depending on their usage, but they are often called validation rules in practice, as shown in the several tools using this mechanism [87, 88, 89].

While OCL is widely used for this purpose, V&V rules can be developed in other languages supported by the modeling or analysis tools. For this reason, we consider the model V&V solution studied here to be the rules' mechanism and the method around them, whether the rules themselves are coded in OCL or some other language. The rules developed for BT sometimes required to use Javascript or Ruby, as there were limitations on how the elements of the meta-model of the modeling tool could be referred to or constrained using OCL.

Regarding the use of OCL to check or analyze a model, we can find several examples of its adaptation to industrial context, offering technical solutions [90]. Some, such as [91], includes OCL as a V&V solution in a process for models and instances design.

We consider the use of verification rules in a broader context, which is a system development process including many kinds of models and taking into account the work of several modeling teams across different projects. We use OCL rules to enforce a semantic and detect errors in the model representing the system. Validation using OCL rules is technically possible but it is currently not practical to develop those in a project context, as it will be explained further in this document.

We use the rules mechanism already existing in Magicdraw. A rule in magicdraw is a constraint put on a given type of modeling element. It is checked on all instances of the modeling element, and return an error and a reference for every occurence where the constraint is violated. A rule is defined by the following attributes:

Constrained element: the modeling element, be it an object or a relationship, affected by the rule.

Severity: the degree of concern regarding the constraint violation detected by the rules (information, warning, error, critical error).

Error message: a textual description of the constraint checked by the rule.

Language: the language in which the script implementing the constraint is written in, such as OCL2.0.

Specification: the script implementing the rule.

An example of a verification rule's attributes in Magicdraw is given in 5.1. The specification is written in OCL 2.0. The rule illustrated here checks that the attribute "documentation", which is a free text field, has been completed.

5.4 BT SysMM V&V

5.4.1 Method Stakeholders

The Figure 5.2 shows the context of SysMM V&V and the roles of its stakeholders. The V&V activities are part of SysMM and embedded within each task of SysMM (e.g., Operational Analysis). They start in parallel and continue until the deliverables of the SysMM task are verified and validated.

Constraint	
Name	2F_OA_UseCase_Has documentation defined
Qualified Name	Validation Rules::01 - Rules Packages Creation::Method::Operational
Specification	self.ownedComment->size() > 0
Constrained Element	«» BT Use Case [UseCase] [BT SysMM::BT Use Cases]
Validation Rule	
Severity	error
Error Message	A Use Case should have a documentation
Abbreviation	UseCase_Documentation

Figure 5.1: Example of a verification rule in Magicdraw

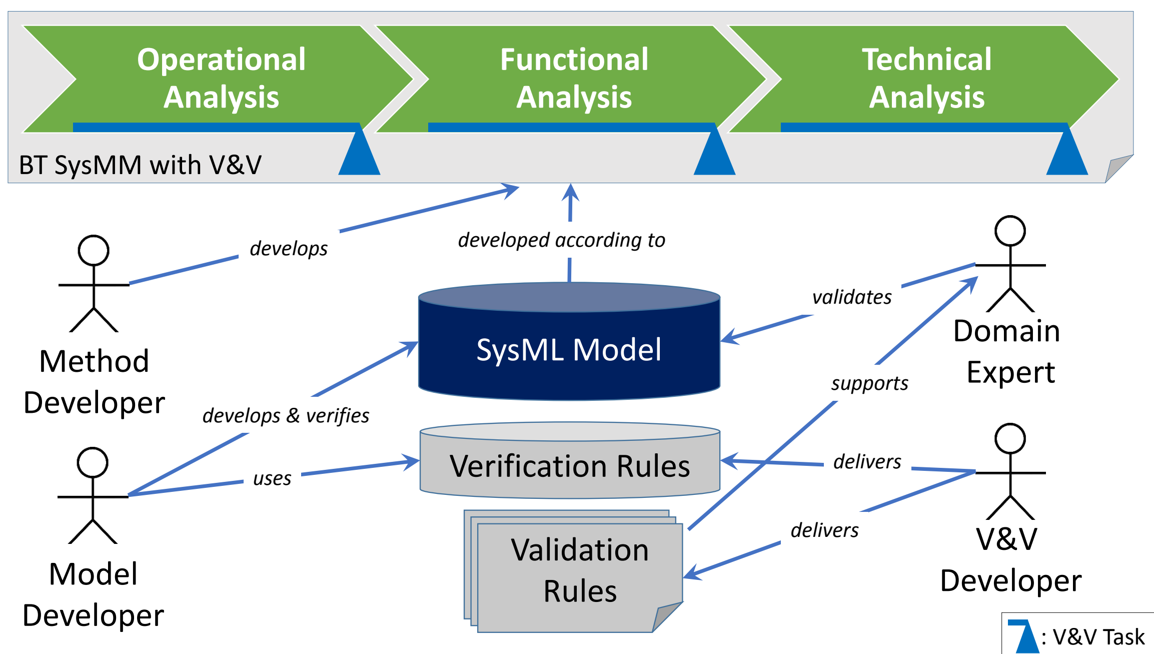

Figure 5.2: BT SysMM Verification and Validation Stakeholders Context

Moreover, there is a common V&V part across all the tasks of SysMM, related to the generic and reusable models (such as the model library elements and glossary).

The context in Figure 5.2 indicates that the SysML model is the system of interest under which the V&V takes place. The verification rules are also represented with a model icon because they are implemented using OCL directly in the systems modeling tool. Both the SysML model and the verification rules are included in a project model, whereas the validation rules are documented in a formal textual format and shared through a common guideline. Validation rules are currently broad and/or abstract con-

siderations that cannot be evaluated by a script. While verification checks the model and its semantic, validation targets the information expressed in the model regarding the system requirements and expectations. We could define lists of validation rules that check specific considerations expressed by domain experts, but quite often, the lack of resources (time and skills) to develop and use such rules during the project is an issue.

Stakeholder	Role Description
Method Developer	Is responsible for defining and developing the system modeling method, its guidelines, training courses and tools' customization specifications. This also includes the V&V method parts and their relationship to other method parts. The method developer possesses a unique governance role in monitoring the deployment of the method on projects to ensure the reusability of delivered system models.
Model Developer	Is a member of the modeling team that is responsible to develop the system models and verify them according to a defined set of verification rules based on project needs. The verification process is done automatically by the system modeling tool and can be set to be active all the time or triggered by the model developers.
V&V Developer	Is responsible to develop and maintain the verification and validation rules based on the input from the method developer, domain experts and project needs. Additionally, this includes analyzing the V&V requirements, implementing, testing and delivering them. It is the role of the V&V developer to ensure the reusability of V&V rules across several projects.
Domain Expert	Is a member of the architects team who possess the authority and knowledge in a particular railway technical domain, e.g., brake, propulsion or train control. The domain expert plays a crucial role in validating the system models' content based on his own experience of the real-world system represented by system models.

Table 5.1: The BT SysMM V&V Stakeholders Roles Description

The SysML model represents an abstraction of the real world system (e.g., train, subsystems or components). Furthermore, the SysML model is being developed based on the defined method and guidelines bundled here with the BT SysMM. The SysMM V&V identifies four stakeholder roles with their own responsibilities and competencies. Table 5.1 lists these four roles and describes them in detail.

It is crucial for these roles to be well-defined in the company, as not everyone should define, develop, apply or change rules implementing the modeling method or defining the conditions that models have to satisfy to be validated. While we can define any arbitrary number of users, the definition of the modeling method and the management of the rules and their packages should be allocated to specific entities. This allows to centralize the skills, development efforts, and rules specifications, while avoiding conflicts and incoherence among the modeling teams.

5.4.2 V&V Method Overview

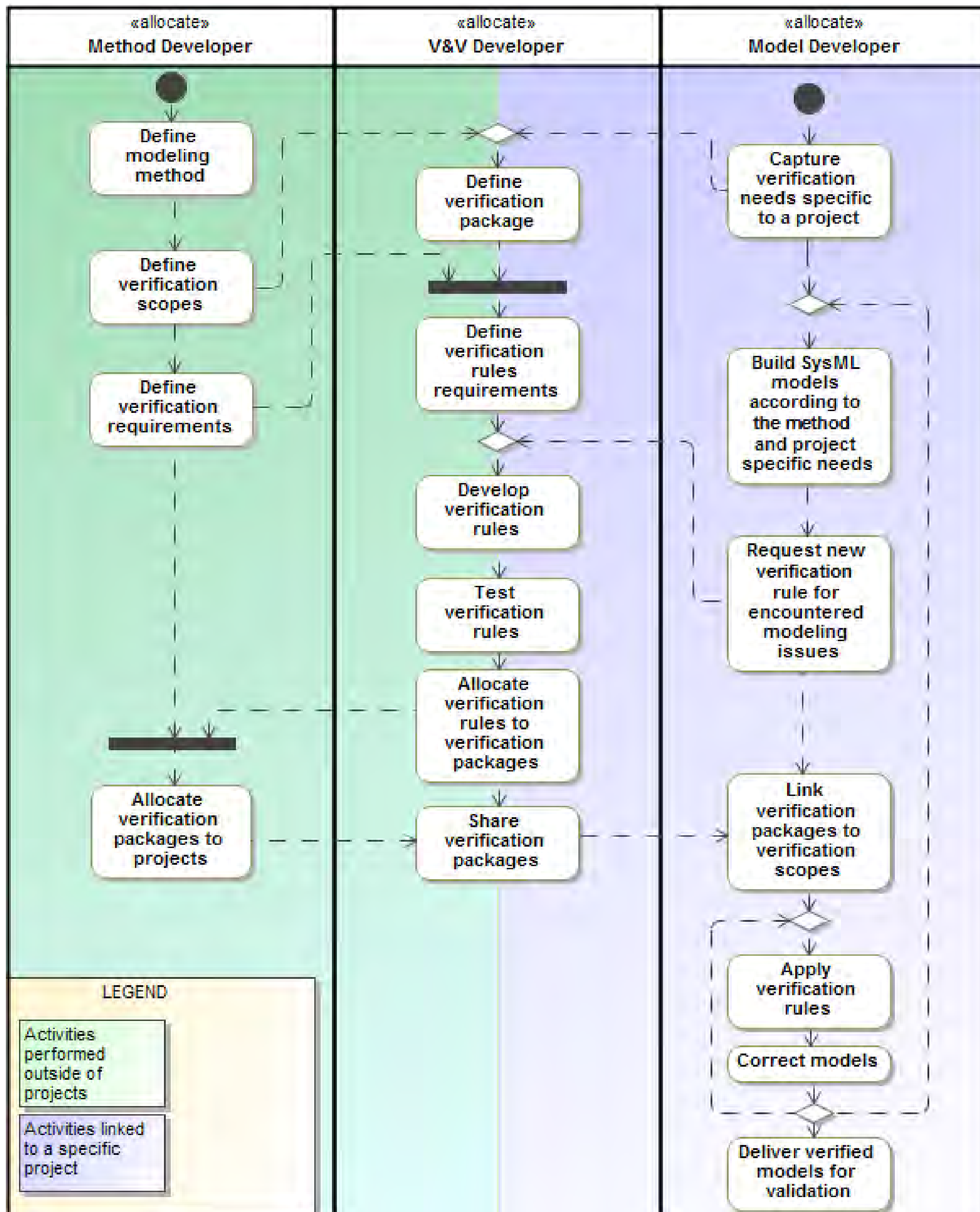

Figure 5.3: Verification Method

As we have defined the different roles in the previous section, we now present the method and work process they follow in order to specify, develop, share and apply verification rules. This is illustrated in Figure 5.3. As validation rules are not yet managed by the rules' mechanism, they are not part of the method presented.

Rules are defined and used for a specific purpose and context. A key aspect of our implemented solution is the allocation of verification rules to modules or packages. This way, rules in a same package can share a same application context and correspond to a same step in the modeling method with the related semantics. As the packages are managed by the method developers, they can be communicated to any team, enabling uniformity and reuse. Rules specific to a project will be contained in their own package. When working on a collaborative tool, the packages can be automatically updated. Choosing the right verification packages enables us to define, apply and adapt our semantics. Packages can be versioned to be able to work on older projects. We can define packages providing the semantics of other modeling methods when working with or for other providers. Packages are to be built so as to separate conflicting rules.

Verification rules are not just a technical solution, they are specifications on how the modelers should work and what they should deliver. In order to specify, communicate, understand and use the rules, a proper documentation is required. Supposing you work with teams with different tools or an external provider, you can communicate the rules that have to be followed during modeling, even if they are not implemented or compatible with the tools. The documentation should at least specify for each rule: an ID, a target, a method, its current place in the life-cycle and the specification/constraint/error addressed by the rule.

5.4.3 Benefits

Aside from the semantics, the rules enforce the (modeling) methods and support the engineers' work processes. Checking the rules on each step results in a report on the quality and level of advancement in the work done, enabling to proceed to the next development step after having checked for errors. Note that by verifying the relationships between concepts/elements, we ensure a certain degree of traceability. Supposing that we have modeled the requirements as artifacts, we can achieve part of the system validation just by ensuring that they are linked to other elements such as functions or scenarios. This is also true across abstraction levels, when switching the SOI from the system to a sub-system.

An advantage of the approach based on verification rules is that it is progressive, empiric, iterative and adaptive. We can specify, update and change the semantics and modeling rules over time. Note that most verification rules should be decided at the start of a project. While we can always develop rules during a project in answer to an immediate need, we should not remove or change any of them once the modeling activities have started. A key point in BT is that new modeling methods are being developed and spread in the different company sites across the world. With rules, they are supported by a common and automatic solution. Rules are transmitted as packages, and executed automatically, generating a report in natural language and links to elements conflicting with the rules so that engineers find and correct errors. Modeling teams can check the models and learn at the same time the method implemented by the rules. They also provide a feedback and request new rules. Rules support the training of modeling teams as the rules enforce the way the method has to be applied. In return, the method developers learn from the experience of modeling teams. This creates a dynamics that optimizes the work performed and the results obtained across projects, each supplying new rules and improvements. This would not be possible if we were to impose a new tool with a fixed semantics.

5.5 Use case example

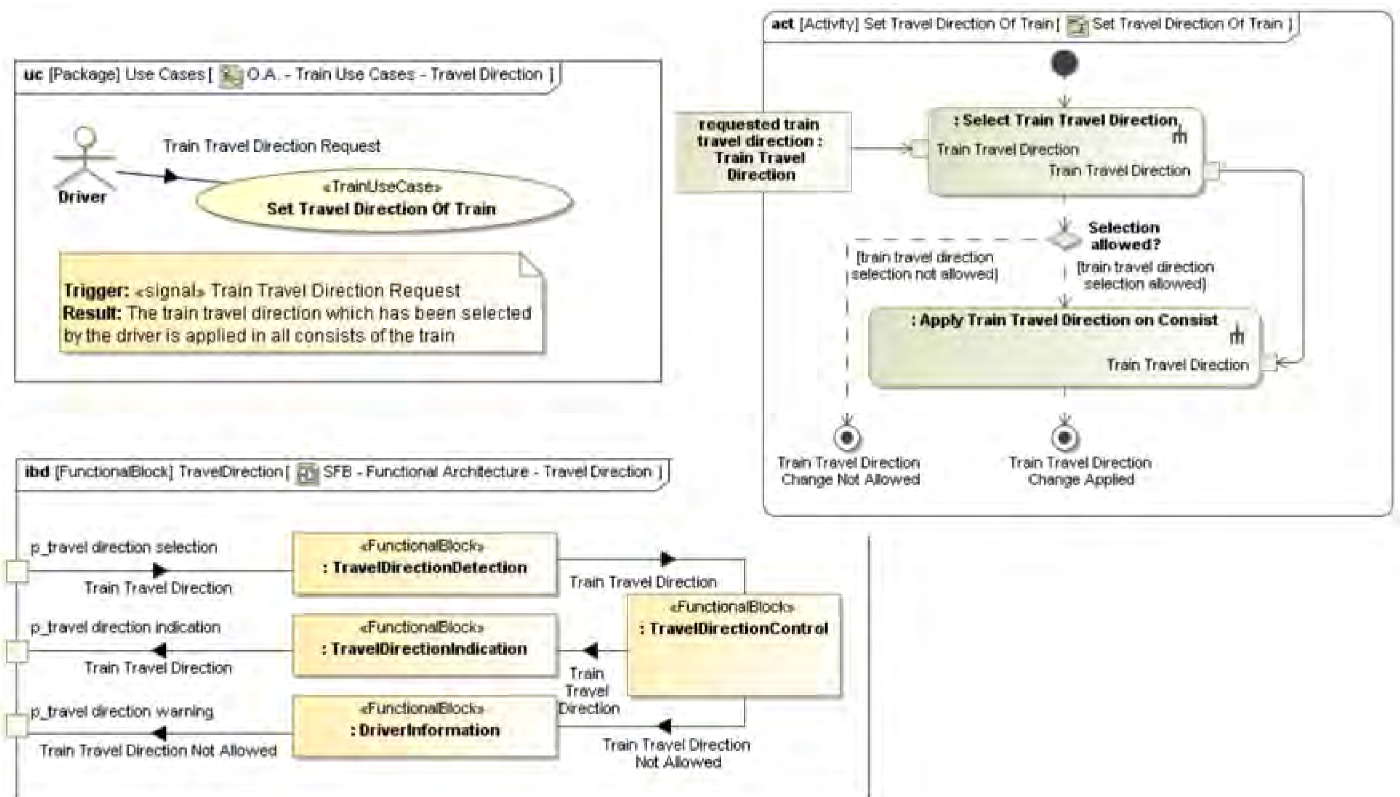

Figure 5.4: BT SysMM Diagrams Example on Which V&V is Applied [5]

Traditionally, the work split during the model development between teams is based on the work breakdown structure which defines a list of scopes covering all functionalities of the SOI (e.g., train or subsystem). The functional scope *travel direction*, taken from [5], is used in this section to illustrate the application of SysMM V&V on an example from the railway domain. A scope here is referred to as a part of the work breakdown structure of the whole function set.

Figure 5.4 shows some of the SysML diagrams delivered using the SysMM operational and functional analysis tasks.

The operational analysis part is demonstrated through the use case and activity diagrams. The use case diagram defines the use case *"Set Travel Direction Of Train"*, its actor (i.e., the driver) and the respective trigger signal *"Train Travel Direction Request"*. The activity diagram describes the internal behavior of this use case in a generic manner independent of any specific functional or technical solution in order to reuse it in several projects.

The functional analysis part is shown with the internal block diagram where previously modeled activities are structured in a functional architecture that fits a particular train platform or project. The functional architecture defines all functions needed to cover the *travel direction* scope with functional blocks and their interfaces. These function blocks are linked back to the activities of the operational analysis and allocated later on to the technical blocks' solution to ensure traceability.

The scope discussed here is one out of other hundreds of scopes normally modeled to describe the safety related functions of a train. Usually, a set of scopes is assigned to particular domain experts and model development team. The model developers, usually system engineers, take the responsibility to develop the SysML models based on the input requirements of their own scope.

Sample Verification Rules:
1. A use case must own at least one activity 2. A use case name must follow the naming convention guidelines (e.g., starting with a verb and all words are capitalized) 3. A triggered use case must have at least one actor and one trigger signal 4. A signal name must follow the naming convention guidelines 5. Model elements, e.g., use cases, must be unique across the whole model 6. Each function is linked to at least one activity
Sample Validation Rules:
1. Are the use cases' actors complete according to the requirements? 2. Are all actors and signals considered in the correct way with respect to the requirements linked to the use case? 3. Does the use case activity describe the exact scenario of real operation as described in the requirements? 4. Is the functional architecture solution (i.e., functional split and allocation) satisfying the relevant requirements?

Table 5.2: BT SysMM Rules Examples

During the modeling activities, the model developers *verify* their models automatically based on the verification rules implemented in the tool. These rules are aligned with the deployed method and implemented using OCL in the systems modeling tool. Table 5.2 lists a sample of textual representation of the verification rules for model elements such as use cases or signals. The verification rules check automatically if model elements are modeled according to the defined method. If not, the model developer is getting a notice about the result of the check i.e., an error, warning or information. One can see from the list that the verification rules check also model consistency and completion.

After the model is verified, it is shared with the responsible domain expert for the sake of *validation*. The second part of Table 5.2 lists a sample of the validation rules relevant to the presented example. A more detailed sample of the rules deployed in BT is given in the annexes. These rules are documented in a formalized textual format and offered to support the domain expert during his model validation activities. The validation rules are always traced back to the system requirements. It is the role of the domain expert to apply his experience in order to check these traceability links and confirm that the model specification is valid with respect to the provided requirements.

Chapter 6

Behavior verification method and model

In the previous chapters, we developed concepts to specify and integrate a system and its behavior, as a well as a mean to check the expression and utilization of those concepts through semantics associated to a SysML profile. We shall now consider how to perform verification and validation of a system behavior.

In this chapter, we present early validation results obtained thanks to the definition of global states and modes describing a train and its behavior at operational level. This work is conducted in the scope of a project in BT and is applied to a case study. We aim at establishing a continuous validation method along a train system development process. The target of the validation is the system behavior, which we define as the way the system reacts under given circumstances.

6.1 Presentation

6.1.1 Context

The behavior of a train system is defined through hundreds of use cases, classified among hundreds of *scopes*. A *scope* is part of a functional breakdown structure that classifies the use cases and the functions according to their domain (e.g. energy, traction, etc.). The scopes are divided among different requirements and functional engineers. In the chosen metro *MOVIA Maxx* case study, the specification at operational level includes 277 use cases contained in 60 root scopes of a classification system, divided among 12 functional and requirements engineers.

Each use case is described by a sequence diagram. Redundancy in specification is avoided by having each engineer work on dedicated scopes. The drawback is that they define behaviors separately, using a non-formalized nor centralized knowledge regarding the system. Consequently, the specifications are unrelated, without any integration. Rather than just specifying what the system-to-be does, we have to specify "what" system we want to obtain [18], meaning an abstract model of the system induced by the specifications. Such a model implements an integrated behavior by conditioning the use cases and track their effect on the evolution of the system state, without specifying how those use cases are realized inside the system.

Regarding the V&V activities, there are limitations in BT. Executable models such as grafcet [92] have been used where parts were missing in the co-simulation, but they correspond to designs provided by subsystems developers or external providers. There is currently no solution in BT nor models to check the system before any implementation.

It is currently possible to check that the system does what it is expected to do through tests, later in the process, using co-simulations or bench tests. However, there are no practical solutions for capturing unknown or unwanted behavior. While it is possible to generate random inputs in a co-simulation, all cases disproving a property have to be analyzed by an engineer to assess its relevance. BT stopped using such a solution, as experience showed that checking a property could result in hundreds or thousands of cases to analyze, most of them irrelevant as they suppose a use of the train system that cannot happen in reality. There is a need to constrain either the system behavior or its inputs.

6.1.2 Issues

Each functional engineer has knowledge on a specific scope of the train, and uses informal information from textual requirements and her own experience to make specifications. We saw that there were no integration specified through the models. Instead, the review of the whole system is done informally by a skilled individual. A lack of a formal description of the system at specification level prevents its automatic verification [18]. There is a need to express and check a common information regarding the whole system, which we partially covered in the previous chapter.

The solution should automatically verify information consistency, so that engineers do no depend on the validation team for their specifications. On the other hand, the validation team should receive integrated, formalized and verified information to build an executable model, rather than interpreting it on their own.

There are potential unknown and unwanted aspects of the system due to the phenomenon of emergence. Integrating the system implies specifying the integration to avoid emergence. Specifying the integration is the role of the functional engineers, not of the validation team, which is currently the one doing the integration when asked for a model of the system.

The goal of the proposed approach is not to give an optimized solution to the V&V of the specification at a given level of development, or even globally, but to provide a way to continuously conduct these activities along the development process and with traceability of both V&V requirements, results and models, based on an existing modeling method. Accessibility, simplicity and quickness is preferred over exhaustivity and formalism, taking into account the industry needs and capabilities. A given solution cannot be immediately implemented across the entire development process. It has to be gradual, following the evolution of both the modeling method and the development process.

6.1.3 Related works

State machines have long been used to define and check system behaviors in discrete systems [78, 93, 75].

The solution proposed aims to check a high-level specification of the system behavior. The chosen approach to build a model is similar to *state analysis* [14], in the sense that "states" of the system are modeled separately and used to control the system behavior. The difference being that in [14] the model is more detailed, specifying a control of hardware. Ingham developed this approach in response to several issues, similar to those encountered in BT:

- Subsystem-level functional decomposition fails to express the whole system behavior.
- There is a gap between the requirements and their implementation.
- The system behavior is not explicitly specified.

6.1.4 Method

In the solution, the system is modeled while separating its description from its behavior. The description of the system relates here to its structure and properties, and more generally what is known regarding the system at a point in time. The system description is modeled using *states*, while the behavior is modeled using *modes*. We use the concepts of *states* and *modes* defined in the chapter 4. The method developed is divided into three parts:

- Modeling method.
- Verification method.
- Creation of the execution model.

6.1.5 Case study

The method has been experimented on a real project data, using high level specifications of a *MOVIA Maxx* metro. The focus was put on the train main functions when operating under normal conditions, excluding maintenance, emergencies, restricted and degraded operations and secondary use cases (e.g., comfort features) to concentrate on the train activation and driving operations. This resulted in the selection of 54 use cases divided into 12 scopes. Following the presented method, data was captured or specified in order to build an integrated model of the system and check its behavior.

6.2 Behavior description through states

6.2.1 States in the case study

Movement	Neutral Section	Electrical Supply
Moving	Neutral	Full internal supply
Standstill	Not neutral	Internal supply depleted
		Line supply
		No supply
		Partial internal supply
		Shore supply

Table 6.1: Example of types of state describing a train operational status

Fifteen types of states were established to describe the system operational condition and situation for the selected use cases. A sample of them with their possible values is shown in Table 6.1, with information regarding whether the train is moving, if it is on a section with an electrical line (not neutral) and which source of energy is supplying the train systems. The *State space* of each of the 15 types of states varies from 2 to 6 different values.

Train states characterize the train at its own level of granularity, from its own point of view, in the context of its whole life-cycle. This information can be found in part in the scenarios defined in the operability analysis, as they specify circumstances under which use cases are performed. Other information are obtained through empirical experience and requirements analysis. This information, independently of the behavior, can be constrained by physical laws, properties to be respected, inter-dependencies, etc.

6.2.2 State constraints

Without a design and working at system level, there may be an issue in measuring or evaluating some of the states. What can be done is defining possible state configurations. To do so, the different types of states can be correlated by constraints and properties that condition the values (states) they can or should take in regard of each other. Those constraints are defined in relation to the system, and not to its functions. Consequently, establishing a correlation between them through constraints results in an integrated description of the system that will be navigated through its behavior.

The system behavior depends in parts on its circumstances, meaning its situation in relation to a context. They can be described by its states. The evaluation of all types of state is a configuration describing the train circumstances.

In order to check the system behavior, it is important to define as many relevant constraints as possible on the state values forming configurations to reduce emergence. The more the system is constrained, the less unknown behaviors there will be, and the fewer cases there will be to consider. While over-constraining the system is a risk, it is not an issue in our setting: since checking the expected behavior is possible, any issue due to over-constraining can be detected early enough. The problem can then be solved, or in the worst case the specifications or the constraints were initially wrong or cannot be fulfilled or checked at this point. On the other hand, a lack of constraints will result in more unknown cases and will present the risk to perform analysis on irrelevant cases while overlooking errors in others.

Types of states can be used to check properties of the system. Definition of such properties can lead to the creation of more types of states or constraints. For example, a train "visibility" is ensured when the train has its lights systems activated under the right circumstances. It can be defined as a constraint on the other types of states to ensure that the train would be evaluated as "visible" when the circumstances ask for it. Such constraints come from requirements and knowledge regarding the expected system. They can constitute formal validation requirements when expressed using states.

We define two types of constraints: *simple constraints* and *complex constraints*. Simple constraints are defined between *pairs* of states values and can be captured and specified by engineers. All possible simple constraints are considered, leading to new specifications and a first integration of the system states. Complex constraints represent known or desired constraints between three or more states values and cannot be exhaustively captured.

Let us denote $T = \{t_1, ..., t_n\}$ as the set of the types of states, with n the number of types of states defined. Each type of state corresponds to a set of possible state values: $\forall i \in \{1, ..., n\}, \exists k \mid t_i = \{s_{i1}, ..., s_{ik}\}$. For every state value s, we also denote by s the logical proposition: "the system is in state s".

Two incompatibles states values x, y of two different types of states t_i, t_j are represented by the simple constraint $\neg(x \wedge y)$. For all types of states, we can define simple constraints as a set of clauses $simpleConsts$ such as:

$$simpleConsts \subseteq \{\overline{x} \vee \overline{y} | \forall i, j \in \{1, ..., n\}, i \neq j, \forall x, y \in t_i \times t_j\} \tag{6.1}$$

The complex constraints are defined by forbidding combinations of state values taken from subsets of three or more types of states. Considering a group of types of states $t_1, ..., t_k$ with $k >= 3$, a complex constraint $compConst$ can be defined as all combination of state values among the subsets $t'_1, ..., t'_k$ such that $t'_i \subseteq t_i$

$$compConst = \wedge_{x_1, ..., x_k \in t'_1, ..., t'_k} (\overline{x}_1 \vee ... \vee \overline{x}_k) \tag{6.2}$$

6.2.3 State constraints in the case study

	Full internal supply	Internal supply depleted	Line supply	No supply	Partial internal supply	Shore supply
neutral	1	1	0	1	1	1
not neutral	1	1	1	0	0	0
Moving	1	0	1	0	0	0
Standstill	1	1	1	1	1	1

Table 6.2: Simple constraints between types of states values

Listing the values of the different types of states, a square matrix can be created where engineers can specify simple constraints between states values. Compatible pairs of states values of two different types of states are marked by a 1 in the matrix, and by a 0 otherwise. An example from the case study is given in Table 6.2, using values from the types of states presented in Table 6.1. Only part of the square matrix is presented.

	Full internal supply	Internal supply depleted	Line supply	No supply	Partial internal supply	Shore supply	Depot	Insertion line	Main line	Station	Neutral section	Not a neutral section
C1	0	1	1	1	0	0	0	1	1	1	1	0

Table 6.3: Example of a complex constraint

Complex constraints are defined as the rows of another matrix. The columns of this matrix correspond to the states values of each type of state. Each row specifies the subsets of state values involved in the constraint, represented by the indicator function (i.e, a 1 means the state is included in the subset). This list has not the ambition of being exhaustive, only expressing known properties from requirements and experience. Contrary to simple constraints, it is not practical, or even possible, to ask for engineers to think of all possible complex constraints. It also presents the risk to repeat complex constraints already induced by simple ones. Besides, specifying the simple constraints and correcting them often leads to the definition of new complex ones.

Correcting a simple constraint means deleting it, as it was too strict and blocked the realization of use cases, and replacing it by a complex constraint that is more specific and carries the actual intent of the initial simple constraint. They often would not have been specified or thought of by other ways. An example of complex constraint C1 is given in Table 6.3.

6.2.4 Use case pre-conditions

States capture the circumstances in which a use case is possible. The preconditions should capture every possible configuration in which a use case is possible, the limitations being expressed through the constraints. A precondition has a subset of authorized state values for each type of state. Considering the subsets $t'_1, ..., t'_n$ of the types of states T for a given precondition *precond*, we have:

$$precond = \wedge_{i \in 1,..,n}(\vee_{s \in t'_i} s) \tag{6.3}$$

Use cases preconditions are defined in a matrix indicating which values of each type of states are compatible with their realizations. Compatible values are marked with a one, incompatible ones with a zero. Those preconditions indicate which values can and should be found in a configuration satisfying the use case preconditions, but do not imply that all combinations of compatible values are possible, as there are constraints to consider. Considering only preconditions of this form is justified by the fact that engineers can focus on the use cases preconditions one state at a time.

6.2.5 Use case pre-conditions in the case study

	Full internal supply	Partial internal supply	Neutral	Not neutral	Moving	Standstill
Wake up train	1	1	1	1	0	1

Table 6.4: Example of a use case precondition

An example is given in Table 6.4 (only authorized values have been displayed for the energy supply). Initially, waking up the train following a scenario to put it into service was not possible as a constraint indicated that a train could not be still on a neutral section. A neutral section is a section where there is no electrical supply from the line, which is the case where the train is parked. The reason for this error was that engineers made the specification while thinking of the train as performing a mission on the main line. A neutral section can be found on the main line, in which case a train should indeed not stop, but a train in a depot is also technically on a neutral section, but still needs to move on its own. This led to the definition of types of states expressing that the train was in a mission or not, and what its environment is, as well as expressing complex constraints to enforce what was intended in the original specification. The initial specifications on their own were either incomplete or not-binding, letting developers of subsystems interpret the information and complete it. Such a completion is not their responsibility and the interpretation can vary between the different teams and providers, creating inconsistencies.

6.3 Verification method

In order to develop an integrated model to verify and validate the system's expected behavior, it is necessary to first have proper inputs. To that end, a solution has been developed for engineers to check some predefined properties of their specifications. The solution is automatic and works like a black box: it is a script coded in R language that takes directly the matrices defined previously as inputs, without a need for other modeling activities. The technical details are presented first, the results and errors detected being discussed after.

6.3.1 State constraints verification

Two basic sanity checks are performed:

1. There is at least one compatible value between two state types.
2. Each state value appears in at least one possible state configuration.

Performing the first sanity check implies checking that the simple constraint matrix is correctly filled.

Calculating possible configuration is done by a script using applications of the graph theory [94]. The solution is intended to correspond to the industrial practice and needs, and as such is not optimal. A more elaborate solution is currently not needed considering that the calculation only take seconds.

The script performs the following actions, logging errors at each check step:

- Check that the matrix is correctly filled.
- Calculate all possible configurations according to simple constraints.
- Filter possible configurations using complex constraints.
- Check the presence of each state value in at least one of configuration of the filtered list.

6.3.2 Use case preconditions verification

For every use case we check that:

- Its precondition admits at least one possible configuration regarding the state constraints
- Each authorized state value appears in at least one possible state configuration.

The script performs the following actions, logging errors at each check step and for each use case:

- Calculate possible configurations.
- Filter the configurations with complex constraints. Check again for the existence of a solution.
- Check the presence of each authorized state value in a least one of configuration of the filtered list.

Simple constraints and complex constraints are applied and checked separately to facilitate their analysis and correction.

6.3.3 Results

The results can easily be formatted and processed, in our case an Excel file. The case study showed that:

- Out of 15 types of states, 2 pairs initially lacked authorized values between them.
- Out of 45 state values, 6 were not initially included in any possible configurations.
- 5 more complex constraints were defined after correcting the simple constraints.
- Out of 54 use cases, 7 initially lacked at least one authorized values for some the types of states.
- 13 use cases did not initially admit a single configuration as a solution.
- 51 use cases had unused values, for a total of 148 cases.

The lack of authorized values between types of states or in preconditions were simple omissions. State values not included in any possible configurations were due to the following errors:

- State were ill-interpreted by the engineers.
- Engineers adopted a point of view that was too narrow, overlooking specific cases were some states values were compatible.

Nearly all use cases had unused state values, meaning state values authorized in the preconditions but not present in any of the related possible configurations. In order of increasing severity, it could mean that:

- A given state was deemed possible in the preconditions but was not.
- The use case should have admitted a configuration with this state but its preconditions were too narrow.
- There was an issue in the way the constraints were defined, blocking possible configurations.

The states are used for the preconditions of all use cases, and their constraints are used in the calculations of all possible state configurations. An error in their definition is where it has the most severe impact.

The method proved that when integrating specified information on current validated steps of a project, there were in fact many errors and misunderstandings that would have to be corrected later on. Those errors were detected here at an earlier stage in the process. In addition, this analysis provides new or proper specifications as opposed to partial or informal ones.

6.4 Execution model

According to our approach, the fundamental unit for organizing the system description is the state, and the fundamental unit for organizing the behavior is the mode. The behavior is modeled by hierarchical state machines, here SysML statecharts, where "state" modeling elements correspond to our concept of mode. Each mode can be activated after checking that the system state configuration allows it and that the right sequence of activities has been executed. A mode here characterizes use cases of the system during which specific conditions on the system state are true.

6.4.1 Holonic structure for states

Types of states are modeled as finite state machines arranged in a structure of holons, similar to what is presented in [70]. A holon is an element that is both a whole, something that can exist and function independently, and a part, meaning it can be connected to other element as part of a structure. Here, each finite state machine is a holon. Rather than representing the system behavior, the holonic structure is used to established traceability between states, some being deduced from others. This allows a first form of integration by providing an evolving description of the system using correlated information.

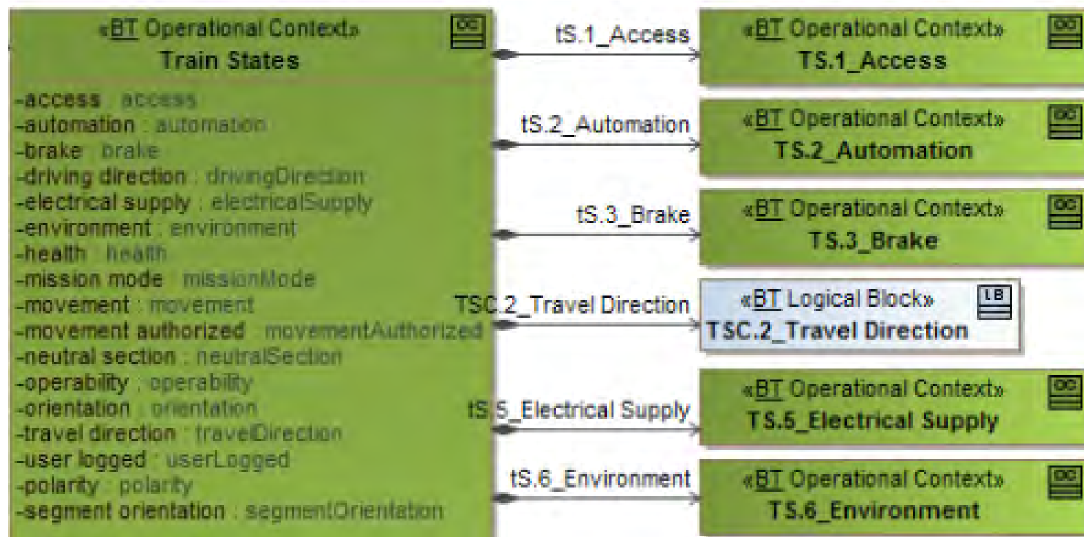

Figure 6.1: Traceability of information for the train types of states

The system description is modeled following a specific process. First, we create a SysML block for each type of states. Each of these blocks are contained under a block gathering the different types of states of the system element characterized. This is illustrated in Figure 6.1.

Each type of state's block corresponds to a SysML property that is traced to the train states, the train element and its behavior. Those properties are updated in the types of state's blocks and used in the behavior (to evaluate modes) and the system (to evaluate constraints and properties). A given property is then repeated in each system block with its values binded to the instance in a type of states where it is evaluated.

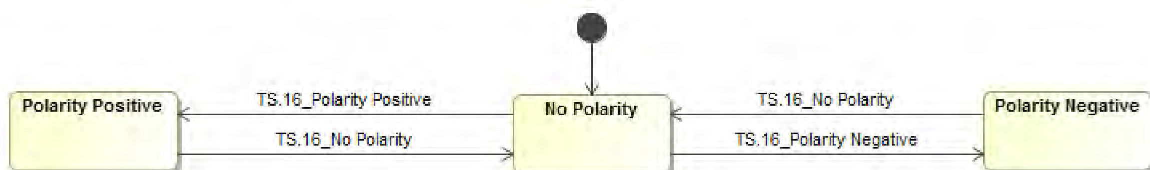

Figure 6.2: State machine specifying the evolution of the type of state "Polarity"

We model each type of state as a non-hierarchical statechart inside each block. They are used to specify the possible transitions from one statue value to another, and update the property corresponding to the current value of the type of state. We create a signal for every transition of state values. An example of such a statechart is given in Figure 6.2

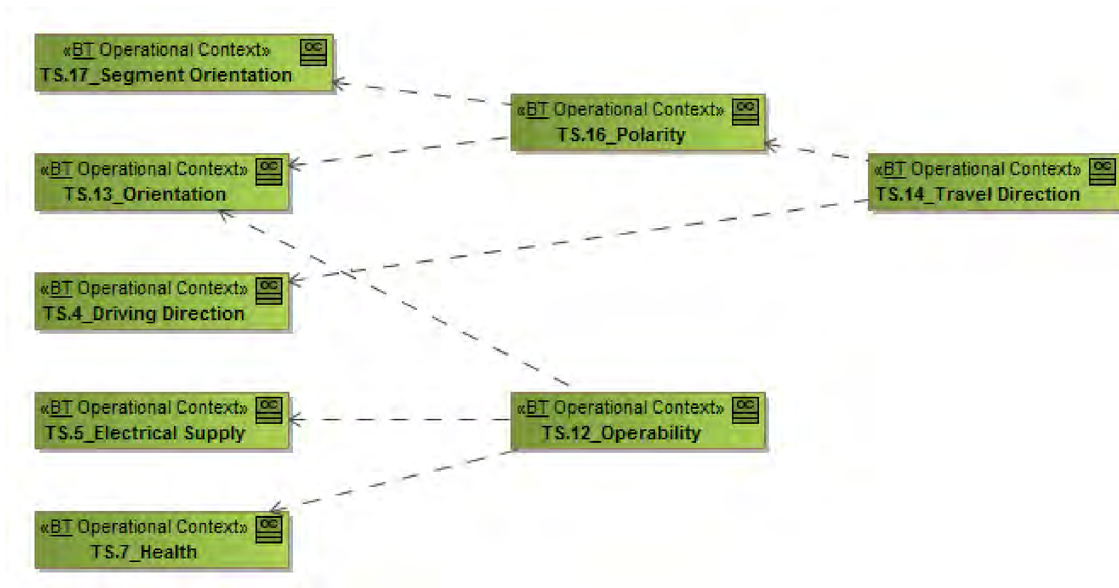

Figure 6.3: Dependencies between train types of states

The value of each type of state evolves depending on signal inputs. Those inputs can come from the current system element environment, its own behavior, or be deduced from the value of others types of states. As the Figure 6.3 shows, there can be dependencies between the different types of states.

The values of types of state depending on others can be updated at the same time the value of the types of state it depends on are evaluated. This result in the creation of state machines inside "control" blocks that can replace individual types of state blocks and state machines.

Internal inputs of a system element, from its behavior to its states, will generally directly change the value of some types of states. External inputs probably carry information from others system elements or the environment and may have to be interpreted through others control state machines. For example, the train energy supply will depend on the main source of electrical current activated, so it has to interpret states of sockets, batteries, etc. which is information from a lower level of abstraction and does not characterize the train. We define inputs ports for signals updating state values around state blocks.

Although the issue was not encountered in the case study, we anticipate that the amount or kind of information needed to deduce the value of a given type of states could result in impractical or too complex control state machines. In such a case, we would define constraints and equations to calculate the value of such types of states, similarly to how we evaluate the activation status of modes, as presented next.

6.4.2 Structure of the behavior

There is a need for both a specification and an executable model of the whole system and its behavior. In order to integrate the specifications, it should be possible to specify dynamic aspects of the use cases. It requires knowing which use case can be realized in a given situation and how their possible realization evolves. Conditions enabling the realization of a use case correspond to our definition of a mode. The preconditions defined earlier enable to know when a use case can be performed, and considered as a basis for defining modes. We will now define a structure of modes to analyze, integrate and model the behavior. We define several types of modes:

- Use case mode: conditions the realization of a given use case
- Scope mode: a mode defined by a precondition composed of subsets that are the union of authorized values in all preconditions of all use cases modes under the corresponding scope.
- Abstract mode: conditions the activation of one or several scope modes.

The way the scope modes and abstract modes are defined is potentially larger than the disjunction of use case preconditions and scope mode conditions they refer to. The intent is to cover a broader context and follow the evolution of use case transition. This is also a way of ensuring that we have implication relationships between the different modes, something we need to build a structure around them. If every mode is implied by another expect for one, which is the root of the resulting implication tree, then we have a single structure of mode that can be explored by evaluating conditions that are more and more specific. Each condition is only evaluated once when evaluating the current active modes. If a mode is inactive, so are all of those that imply it, avoiding evaluating their conditions.

Use case modes directly condition the execution of use cases. Other types of modes only condition them indirectly by conditioning the use case modes or the modes containing those. A mode is defined by the use cases it characterizes (directly or not) and the conditions in which it is activated. As the relationship between modes and use cases is established, the main characteristic needed for the definition of new modes is the conditions for which they are activated.

The conditions for a mode can be considered as a group of subsets of each type of state's values. As long as all state values of a state configuration are part of these subsets, the mode is active. Given $T = \{t_1, ..., t_n\}$ the set of all types of states, a mode has the same structure as a UC precondition (see section 2.5) and is defined by a set of subsets $t'_i \subseteq t_i$.

The behavior can be modeled using statecharts. As the use cases are managed in scopes allocated to different engineers and that they are too numerous to be put in one statechart, all use cases modes should be put under global modes corresponding to their scope, where they are to be modeled in a corresponding statechart. Considering $\{t'_1, ..., t'_n\}$ the conditions of a scope mode, we define the conditions of the k use cases under this scope as $\forall j \in \{1, ..., k\}, \{t'_{j1}, ..., t'_{jn}\}$. We have:

$$\forall i \in \{1, ..., n\}, t'_i = \cup_{j=1}^{k} t'_{ji} \tag{6.4}$$

In order to integrate the statecharts defined in each scope, there needs to be a way to evaluate whether the different scopes modes are activated or not. We propose to create a structure of implications enabling to determine activated modes by evaluating their conditions.

Satisfying the conditions of a use case mode means the conditions of its scope mode are satisfied: activating a use case mode implies activating its scope mode. In the same way, some scope mode could imply others, which is the basis for our implication structure. Some scope modes could also have the same conditions, in which case we create one statechart in each scope to specify the behavior but only define one corresponding scope mode.

All scopes modes may not be linked by implication relationships, in which case we define abstract modes. Abstract modes are obtained by the union of two modes preconditions. We only define abstract modes for pairs of scope modes that do not imply others.

6.4.3 Structure of modes in the case study

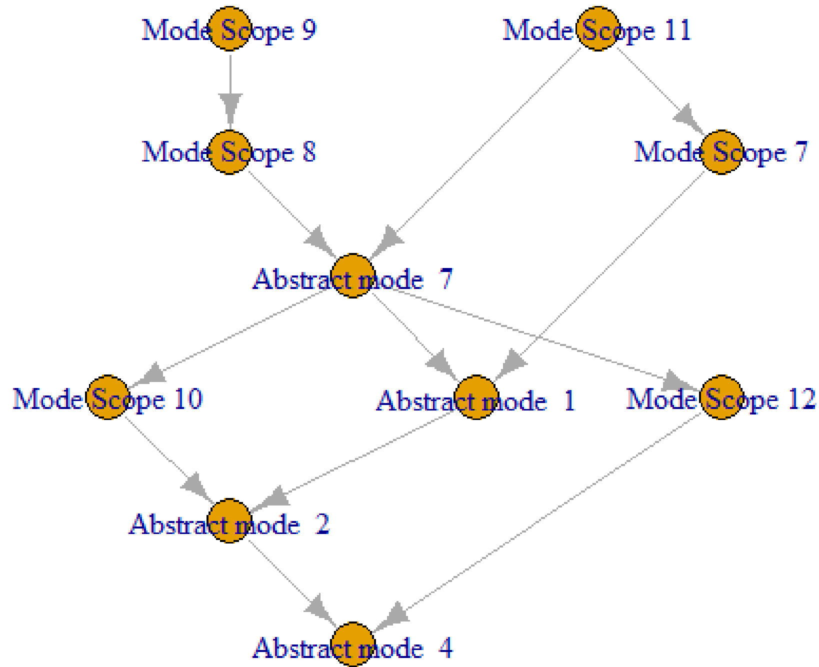

Figure 6.4: Implication structure of scope modes and abstract modes

The structure of modes is generated thanks to a script. The result of its application on the case study is shown in Figure 6.4, using the modes of 6 scopes for visibility. As the modes imply each other, a path of implication correspond to preconditions that are more and more specific. Keeping only the longest paths, obtained by transitive reduction (which as been applied in the example), correspond to progressive definitions of increasing details in the preconditions that are each evaluated once.

We obtain a hierarchy of constraints, each evaluated only once and more specific than the previous ones. The conditions of a mode activated are reused to check the evaluation of others modes conditions implicating it.

6.5 Synthesis

6.5.1 Method

The system is modeled while considering separately its states and its behavior. The same approach is then applied on its elements, detailing the behavior while maintaining states and their traceability. The goal is to specify and check an integrated system and its behavior that would otherwise be either unspecified or emergent depending on the level of development. The method developed follows this process:

1. Define types of states providing information on the system at its own level of granularity.
2. Define simple and complex constraints between the values of different types of states to correlate the information.
3. Define enabling circumstances of the system use cases using the system states.
4. Check all states constraints and use cases preconditions.
5. Generate the structure of modes.
6. Build the system description and behavior models.

Once an integrated model of the system is obtained, more V&V activities can be performed, such as simulation. The model evolves by adding the system elements (such as the subsystems), their description (states) and their behavior (modes). The states of the system remains the same (unless they were ill-specified) but can now be deducted from lower level information. This requires specifying new state machine managing the evolution of the system states. The behavior of the system can be changed so that some of its mechanisms are now detailed in the system elements' behaviors.

6.5.2 Traceability

The information expressed by the states can evolve internally through deduction among types of states. For example, the train operability state depends on the train energy supply state. There needs to be states machines managing the evolution of the operability states depending on the energy supply states. They also can be modified by the behavior or be communicated by the environment. The use cases, however, express interaction with the system processed through its behavior, and do not directly change the system state. This way, both the behavior and the actions on the system can evolve and be detailed without modifying the description of the system.

Information is managed through the states and properly defined in each system elements. Dependencies between information is managed through control states. Information is traced among levels of abstraction not by being refined but by being expressed in the right system element at its local level of abstraction.

Chapter 7

Conclusion

In this chapter, we analyze the main contributions of this thesis and consider the gains obtained. We compare them to the needs expressed by BT and conclude on future perspectives.

7.1 Contributions synthesis

7.1.1 Concepts of states and modes

We present here a synthesis of the contribution defining the concepts of states and modes, presented in the chapter 4.

States and modes : summary

We provided a definition of states and modes as concepts that can be used in different languages and semantics, and hinted at their usefulness in specifying and designing the system behavior.

We now have:

- Concepts characterizing the system and linked to model elements (**N_3.1.**).
- A clarification of the concepts of state and modes (**N_3.1.1-4.**).
- A way to express information at the SOI's level of granularity and abstraction (**N_3.4.**).

- Shared information that goes beyond the scope of one engineer's work (**N_3.**.
- A way to specify dynamic aspects of the behavior (**N_1.**).

What does this definition means for BT?

These definitions are prerequisites for the rest of the work. They ensure a common understanding of the basic concepts for modeling, which can improve communication et cooperation betweens BT engineering teams.

The definitions of states and modes enable functional engineers to correctly specify the use of a train in an operational context. The lack of a way to properly define and manage such information led to the development of useless functions. This in turn directed the engineers to create flawed validation plan from ill-specified or irrelevant scenarios. The solution proposed accelerates the creation of a validation plan.

What does this definition means for MBSE?

The interest for MBSE is much more pronounced, as it is a solid base for the formalism and the design of existing and future methods. The current work aims to eliminate the ambiguity on those abstract terms, which are frequently used in the field.

The workshops conducted in BT provided a positive feedback on the proposed definitions. These definitions were also presented at an international conference [95], where researchers and engineers agreed on the intent and usefulness of providing such definitions. The type of executable model presented is intended to be the base of the development of MIL in BT. The definitions provided are a proposal to solve the challenge identified at **C_7** and provide a basis for **C_4**.

7.1.2 Model verification method

One of the main needs of BT was the definition of a method to improve the quality of the collaborative work of the engineers during the modeling process, as expressed in **N_3.** and **N_3.3.**.

Model verification method : summary

The method developed ensures that the roles for the design and approval of the model verification rules are well defined. The verification rules can then be implemented us-

ing the OCL language and will ensure that the model remains consistent through its specification during the modeling stage, satisfying the needs **N_2.** and **N_2.3.**

The method also describes how the participants should interact and ensure that they do so using shared semantics to prevent ambiguity and interpretation errors, which corresponds to the needs **N_3.** and **N_3.2.**.

What does this model verification method means for BT?

The implementation of this method allows to perform automatic and efficient V&V activities on the model and accelerates the system development process with less time consumed on reviews of the model. The solution proposed has been tested, deployed and validated, and is now in use. The use of shared semantics and reusable rules package allows to recycle large parts of work between projects. This reusability is a major gain, as it saves time and ensures that the good practices from previous experience are reused.

Regarding the need for model verification, the solution has now been deployed on six projects, in Europe and India. It has been concluded that this solution enabled to replace a three-month review process done manually by engineers. This corresponds to an average saving of 1500 hours of work for each project. The amount of errors detected and corrected is about 2000 on average.

What does this model verification method means for MBSE?

Information regarding specifications and system concepts can now be expressed and traced in the models. Knowledge concerning the system can be contained in the models and not just in the engineers' minds. Having information correctly expressed and traced can also avoid some redundancy in the specification.

We provided a way to enforce modeling rules to properly define and keep track of the information in models created following a modeling method and a profile. We now have:

- An automatic verification solution for specification teams to check the information in their models.
- A way to check models so the information they contain can be used for V&V activities.
- A way to define the new concepts static semantics in the expression and manipulation of model elements according to the method.

- A way to enforce traceability of information across requirements and modeling elements in and between development steps along the design process.
- A way to share and communicate information through a same way of expressing and understanding models.

7.1.3 Behavior verification method and model

Behavior verification method and model: summary

The third contribution provides both a verification method for the whole behavior and a way to model it. The verification method enables to define simple and complex constraints which restrict the model's possible states. These constraints are used to define the system behavior **N_1.**. Constraints are used to calculate all possible state configurations of the train, checking that specifications and expectations regarding the evolution of the system are consistent, answering the needs **N_2.** and **N_2.2.**.

States are used to define the preconditions of the use cases, satisfying the need **N_1.3.**. The preconditions are checked regarding the constraints and used to define a structure of modes to model the behavior, answering the needs **N_1.2.** and **N_1.4.**. The activation status of all modes can be deduced from a single evaluation of each condition used in modes and use cases preconditions, enabling to consider and then specify dynamic aspects. This corresponds to the need **N_1.1.**.

The definition of these constraints is made at a high level of abstraction and are forwarded during the different stage of the modeling, which ensures traceability of the requirements. Simple and complex constraints, use cases preconditions and modes are specifications that will be used for the rest of the development, as they do not characterize a design but the behavior that any design has to induce. This answer the needs **N_2.1.**, **N_2.3.**, **N_3.** and **N_3.3.**

What does this solution means for BT?

This method allows automatics V&V activities on the system specifications and prevents conceptual errors. The constraint specification system is very simple, yet it is efficient enough to detect multiple anomaly types.

The main gain is the possibility to correlate information and check the behavior as a whole. Engineers use a same, formalized information to make specifications, which are automatically checked. The solution was tested on the full information of a real project, MOVIA Maxx, after the current V&V process was performed. It appeared that the

constraints and preconditions provided by engineers still contained errors. Nearly all preconditions of the use cases had unused values, and more importantly there were use case preconditions that were not satisfied by any state configuration, meaning that the specification contained critical errors that could not be detected at this step by existing solutions in BT.

The solution proposed is also interesting regarding the re-utilization of models and specifications by providing context. Indeed, taking information from a previous project with missing information regarding the modes can be detrimental.

What does this solution means for MBSE?

Our results helped proving that our definition of system states enables to specify the system and its behavior, its evolution and the conditions under which use cases can be performed. The most direct benefit of this work is a means to check the preconditions of the use cases, not just individually but as part of a constrained whole. The simplicity of the solution makes it available to any engineer working on use case specification and can be easily integrated in an existing method, process or tool. One of the gains expressed by the BT engineers who experimented our approach was that preconditions were clearer but more importantly centralized, and instead of repeating similar information and preconditions in many requirements, this information is expressed by the same elements, the states.

The generation of a structure of modes enables us to integrate the behavioral models developed in each of the scopes of study. Those models can be transformed into executable models. We also have constraints and properties that have to be checked using these models. This enables to conduct V&V activities on the whole system behavior, which was the main goal of this thesis.

7.2 Impact of the global solution

7.2.1 Progress toward the overall objective of BT

BT's objective is to have a continuous validation of a train system behavior along its development process. We saw that the definition of the behavior, its integrated specifications and the validation criteria were missing. The object of this thesis was to identify those issues and provides solutions.

We created a method to specify and integrate a system behavior while enabling its validation. It satisfies the requirements listed in the chapter 2, as shown in their

summaries.

7.2.2 Consequences on BT

The implementation of the three contribution has several impacts on the existing method. It could be summarized as an improvement over the existing process. The definitions of the abstract concept of modes and states allow a clearer communication betweens the services and making their cooperation easier. The method proposed to design new integrated system aims to make all contributions to the model more efficient, as the information is centralized, shared and verified more easily. The two last contribution have also a measurable impact on the V&V process, as a large part of the verification activities can be automated and reduce the scope where an expert intervention is required.

All these improvements lead to a more optimal process, which takes less time, ensures a better synergy betweens services and improves the V&V efficiency, as shown by the quantitative gains mentioned before.

7.2.3 Consequences on MBSE

The way the contributions work toward solving the research challenges presented in the introduction are presented here:

C_1 Having an integrated model of the system at the early stage of design.

We have used the concepts of states and modes to describe a system and its behavior, without relying on a design or an architecture of functions, subsystems or components.

C_2. Making Use of SysML elements to specify requirements regarding the system behavior.

We presented how SysML elements could be used to define states and modes, allowing to specify the behavior. Being independent from a design, they can be used as requirements for an implementation.

C_3. Anticipating and/or managing emergence phenomenon in the behavior.

We defined simple and complex constraints in order to reduce the amount of possible configurations, and hence the possible reactions of a train system.

C_4. Formalizing, completing and standardizing information in the specifications.

The use of verification rules supported by a method enables to properly implement a modeling method as part of a development process. This enforces a semantics associated to the modeling elements. The definition of states and modes has enabled engineers to work on the same information when creating scenarios and preconditions.

C_5. Integrating and applying V&V solutions in the overall development and modeling process.

The challenge here was to provide a MIL solution to complete BT V&V process. We have provided a way to create models as well as validations requirements. The verification rules and the verification method can be considered as part of the V&V process.

C_6. Expressing and continuously checking system properties through different or evolving models.

The system properties have been expressed through the definitions of constraints, be they simple or complex. As these constraints are defined on states that are part of the global system specifications, they remain relevant along the development process. While this thesis does not cover the actual use of these properties across the whole development process, it is assumed to be simple enough considering the solutions available.

C_7. Supporting the MBSE approach by a formal ontology

Our definition of states and modes is an effort toward establishing an ontology for MBSE. We argue that they are core concepts needed to represent a system and its behavior. Having rules to enforce the semantics in models contributes to formalize the concepts they express.

7.3 Future work

7.3.1 Remaining work to be done in the company

Modeling methods in BT, especially regarding the operability where it is new, still have to be developed and completed. While we provide definitions of states and modes and

their application, they still have to be implemented in the official modeling method. A new team in Canada has been put in charge of developing the new modeling method at the end of the thesis and has been given the solutions and comments presented here as inputs.

The new specifications and means of verification and integration have to be deployed in actual projects, something prevented until now by the scale of the effort required and the need to have a finished, validated solution before attempting to deploy it. Verification rules were a particular case, as they were the first to be developed, followed a whole process of test and validation, and supported the development without requiring additional efforts from the engineers or a change in the methods.

As the verification rules are currently deployed and used in the company, it is becoming necessary to create a proper organization following the roles detailed in the method. A centralized team developing and managing verification rules will be created, with designated method experts working with projects managers deciding on the rules packages to be allocated to each project. The team developing the rules could then develop solution to quickly generate more complex rules, enabling to dynamically develop validation rules at the request of validation experts reviewing the models. An estimated team of five to ten people could be constituted, depending on the rate at which the solution is deployed and used.

Executable models have to be integrated in the current co-simulation solution in BT, for which the process has already been started and depends in part on external providers. Once the models can be implemented in a co-simulation, a full MIL analysis will have to be conducted. Engineers are working on developing the co-simulation solution in BT. Their method has been reviewed based on the analysis presented here. A one-year postdoctoral position could be created to implement the executable models and integration method into this co-simulation. Developing model checking solutions on the co-simulation would be investigated. The creation of SysML executable models for a whole project would require three or four people trained to the new methods and solutions.

The solutions presented increase the reutilization capabilities of models and specifications regarding operational and functional aspects. To improve re-usability even further, BT wishes for a modeling approach based on building blocks. There is a need to develop a framework to develop models to reuse only part of a model.

The functional behavior of the train can now be well specified and well tested. The next step would be to define a way to properly define, verify and validate the train physical performances through the parametrization of models.

7.3.2 Research challenge follow-up and perspectives

The work presented relates to several research challenges, as presented earlier. The main contributions of this work are based on the definition of the concepts of states and modes. It participates in defining a formal system ontology in MBSE but does not achieve it. Concepts such as system, models, verification and validation can have various definitions. It is currently not possible to cite one source as a reference, the standards providing variations of definition rather than unifying ones.

As we considered challenges regarding ontology, modeling method and means of integration, we head toward the next challenge presented in the INCOSE roadmap: the need for distributed and secure model repositories crossing multiple domains. The integrated information in our models ought to be referenced from a central place across projects, domains and development steps, something that has not been considered yet.

The use of OCL rules to enforce an ontology and a semantics is a current field of research. We performed verification on SysML models, and provided some bases to implement an ontology and semantics associated to a SysML profile. A generalized solution enabling such an endeavor could be investigated.

This work aims to trigger the MBSE community and particularly the SysML working group in upcoming conference workshops to consider V&V more in detail in future SysML versions (e.g., 2.0) in order to solve industrial adoption challenges from a language perspective. The SysML V2 working group [96] states that the next version of SysML should enable a concise representation of the concepts and be able to validate that the model is logically consistent. It should also be highly adaptable and customizable in regard of domain specific concepts. This thesis constitutes a first step in this direction. It has been adapted to the railway system but could be generalized.

Lastly, a new project conducted in BT company has to be considered, regarding the development of an artificial intelligence. The goal is to develop an autonomous train. In order to do that, functional engineers have to specify the behavior of the AI agent, as it has to reproduce a train driver behavior. Some security constraints that have been empirically enforced through physical separation of commands in the train, as well as the training of a human driver, do no longer apply. There is a need to model, specify and constrain a train behavior and that of its driver as inputs to the development of an artificial intelligence. The present work offers a solution that was not available before in BT and will be necessary in these new endeavors.

Bibliography

[1] *ISO/IEC/IEEE 24765: 2017(E): ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary.* IEEE, 2017. [Online]. Available: https://books.google.fr/books?id=NS02tAEACAAJ

[2] L. E. Hart, "Introduction to model-based system engineering (MBSE) and SysML," in *Delaware Valley INCOSE Chapter Meeting, Ramblewood Country Club, Mount Laurel, New Jersey*, 2015.

[3] OMG, "OMG Systems Modeling Language (OMG SysML^{TM}) Version 1.4," 2015. [Online]. Available: http://www.omg.org/spec/SysML/1.4/

[4] S. Friedenthal and M. Sampson, "INCOSE IW 2014 MBSE Workshop." Jan. 2014.

[5] M. Chami, P. Oggier, O. Naas, and M. Heinz, "Real World Application of MBSE at Bombardier Transportation," in *The Swiss Systems Engineering Day (SWISSED 2015), Kongresshaus Zurich*, 8th September 2015. [Online]. Available: http://ssse.ch/sites/default/files/page_images/%3Cem%3EEdit%20Basic%20page%3C/em%3E%20SWISSED%202015/MBSE%20at%20BT%20-%20SWISSED2015%20-%2020150908%20-%20V1_1.pdf

[6] *ISO/IEC/IEEE 15288: 2015(E): ISO/IEC/IEEE International Standard - Systems and Software Engineering: Software Life Cycle Processes.* IEEE, 2015. [Online]. Available: https://ieeexplore.ieee.org/servlet/opac?punumber=4475823

[7] J. G. Lamm and T. Weilkiens, "Functional Architectures in SysML," *Proceedings of the Tag des Systems Engineering (TdSE '10). Munich, Germany*, 2010. [Online]. Available: https://www.researchgate.net/profile/Tim_Weilkiens/publication/267802862_Functional_Architectures_in_SysML/links/558bb23208ae02c9d1f967bf.pdf

[8] R. J. Abbott, "Emergence and systems engineering: putting complex systems to work," in *Symposium on Complex Systems Engineering, RAND Corporation, Santa Monica, CA*, 2007, pp. 11–12.

[9] B. Haskins, J. Stecklein, B. Dick, G. Moroney, R. Lovell, and J. Dabney, "8.4.2 error cost escalation through the project life cycle," *INCOSE International Symposium*, vol. 14, pp. 1723–1737, 06 2004.

[10] M. Debbabi, F. Hassaïne, Y. Jarraya, A. Soeanu, and L. Alawneh, *Verification and Validation in Systems Engineering.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. [Online]. Available: http://link.springer.com/10.1007/978-3-642-15228-3

[11] IEEE, *1012-2012 IEEE Standard for System and Software Verification and Validation.* [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=6204024

[12] L. van Ruijven, "Ontology for Systems Engineering as a base for MBSE," *INCOSE International Symposium*, vol. 25, no. 1, pp. 250–265, Oct. 2015. [Online]. Available: http://doi.wiley.com/10.1002/j.2334-5837.2015.00061.x

[13] M. Chami and J.-M. Bruel, "A Survey on MBSE Adoption Challenges," *Sector Systems Engineering Conference (INCOSE EMEASEC 2018)*, p. 16, 2018.

[14] M. D. Ingham, R. D. Rasmussen, M. B. Bennett, and A. C. Moncada, "Engineering complex embedded systems with state analysis and the mission data system," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 12, pp. 507–536, 2005. [Online]. Available: https://doi.org/10.2514/1.15265

[15] J.-R. Abrial, *Modeling in Event-B: system and software engineering.* Cambridge; New York: Cambridge University Press, 2010, oCLC: 646068815.

[16] Thiago Rocha Silva, "A behavior-driven approach for specifying and testing user requirements in interactive systems," PhD Thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2018. [Online]. Available: http://thesesups.ups-tlse.fr/3940/

[17] A. Benveniste, B. Caillaud, and D. e. a. Nickovic, "Contracts for Systems Design: Theory," Inria Rennes Bretagne Atlantique ; INRIA, Research Report RR-8759, Jul. 2015. [Online]. Available: https://hal.inria.fr/hal-01178467

[18] M. Soeken and R. Drechsler, *Formal Specification Level.* Cham: Springer International Publishing, 2015. [Online]. Available: http://link.springer.com/10.1007/978-3-319-08699-6

[19] A. Salado and P. Wach, "Constructing True Model-Based Requirements in SysML," *Systems*, vol. 7, no. 2, p. 19, Mar. 2019. [Online]. Available: https://www.mdpi.com/2079-8954/7/2/19

[20] Y. Mordecai and D. Dori, "Model-based requirements engineering: Architecting for system requirements with stakeholders in mind," in *2017 IEEE International Systems Engineering Symposium (ISSE).* Vienna, Austria: IEEE, Oct. 2017, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/8088273/

[21] Mike Ryan, Stephen Cook, and William R. Scot, “Application of MBSE to requirements engineering - Research challenges,” *Systems Engineering/Test and Evaluation Conference*, 2013. [Online]. Available: https://ro.uow.edu.au/eispapers/6335/

[22] R. Fujimoto, C. Bock, W. Chen, E. Page, and J. H. Panchal, *Research challenges in modeling and simulation for engineering complex systems*. Springer, 2017.

[23] Mikel D. Petty, “Modeling and Validation Challenges for Complex Systems,” in *Engineering Emergence A Modeling and Simulation Approach*, 2011.

[24] Eric Vugrin, Timothy Trucano, Laura Swiler, Patrick Finley, Tatiana Flanagan, Asmeret Naugle, Jeffrey Tsao, and Stephen Verzi, “Recommend Research Directions for Improving the Validation of Complex Systems Models,” Sandia National Laboratories, Tech. Rep., 2017.

[25] L. Lemazurier, V. Chapurlat, and A. Grossetête, “An MBSE Approach to Pass from Requirements to Functional Architecture,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 7260 – 7265, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896317319183

[26] Emmanouela Stachtiari, Anastasia Mavridou, Panagiotis Katsaros, Simon Bliudze, and Joseph Sifakis, “Early Validation of System Requirements and Design Through Correctness-by-Construction,” 2018.

[27] Catherine Dubois, Michalis Famelis, Martin Gogolla, Leonel Nobrega, Ileana Ober, Martina Seidl, and Markus Völter, “Research Questions for Validation and Verification in the Context of Model-Based Engineering,” *International Workshop on Model Driven Engineering, Verification and Validation - MoDeVVA 2013*, 2014.

[28] “Work breakdown structure, 000454 (BT internal document).”

[29] M. F. Michael Bogaert, Francis Tison, “Functional design training (BT internal document).”

[30] J. Yao, “0r_1 Operability Concept (BT internal document).”

[31] E. Hull, K. Jackson, and J. Dick, *Requirements engineering*, 3rd ed. London ; New York: Springer, 2011.

[32] AIAA, *ANSI/AIAA G-043 A-2012 Guide to the Preparation of Operational Concept Documents*. ANSI, 2012. [Online]. Available: https://webstore.ansi.org/standards/aiaa/ansiaiaa043a2012

[33] A. C. Ouafa Baizigue, Jerome Canipel, “Functional methodology using tools (BT internal document).”

[34] E. Doba, “Vehicle Platform SysML Modelling Methodology (BT internal document).”

[35] N. H. Cyril Jonvel, "Vehicle Functional Architecture Modelling Methodology (BT internal document)."

[36] M. C. O. Naas, "MBSE training course (BT internal document)."

[37] C. S. Wasson, *System engineering analysis, design, and development: Concepts, principles, and practices*, 2016, oCLC: 949896718.

[38] J. Karl-Heinz and M. TiegelKamp, "IEC61131-3 Programming Industrial Automation Systems." [Online]. Available: http://www.dee.ufrj.br/controle_automatico/cursos/IEC61131-3_Programming_Industrial_Automation_Systems.pdf

[39] MODELISAR and Modelica, "Functional Mock-up Interface for Model Exchange and Co-Simulation V2.0," Jul. 2014. [Online]. Available: https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf

[40] T. Weilkiens, *SYSMOD - The Systems Modeling Toolbox - Pragmatic MBSE with SysML*, 2016. [Online]. Available: https://www.researchgate.net/publication/311772243_SYSMOD_-_The_Systems_Modeling_Toolbox_-_Pragmatic_MBSE_with_SysML_2nd_edition

[41] D. D. Walden, G. J. Roedler, K. Forsberg, R. D. Hamelin, T. M. Shortell, and International Council on Systems Engineering, Eds., *Systems engineering handbook: a guide for system life cycle processes and activities ; INCOSE-TP-2003-002-04, 2015*, 4th ed. Hoboken, NJ: Wiley, 2015, oCLC: 931708827.

[42] D. Long and Z. Scott, *A primer for model-based systems engineering*, 2011. [Online]. Available: http://www.ccose.org/media/upload/MBSE_Primer_2ndEdition_full_Vitech_2011.10.pdf

[43] T. Weilkiens, J. G. Lamm, S. Roth, and M. Walker, *Model-based system architecture*, ser. Wiley series in systems engineering and management. Hoboken, New Jersey: Wiley, 2016, oCLC: 953572265.

[44] T. Cziharz, P. Hruschka, S. Queins, and T. Weyer, "Handbook of Requirements Modeling According to the IREB Standard," p. 115, 2016.

[45] T. Fayolle, "Specifying a train system using astd and the B method : Technical Report," LACL, Laboratoire d'Algorithmique, Complexité et Logique, Tech. Rep., 2014.

[46] D. Kaslow, G. Soremekun, H. Kim, and S. Spangelo, "Integrated model-based systems engineering (MBSE) applied to the Simulation of a CubeSat mission," in *2014 IEEE Aerospace Conference*. Big Sky, MT, USA: IEEE, Mar. 2014, pp. 1–14. [Online]. Available: http://ieeexplore.ieee.org/document/6836317/

[47] R. Karban, A. Crawford, G. Trancho, M. Zamparelli, S. Herzig, I. Gomes, E. Brower, and M. Piette, "The OpenSE Cookbook: a practical, recipe based collection of patterns, procedures, and best practices for executable systems engineering for the Thirty Meter Telescope," in *Modeling, Systems Engineering, and Project Management for Astronomy VIII*, G. Z. Angeli and P. Dierickx, Eds. Austin, United States: SPIE, Jul. 2018, p. 31. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10705/2312281/The-OpenSE-Cookbook--a-practical-recipe-based-collection-of/10.1117/12.2312281.full

[48] Jean-Marie Gauthier, Fabrice Bouquet, Ahmed Hammad, and Fabien Peureux, "Tooled Process for Early Validation of SysML Models using Modelica Simulation," 2015.

[49] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mau\s s, M. Monteiro, T. Neidhold, D. Neumerkel, and others, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany.* Linköping University Electronic Press, 2011, pp. 105–114. [Online]. Available: http://www.ep.liu.se/ecp/article.asp?issue=063&volume=&article=13

[50] U. Wurstbauer, M. Herrnberger, A. Raufeisen, and V. Fä\s sler, *Efficient Development of Complex Systems using a Unified Modular Approach.* Deutsche Gesellschaft für Luft-und Raumfahrt-Lilienthal-Oberth eV, Bonn, 2015. [Online]. Available: http://www.dglr.de/publikationen/2015/370201.pdf

[51] Y. A. Feldman, L. Greenberg, and E. Palachi, "Simulating Rhapsody SysML Blocks in Hybrid Models with FMI," Mar. 2014, pp. 43–52. [Online]. Available: http://www.ep.liu.se/ecp/article.asp?issue=096%26article=4

[52] R. S. Moura and L. A. Guedes, "Simulation of Industrial Applications using the Execution Environment SCXML." IEEE, Jul. 2007, pp. 255–260. [Online]. Available: http://ieeexplore.ieee.org/document/4384765/

[53] M. Chen, X. Qin, H.-M. Koo, and P. Mishra, *System-Level Validation.* New York, NY: Springer New York, 2013. [Online]. Available: http://link.springer.com/10.1007/978-1-4614-1359-2

[54] H. Graves, "Current State of ontology in engineering systems," OMG, Tech. Rep., 2012. [Online]. Available: http://www.omgwiki.org/MBSE/doku.php?id=mbse:ontology

[55] U. A\s smann, S. Zschaler, and G. Wagner, "Ontologies, meta-models, and the model-driven paradigm," in *Ontologies for software engineering and software technology.* Springer, 2006, pp. 249–273. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-34518-3_9

[56] D. Price, A. B. Feeney, and A. Jones, "A Semantic Framework for Systems Engineering Standards," *INCOSE International Symposium*, vol. 23, no. 1, pp. 1256–1270, Jun. 2013. [Online]. Available: http://doi.wiley.com/10.1002/j.2334-5837.2013.tb03084.x

[57] D. B. Matthews, "Semantic Web Technologies," *Semantic Web Technologies*, p. 21, 2005.

[58] A. Baruzzo and M. Comini, "Static verification of UML model consistency," in *3rd Workshop on Model Design and Validation (MoDeV2a)*. Citeseer, 2006, pp. 111–126.

[59] M. Gogolla and F. Hilken, "Model validation and verification options in a contemporary UML and OCL analysis tool," *Modellierung 2016*, 2016.

[60] A. Hafeez and A. u. Rehman, "Ontology Based Verification of UML Class/OCL Model," *Mehran University Research Journal of Engineering and Technology*, vol. 37, no. 4, pp. 521–534, Oct. 2018. [Online]. Available: http://publications.muet.edu.pk/index.php/muetrj/article/view/560

[61] D. A. Wagner, M. B. Bennett, R. Karban, N. Rouquette, S. Jenkins, and M. Ingham, "An ontology for State Analysis: Formalizing the mapping to SysML," in *2012 IEEE Aerospace Conference*. Big Sky, MT: IEEE, Mar. 2012, pp. 1–16. [Online]. Available: http://ieeexplore.ieee.org/document/6187335/

[62] Ludwig von Bertalanffy, *General System Theory*, 1968.

[63] K. M. Adams and T. J. Meyers, "Systems theory: a formal construct for understanding systems," *International Journal of System of Systems Engineering*, vol. 2, no. 2-3, pp. 163–192, 2011.

[64] K. M. Adams and J. H. Mun, "The application of systems thinking and systems theory to systems engineering," in *Proceedings of the 26th National ASEM Conference: Organizational Transformation: Opportunities and Challenges*, vol. 10. American Society for Engineering Management Rolla, MO, 2005, pp. 493–500.

[65] J. Martin, J. B. SE, G. C. AUT, D. Hybertson, R. Martin, H. S. UK, J. Singer, M. Singer, and T. T. JP, "Team 4: Towards a Common Language for Systems Praxis," *International Federation For Systems Research*, p. 75, 2012.

[66] J. Singer, H. Sillitto, J. Bendz, G. Chroust, D. Hybertson, H. Lawson, J. Martin, R. Martin, M. Singer, and T. Takaku, "The *Systems Praxis Framework*," in *Systems and Science at Crossroads – Sixteenth IFSR Conversation*, ser. SEA-SR-32. Linz, Austria: Inst. f. Systems Engineering and Automation, Johannes Kepler University, September 2012, pp. 89–90. [Online]. Available: http://www.ifsr.org/index.php/category/archives/ifsr-conversations;printablebrochureavailablefromhttp://systemspraxis.org

[67] J.-L. Le Moigne, *La théorie du système général: théorie de la modélisation.* jean-louis le moigne-ae mcx, 1994.

[68] D. Ing, "Rethinking Systems Thinking: Learning and Coevolving with the World." vol. 30, no. 5, 2013.

[69] R. Descartes and G. Gröber, *Discours de la méthode: 1637.* Heitz, 1905.

[70] L. Pazzi, "Modeling Systemic Behavior by State-Based Holonic Modular Units," in *Model-Driven Engineering Languages and Systems*, J. Dingel, W. Schulte, I. Ramos, S. Abrahão, and E. Insfran, Eds. Cham: Springer International Publishing, 2014, vol. 8767, pp. 99–115. [Online]. Available: http://link.springer.com/10.1007/978-3-319-11653-2_7

[71] G. Guizzardi, "Ontological foundations for structural conceptual models," Ph.D. dissertation, Centre for Telematics and Information Technology, Telematica Instituut, Enschede, The Netherlands, 2005, oCLC: 225528610. [Online]. Available: https://www.inf.ufes.br/~gguizzardi/OFSCM.pdf

[72] OMG, "OMG Unified Modeling Language (UML) Version 2.5.1," 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/PDF

[73] A. Naumenko, "Triune Continuum Paradigm: a paradigm for General System Modeling and its applications for UML and RM-ODP," 2002.

[74] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal," in *Formal methods for the design of real-time systems.* Springer, 2004, pp. 200–236. [Online]. Available: http://link.springer.com/10.1007%2F978-3-540-30080-9_7

[75] E. Börger and R. Stärk, *Abstract state machines: a method for high-level system design and analysis.* Springer Science & Business Media, 2003.

[76] C. S. Wasson, "System Phases, Modes, and States Solutions to Controversial Issues," *Wasson Strategics, LLC. http://www. wassonstrategics. com*, 2010.

[77] A. M. Olver and M. J. Ryan, "On a Useful Taxonomy of Phases, Modes, and States in Systems Engineering," *Systems Engineering / Test and Evaluation Conference SETE2014*, Apr. 2015.

[78] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[79] *SMC Systems Engineering Primer & Handbook*, 2005.

[80] J.-L. Voirin, *Model-based System and Architecture Engineering with the Arcadia Method.* ISTE Press - Elsevier, 2018.

[81] J. Jenney, "Define Life Cycle System Modes," *http://themanagersguide.blogspot.com/2011/01/6322-define-life-cycle-system-modes.html*, 2011.

[82] M. Chami, A. Aleksandraviciene, A. Morkevicius, and J.-M. Bruel, "Towards Solving MBSE Adoption Challenges: The D3 MBSE Adoption Toolbox," *INCOSE International Symposium*, vol. 28, no. 1, pp. 1463–1477, Jul. 2018. [Online]. Available: http://doi.wiley.com/10.1002/j.2334-5837.2018.00561.x

[83] S. Bonnet, D. Exertier, and V. Normand, "Not (strictly) relying on SysML for MBSE: language, tooling and development perspectives," 2015, oCLC: 255348295. [Online]. Available: http://download.polarsys.org/capella/publis/IEEE_Capella_SysML_paper.pdf

[84] K. Berkenkotter, "OCL-based Validation of a Railway Domain Profile," 2006, oCLC: 248918751. [Online]. Available: https://st.inf.tu-dresden.de/OCLApps2006/topic/acceptedPapers/11_Berkenkotter_ValidationDomainProfile.pdf

[85] I. Dragomir, I. Ober, and C. Percebois, "Contract-based modeling and verification of timed safety requirements within sysml," *Software and System Modeling*, vol. 16, no. 2, pp. 587–624, 2017. [Online]. Available: https://doi.org/10.1007/s10270-015-0481-1

[86] O. M. G. Specification, "Object Contraint Language V.2.4," *Object Management Group pct/07-08-04*, Feb. 2014. [Online]. Available: http://www.omg.org/spec/OCL/2.4/

[87] Eclipse - Papyrus Guide, "Validate (ocl) constraints of a profile." [Online]. Available: https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.papyrus.dsml.validation.doc%2Ftarget%2Fgenerated-eclipse-help%2Fdsml-validation.html

[88] Sparx Systems, "Model Validation," 2016. [Online]. Available: https://www.sparxsystems.fr/resources/user-guides/model-domains/model-validation.pdf

[89] No Magic Documentation, "Creating validation rules - MagicDraw 18.2." [Online]. Available: https://docs.nomagic.com/display/MD182/Creating+validation+rules

[90] M. Altenhofen, T. Hettel, and S. Kusterer, "OCL support in an industrial environment," 2006. [Online]. Available: http://www-st.inf.tu-dresden.de/OCLApps2006/topic/acceptedPapers/03_Altenhofen_OCLSupport.pdf

[91] R. Delmas, A. F. Pires, and T. Polacsek, "A verification and validation process for model-driven engineering," C. Vallet, D. Choukroun, C. Philippe, G. Balas, A. Nebylov, and O. Yanova, Eds. EDP Sciences, 2013, pp. 455–468. [Online]. Available: http://www.eucass-proceedings.eu/10.1051/eucass/201306455

[92] P. Baracos, *Grafcet step by step.* Famic Automation, Incorporated, 1992.

[93] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 2001. [Online]. Available: http://faraday.fie.umich.mx/~rrusiles/Fie/Horizontal/Hopcroft_Introduction_to_Automata_Theory_Languages_and_Computation.pdf

[94] R. Balakrishnan and K. Ranganathan, *A Textbook of Graph Theory*, ser. Universitext. New York, NY: Springer New York, 2012.

[95] R. Baduel, J. Bruel, I. Ober, and E. Doba, *Definition of states and modes as general concepts for system design and validation.* HAL, 2018. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01989427

[96] OMG, "SysML v2 RFP Working Group." [Online]. Available: http://www.omgwiki.org/

Annexes

The annexes contains three elements, each relevant to one of the contribution.

Operation modes

The first page contains a standard in development in BT, that expresses the different "operation modes" to be used in BT. This document is analyzed in the chapter 2, showing issues in the way modes and the information they contain are expressed. This highlights the need for properly defined concepts of states and modes, which is the contribution presented in chapter 4.

Sample of the documented verification rules

The two pages following the first show parts of the verification rules developed and documented for BT, as part of the contribution presented in chapter 5. It can be seen on the columns that a classification system has been created to manage the rules by specifying their types, the scopes where they are to be applied and their status. The status indicate if the rules are to be developed, to be tested, to be allocated to a package, are ready to use, or deprecated.

Behavior verification and mode generation script

The last 13 pages show the code of the script used in the chapter 6. This script enables to check specifications regarding the system and its behavior for errors, as well as generating a structure of modes to create an integrated model of the system behavior.

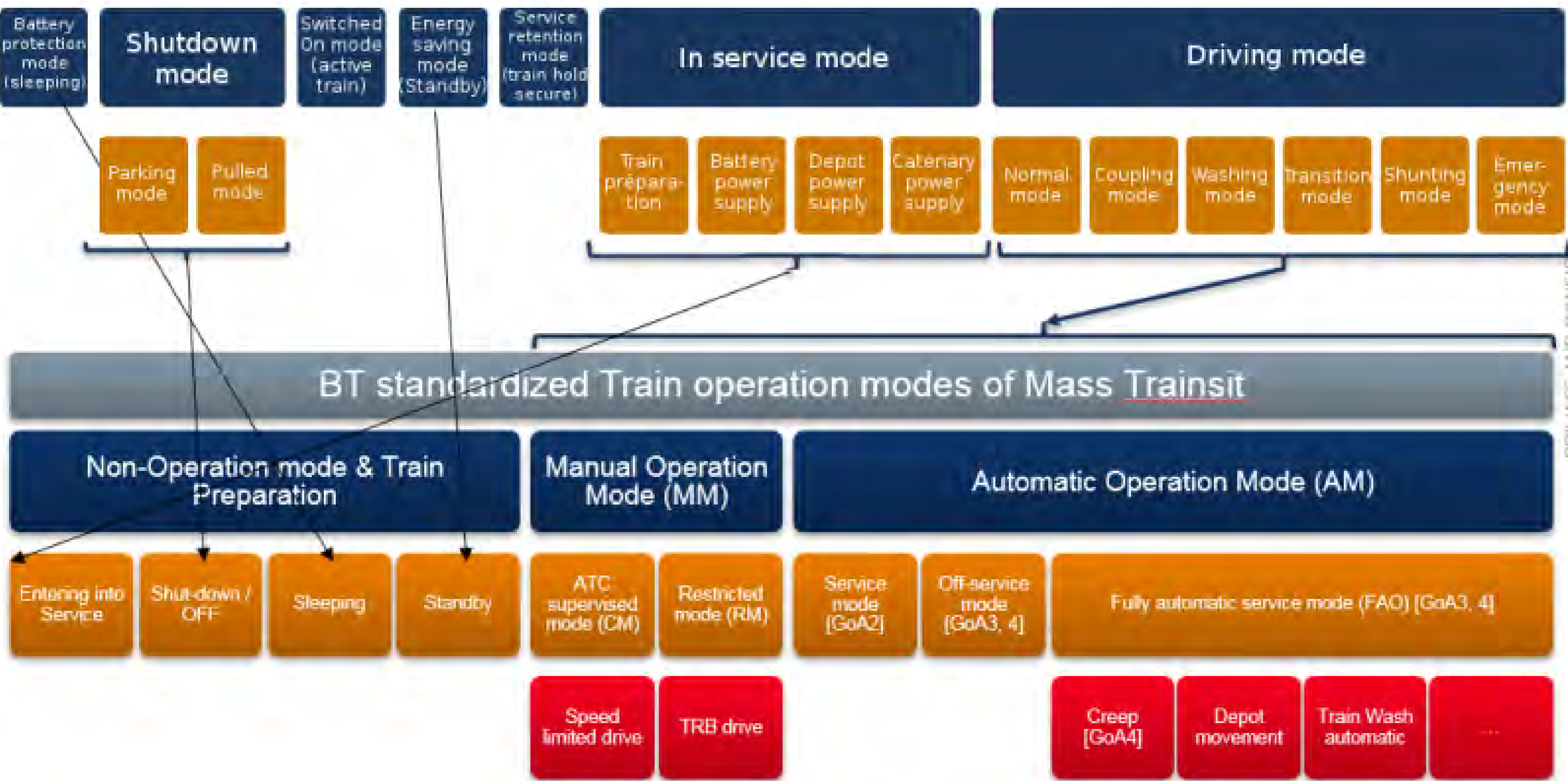

Figure 7.1: Operation modes [30]

Method Validation Rules

#	Name	Constrained Element	Validation Scope	Status	Rule Type	Abbreviation	Severity	Error Message
1	2F_FA_FunctionalBlock_Provides Activities	BT Functional Block [Class]	Functional Analysis	ready	traceability	FuncBlock_ProvideActivity	error	A Functional Block provides at least one Activity
2	2F_FA_FunctionalBlock_Satisfy a Requirement	BT Functional Block [Class]	Functional Analysis	ready	traceability	FuncBlock_SatisfiesRequirement	error	A Functional Block satisfies at least one requirement
3	2F_FA_FunctionalBlock_Compose or Aggregate a Functional Context	BT Functional Block [Class]	Functional Analysis	ready	traceability	FuncBlock_FuncContext	error	A Functional Block should be aggregated or compose a Functional Context
4	2F_FA_FunctionalContext_Owns an IBD	BT Functional Context [Class]	Functional Analysis	ready	traceability	FuncContext_IBD	error	A Functional Context in 2F_FA should contain at least one IBD
5	2F_FA_FunctionalContext_Composed or Aggregated by Functional Block	BT Functional Context [Class]	Functional Analysis	ready	traceability	FuncContext_FuncBlock	error	A Functional Context in 2F_FA should contain at least one Sequence Diagram
6	2F_FA_FunctionalContext_Owns a Sequence Diagram	BT Functional Context [Class]	Functional Analysis	ready	traceability	FuncContext_SeqDiag	error	A Functional Context in 2F_FA should contain at least one Sequence Diagram
7	2F_FA_Messages_All messages in SD should contain a functional signal	Message	Operational Analysis	ready	content	SeqDiag_Msg_Signal	error	All messages in the SD should contain a functional signal
8	2F_OA_Messages_All messages in SD should contain an operational signal	Message	Operational Analysis	ready	content	SeqDiag_Msg_Signal	error	All messages in the SD should contain an operational signal
9	2F_OA_Actor_Aggregated by an Operational Context	Actor	Operational Analysis	ready	traceability	Actor_OpContext	error	An Actor in 2F_OA should be aggregated by at least one Operational Context
10	2F_OA_Actor_Association to a Use Case	Actor	Operational Analysis	ready	traceability	Actor_UseCase	warning	An Actor in 2F_OA should be associated to a Use Case
11	2F_FA_IBD_Owned By a Functional Context	Diagram	Functional Analysis	ready	traceability	IBD_OpContext	error	An IBD in 2F_FA should be contained by a Functional Context
12	2F_OA_IBD_Owned By an Operational Context	Diagram	Operational Analysis	ready	traceability	IBD_OpContext	error	An IBD in 2F_OA should be contained by an Operational Context
13	2F_OA_OperationalBlock_Aggregated by an Operational Context	BT Operational Block [Class]	Operational Analysis	ready	traceability	OpBlock_OpContext	error	An Operational Block should be aggregated by at least one Operational Context
14	2F_OA_OperationalContext_Aggregates an Operational Block	BT Operational Context [Class]	Operational Analysis	ready	traceability	OpContext_OpBlock	error	An Operational Context in 2F_OA should aggregate a single Operational Block
15	2F_OA_OperationalContext_Aggregates an Actor	BT Operational Context [Class]	Operational Analysis	ready	traceability	OpContext_Actor	error	An Operational Context in 2F_OA should aggregate at least one Actor
16	2F_OA_OperationalContext_Owns an IBD	BT Operational Context [Class]	Operational Analysis	ready	traceability	OpContext_IBD	error	An Operational Context in 2F_OA should contain at least one IBD
17	2F_OA_OperationalContext_Owns a Sequence Diagram	BT Operational Context [Class]	Operational Analysis	ready	traceability	OpContext_SeqDiag	error	An Operational Context in 2F_OA should contain at least one Sequence Diagram
18	2F_FA_Requirement_Satisfied by a Functional Block	BT Requirement [Class]	Functional Analysis	ready	traceability	Req_FunctionalBlock	error	A requirement in 2F_FA should be satisfied by a Functional Block
19	2F_OA_Requirement_is Satisfied by a Use Case	BT Requirement [Class]	Operational Analysis	ready	traceability	Req_UseCase	error	A requirement in 2F_OA should be satisfied by a Use Case

20	2F_FA_SequenceDiagram_Owned by a Functional Context	Interaction	Functional Analysis	ready	traceability	SeqDiag_FuncContext	error	A Sequence Diagram in 2F_FA should be contained by a Functional Context
21	2F_OA_SequenceDiagram_Owned by an Operational Context	Interaction	Operational Analysis	ready	traceability	SeqDiag_OpContext	error	A Sequence Diagram in 2F_OA should be contained by an Operational Context
22	2F_OA_SequenceDiagram_Has a documentation	Interaction	Operational Analysis	ready	content	SeqDiag_Doc	warning	A Sequence Diagram in 2F_OA should have a documentation
23	2F_FA_Signal_Allocated to a Technical Signal	BT Functional Signal [Signal]	Functional Analysis	ready	traceability	Signal_AllocationToTechnical	error	A signal in 2F_FA should be allocated to a technical signal
24	2F_FA_Signal_Has a Functional stereotype	Signal	Functional Analysis	ready	content	Signal_Func_Stereotype	error	A signal in 2F_FA should have a functional stereotype
25	2F_OA_Signal_Has an Operational stereotype	Signal	Operational Analysis	ready	content	Signal_Oper_Stereotype	error	A signal in 2F_OA should have an operatonal stereotype
26	2F_TA_Signal_Has a Functional Signal allocated to it	BT Technical Signal [Signal]	Technical Analysis	ready	traceability	Signal_FuncSignalAllocated	error	A signal in 2F_TA should havev a funcctional signal allcoated to it
27	2F_OA_Signal_Name begin by R, I, C or Stat	BT Operational Signal [Signal]	Operational Analysis	ready	content	Signal_Name	error	A signal name in 2F_OA should begin by either "R", "I", "C" or "Stat".
28	2F_OA_InformationFlow_Request Signal direction	InformationFlow	Operational Analysis	ready	content	Signal_R_Direction	error	A signal which name begin by "R" is always linked to a flow that goes from an Actor to an Operational Block.
29	2F_OA_TrainActivity_Same name as Use Case owner	Activity	Operational Analysis	ready	traceability	TrainAct_UC_Name	warning	A Train Activity should have the same name as the Use Case that contains it.
30	2F_OA_UseCase_Owns a single Train Activity	BT Use Case [UseCase]	Operational Analysis	ready	traceability	UseCase_TrainAct	error	A Use case in 2F_OA should contain a single Train Activity bearing the same name
31	2F_OA_UseCase_Satisfy a Requirement	BT Use Case [UseCase]	Operational Analysis	ready	traceability	UseCase_Req	error	A Use case in 2F_OA should satisfy a requirement
32	2F_OA_UseCase_Associated to an Actor	BT Use Case [UseCase]	Operational Analysis	ready	traceability	UseCase_Actor	error	A Use Case should be associated to an Actor
33	2F_OA_UseCase_Has documentation defined	BT Use Case [UseCase]	Operational Analysis	ready	content	UseCase_Documentation	error	A Use Case should have a documentation
34	2F_OA_SequenceDiagram_All SD should contain Pre-conditions	Interaction	Operational Analysis	ready	content	SeqDiag_Preconditions	warning	Each Sequence Diagram shall contain Pre-conditions
35	2F_OA_OperationalContext_Consist linked to Actors through ports	BT Operational Context [Class]	Operational Analysis	ready	traceability	OpBlock_Port and Actor	error	In a given Operational Context, the only connexions are between actors and ports of the Operational Block
36	2F_OA_OperationalContext_Connector unicity	BT Operational Context [Class]	Operational Analysis	ready	multiplicity	OpContext_Connector_Unicity	error	In a given Operational Context, there are only one connector and port per actor interacting with the SOI
37	2F_OA_Use case has triggerSignal	BT Triggered Use Case [UseCase]	Operational Analysis	ready	content	UCTriggerSignal	error	The use case has no trigger signal. [Remedy: Set the trigger signal.]

```
###############################################

    # IMPORT LIBRARIES

###############################################

library(ggplot2)
library("factoextra")
library(e1071)
library(ade4)
library(philentropy)
library(gplots)
library(Hmisc)
library(igraph)
library(Corbi)
library(nem)
library(rlist)
library(data.table)

###############################################

    # IMPORT DATA

###############################################

# Setting workspace

setwd("C:/Users/rbaduel/Documents/Bombardier/Workspace/Rcluster")

# Importing data and ensuring suitable variables type

precondBrut <- data.frame(read.table("StatesPreconditionsCleaned_V2.txt", header = TRUE,
skipNul = TRUE, row.names = 1))
precondFull <- apply (precondBrut, c (1, 2), function (x) { (as.integer(x)) })

postcondBrut <- data.frame(read.table("StatesPostconditionsCleaned_V0.txt", header = TRUE,
skipNul = TRUE, row.names = 1))
postcondFull <- apply (postcondBrut, c (1, 2), function (x) { (as.integer(x)) })

configsBrut <- data.frame(read.table("StateConstraintsCleaned_V3.txt", header = TRUE, skipNul
 = TRUE, row.names = 1))
configs <- apply (configsBrut, c (1, 2), function (x) { (as.integer(x)) })

complexConstBrut <- data.frame(read.table("ComplexStateConstraintsCleaned_V0.txt", header =
TRUE, skipNul = TRUE, row.names = 1))
complexConst<- apply (complexConstBrut, c (1, 2), function (x) { (as.integer(x)) })


###############################################

    # DEFINE VARIABLES

###############################################



# Remove duplicates cancelled to ease the repartition of functions in scopes
#dupl <- precondFull[duplicated(precondFull),]
#
# elDupl <- !duplicated(precondFull)&duplicated(precondFull, fromLast = TRUE)
#
# precond <- precondFull[!duplicated(precondFull),]
#
# precond -> precondFull

precond <- precondFull

# Group preconditions and state constraints by state types

stateSep <- c(2,3,2,3,6,4,3,3,3,2,4,3,3,2,2)
```

```
parts <- partition.matrix(precond, colsep=stateSep )

confParts <- partition.matrix(configs, colsep=stateSep )

# Group use case by scopes

foncSep <- c(6,6,5,2,5,4,5,5,2,2,2,3,2,5)

scopes <- partition.matrix(precondFull, rowsep=foncSep)

# Define state constraint graph

gConf <- as.undirected(graph.adjacency(configs))

# Define list of possible configurations according to simple constraints

confList <- cliques(gConf, min = 15)

###############################################

    # FUNCTIONS: MODES

###############################################

filterStateDep <- function(listStatePart){

    # GIVEN: a list of preconditions partitionned by state types
    # DO: return a list of the state values state types expressing at least one constraint
    in the matrix

    staTypDep <- vector()

    for(i in c(1:length(listStatePart)))
    {
        if (any(listStatePart[[i]]==0))
        {
            staTypDep <- c(staTypDep , colnames(listStatePart[[i]]))
        }
    }
    return(staTypDep)
}

getImplicMatrix <- function(precondMat){

    # GIVEN: a precondition matrix
    # DO: return a matrix of the implications between the preconditions

    # Make a list of authorized state values for each precondition in the matrix

    authStates <- vector("list", nrow(precondMat))

    for(i in c(1:nrow(precondMat)))
    {
        iStates <- vector()
        for (k in c(1:ncol(precondMat)))
        {
            if (precondMat[i,k] == 1)
            {
                iStates <- c(iStates, k)
            }
        }

        authStates[[i]] <- iStates
    }

    # Compare authorized values between each pair of precondition

    implicMat <- matrix(0, length(authStates), length(authStates))

    for (i in c(1:(length(authStates))))
```

```
    {
        for (j in c(1:length(authStates)))
        {
            if((i != j)&(length(setdiff(authStates[[i]],authStates[[j]])) == 0))
            {
                implicMat[i,j] = 1
            }
        }
    }
    rownames(implicMat) = rownames(precondMat)
    colnames(implicMat) = rownames(precondMat)

    return(implicMat)
}

getNoImply <- function(implicMat){

    # GIVEN: an implication matrix
    # DO: return elements that do not imply any other

    imply <- vector()

    for (i in c(1:nrow(implicMat)))
    {
        if(any(c(implicMat[i,])==1))
        {
            imply <- c(imply , i)
        }
    }
    noImply <- c(setdiff(c(1:nrow(implicMat)), imply))
    return(noImply)
}

getNotImplied <- function(implicMat){

    # GIVEN: an implication matrix
    # DO: return elements that are not implied by any other

    implied <- vector()

    for (i in c(1:ncol(implicMat)))
    {
        if(any(c(implicMat[,i])==1))
        {
            implied <- c(implied , i)
        }
    }
    notImplied <- c(setdiff(c(1:ncol(implicMat)), implied))
    return(notImplied)
}

getAbstModes <- function(precondMat){

    # GIVEN: a matrix of modes preconditions without implications
    # DO: return a precondition matrix of abstract modes

    nAsbtMode <- (nrow(precondMat)*(nrow(precondMat)-1)/2)
    abstModMat <- matrix(0L, nrow = nAsbtMode,ncol = ncol(precondMat))
    colnames(abstModMat) <- colnames(precondMat)

    k <- 1

    for(i in c(1:(nrow(precondMat)-1)))
    {
        for(j in c((i+1):(nrow(precondMat))))
        {
            for(z in c(1:ncol(precondMat)))
            {
                if(precondMat[i,z] == precondMat[j,z])
                {
                    abstModMat[k,z] = precondMat[i,z]
```

```
                } else {
                    abstModMat[k,z] = 1
                }
            }

            k <- k+1
        }
    }

    abstModMat <- abstModMat[!duplicated(abstModMat),]
    return(abstModMat)
}


getStructureMode <- function(precondMat,implicMat){

    # GIVEN: a precondition matrix and the linked implication matrix
    # DO: return a structure of modes  based on constraints

    # Generate abstract modes

    abstModMat <- getAbstModes(precondMat)
    colnames(abstModMat) <- colnames(precondMat)

    concatAbstModMat <- abstModMat

    while(!is.null(nrow(abstModMat)))
    {
        concatAbstModMat <- rbind(concatAbstModMat,abstModMat)
        abstModMat <- getAbstModes(abstModMat)
    }
    rownames(concatAbstModMat) <- paste("Abstract mode ",1:nrow(concatAbstModMat))
    concatModes <- rbind(precondMat,concatAbstModMat)
    concatModes <- concatModes[!duplicated(concatModes),]

    # Remove intermediary abstract modes, so that only those directly linked to an original
    mode remain

    impStruct <- getImplicMatrix(concatModes)
    redImpStruct <- transitive.reduction(impStruct)

    nbModes <- nrow(precondMat)


    absModeImplicToKeep <- vector()
    absModeImpliedToKeep <- vector()

    for(i in c((nbModes+1):nrow(redImpStruct)))
    {
        if(any(c(redImpStruct[i,(1:nbModes)])==1))
        {
            absModeImplicToKeep <- c(absModeImplicToKeep,i)
        }
        if(any(c(redImpStruct[(1:nbModes),i])==1))
        {
            absModeImpliedToKeep <- c(absModeImpliedToKeep,i)
        }

    }

    absModeToKeep <- union(absModeImplicToKeep,absModeImpliedToKeep)

    toKeep <- c((1:nbModes),absModeToKeep)

    concatModes <- concatModes[toKeep,]

    return(concatModes)
}


##################################################
```

```
    # FUNCTIONS: STATES

###############################################


checkConstraints <- function(constMat,stateSep){

    # GIVEN: a constraint matrix and a vector indicating its separation in state types
    # DO: check that each state value in the constraint matrix is compatible with at least
    one value of every other type of state

    constMatParts <- partition.matrix(constMat, colsep=stateSep )

    for (i in c(1:(nrow(constMat))))
    {
        for (j in c(2:(length(constMatParts))))
        {
            if (any(c(constMatParts[[j]][i,])==1)==FALSE)
            {
                cat("Missing possible case between state value ", row.names(constMat)[i],"
                and set of state values ",colnames(constMatParts[[j]]), "\n")
            }
        }
    }
}

getFalseConfListByComplexConst <- function(confList,complexConst, stateSep){

    # GIVEN: a list of state values configurations, a matrix expressing complex
    constraintsand a vector indicating its separation in state types
    # DO: return incompatible configurations according to complex constraints

    comConstP <- partition.matrix(complexConst, colsep=stateSep)

    falseConfList <- vector()

    for (i in c(1:nrow(complexConst)))
    {
        constVals <- vector()
        for (j in c(1:ncol(complexConst)))
        {
            if (complexConst[i,j]==1)
            {
                constVals <- c(constVals, j)
            }
        }

        for (k in c(1:length(confList)))
        {
            occur <- match(confList[[k]], V(gConf))
            if(length(intersect(constVals, occur)) == typesConst)
            {
                falseConfList <- c(falseConfList, k)
            }
        }
    }

    return(falseConfList)
}

checkStateValueInConf <- function(stateValues, confList){

    # GIVEN: a list of states values and a list of possible configurations
    # DO: check that every state value exists in at least one possible configuration, and
    return the list of unused states

    unusedStates <- stateValues

    for (i in c(1:length(confList)))
    {
```

```
        unusedStates <- setdiff(unusedStates , confList[[i]])
    }

    if(length(unusedStates) != 0)
    {
        for(i in c(1:length(unusedStates)))
        {
            cat("No configurations using state value: ",V(gConf)[unusedStates[i]]$name, "\n")
        }
    }

    return(unusedStates)
}

checkNearSolutionsSimpConst <- function(constMat,unusedStates){

    # GIVEN: a simple constraint matrix and a lsit of unused state values in the possible
    configuraiton it expresses
    # DO: Check types of state that are lacking in configurations that are nearest to a
    solution for state values that lack a possible configuration.

    for (i in c(1:length(unusedStates)))
    {

        incompStates <- vector()
        for (j in c(1:ncol(constMat)))
        {
            if(configs[unusedStates[i],j]==0)
            {
                incompStates <- c(incompStates, j)
            }
        }

        reduceConf <- submatrix(constMat, incompStates, incompStates)
        redConfG <- as.undirected(graph.adjacency(reduceConf))
        redConfSols <- max_cliques(redConfG, min = 14)
        if (length(redConfSols) == 0)
        {
            cat("Simple constraint: more than one missing state type for a configuration
            with", row.names(configs)[unusedStates[i]], "\n")
        } else {
            redSolsF <- setdiff(V(redConfG), Reduce(union, redConfSols))
            cat("Simple constraint: unresolved state value for a configurations with ",
            row.names(configs)[unusedStates[i]],": ", row.names(reduceConf[redSolsF,]), "\n")
        }
    }
}

################################################

    # FUNCTIONS: PRECONDITIONS

################################################


checkPrecond <- function(precondMat, stateSep){

    # GIVEN: a precondition matrix and a vector indicating its separation in state types
    # DO: Check that each precondition has at least one authorized value for every type of
    state

    parts <- partition.matrix(precondMat, colsep=stateSep)

    for (i in c(1:(nrow(precondMat))))
    {
        for (j in c(1:(length(parts))))
        {
            if (any(c(parts[[j]][i,])==1)==FALSE)
            {
                cat("Missing authorized value for use case", row.names(precond)[i], ": ",
                colnames(parts[[j]]), "\n")
```

```
            }
        }
    }
}



# Checking the existence of solutions for each action preconditions

authStates <- vector("list", nrow(precond))

errSol <- vector()

possConf <- vector("list", nrow(precond))

for (i in c(1:nrow(precond)))
{

    # Checking that each action admits at least one solution

    istates <- vector()

    for (k in c(1:ncol(precond)))
    {
        if (precond[i,k] == 1)
        {
            istates <- c(istates, k)
        }
    }

    authStates[[i]] <- istates

    iconf <- submatrix(configs, istates, istates)
    iconfG <- as.undirected(graph.adjacency(iconf))
    iconfSols <- cliques(iconfG, min = 15)

    if (length(iconfSols) == 0)
    {
        cat("Simple constraints: no solutions for action ", row.names(precond)[i], "\n")
        errSol <- c(errSol, i)
    } else {

        # Filter configurations according to complex constraints

        comConstP <- partition.matrix(complexConst, colsep=sep)
        falseConfList <- vector()

        for (s in c(1:nrow(complexConst)))
        {
            typesConst = 0
            for (d in c(1:length(comConstP)))
            {
                if (any(comConstP[[d]][s,])==1)
                {
                    typesConst = typesConst + 1
                }
            }
            constVals <- vector()
            for (f in c(1:ncol(complexConst)))
            {
                if (complexConst[s,f]==1)
                {
                    constVals <- c(constVals, f)
                }
            }

            for (g in c(1:length(iconfSols)))
            {
                occur <- match(iconfSols[[g]], V(gConf))
                if(length(intersect(constVals, occur)) == typesConst)
                {
```

```
                    falseConfList <- c(falseConfList, g)
                }
            }
        }
        list.remove(iconfSols, falseConfList)

        if (length(iconfSols) == 0)
        {
            cat("Complex constraints: no solutions for use case ", row.names(precond)[i],
            "\n")
            errSol <- c(errSol, i)
        } else {

            # stock possible configurations for each action

            iconfSols2 <- vector("list", length(iconfSols))

            for (z in c(1:length(iconfSols)))
            {
                iconfSols2[[z]] <- istates[iconfSols[[z]]]
                # iconfSols2[[z]] <- colnames(precond[,istates[iconfSols[[z]]]])
            }
            possConf[[i]] <- iconfSols2
        }
    }
}



################################################

    # CHECK CONSTRAINTS

################################################



# Checking that each state value is compatible with at least one value of every other type
of state

checkConstraints(configs, stateSep)

# Checking the existence of configurations for each state value according to simple
constraints

unusedStates <- checkStateValueInConf(V(gConf), confList)

# Checking the existence of configurations for each state value according to complex
constraints

falseConfList <- getFalseConfListByComplexConst(confList,complexConst, stateSep)

if (length(falseConfList) != 0){
    compConfList <- confList
    list.remove(compConfList, falseConfList)

    unusedStates2 <- V(gConf)

    unusedStates2  <- checkStateValueInConf(unusedStates2, compConfList)
}

################################################

    # CHECK PRECONDITIONS

################################################

# Checking that each use case has at least one authorized value for every type of state

checkPrecond(precond, stateSep)
```

```
# Checking the existence of solutions for each action preconditions


authStates <- vector("list", nrow(precond))

errSol <- vector()

possConf <- vector("list", nrow(precond))

for (i in c(1:nrow(precond)))
{

    # Checking that each action admits at least one solution

    istates <- vector()

    for (k in c(1:ncol(precond)))
    {
        if (precond[i,k] == 1)
        {
            istates <- c(istates, k)
        }
    }

    authStates[[i]] <- istates

    iconf <- submatrix(configs, istates, istates)
    iconfG <- as.undirected(graph.adjacency(iconf))
    iconfSols <- cliques(iconfG, min = 15)

    if (length(iconfSols) == 0)
    {
        cat("Simple constraints: no solutions for action ", row.names(precond)[i], "\n")
        errSol <- c(errSol, i)
    } else {

        # Filter configurations according to complex constraints

        comConstP <- partition.matrix(complexConst, colsep=sep)
        falseConfList <- vector()

        for (s in c(1:nrow(complexConst)))
        {
            typesConst = 0
            for (d in c(1:length(comConstP)))
            {
                if (any(comConstP[[d]][s,])==1)
                {
                    typesConst = typesConst + 1
                }
            }
            constVals <- vector()
            for (f in c(1:ncol(complexConst)))
            {
                if (complexConst[s,f]==1)
                {
                    constVals <- c(constVals, f)
                }
            }

            for (g in c(1:length(iconfSols)))
            {
                occur <- match(iconfSols[[g]], V(gConf))
                if(length(intersect(constVals, occur)) == typesConst)
                {
                    falseConfList <- c(falseConfList, g)
                }
            }
        }
        list.remove(iconfSols, falseConfList)
```

```
        if (length(iconfSols) == 0)
        {
            cat("Complex constraints: no solutions for use case ", row.names(precond)[i],
            "\n")
            errSol <- c(errSol, i)
        } else {

            # stock possible configurations for each action

            iconfSols2 <- vector("list", length(iconfSols))

            for (z in c(1:length(iconfSols)))
            {
                iconfSols2[[z]] <- istates[iconfSols[[z]]]
                # iconfSols2[[z]] <- colnames(precond[,istates[iconfSols[[z]]]])
            }
            possConf[[i]] <- iconfSols2
        }
    }
}


# Cleaning preconditions by removing unused values (values that don't/cannot appear in
possible configurations)

cleanPrecond <- matrix(0, nrow(precond), ncol(precond))
rownames(cleanPrecond) <- rownames(precond)
colnames(cleanPrecond) <- colnames(precond)
diffPrecond <- cleanPrecond

for (i in c(1:nrow(precond)))
{
    if (!is.null(possConf[[i]]))
    {
        iStateUnion <- Reduce(union, possConf[[i]])
        valDiff <- setdiff(authStates[[i]], iStateUnion )
        cleanPrecond[i,iStateUnion]=1
        diffPrecond[i,iStateUnion]=1
        diffPrecond[i,valDiff]=2
    }
}


# Exporting new precondition matrix and one highlighting differences compared to the original

write.csv(cleanPrecond, "cleanedPreconditions.csv")
write.csv(diffPrecond, "DeletedPreconditions.csv")

# Find all functions that a given possible configurations enable

actInConf <- vector("list", length(confList))

for (i in c(1:length(confList)))
{
    vertConf <- Reduce(union, confList[[i]])
    for (j in c(1:nrow(precond)))
    {

        if(length(setdiff(vertConf,authStates[[j]])) == 0)
        {
            actInConf[[i]] <- c(actInConf[[i]],j)
        }
    }
}

# Write and export the coexistence matrix

coexist <- matrix(0, nrow(precond), nrow(precond))

for (i in c(1:length(actInConf)))
{
```

```
    for(j in c(1:length(actInConf[[i]])))
    {
        for (k in c(j:length(actInConf[[i]])))
        {
            m <- actInConf[[i]][j]
            n <- actInConf[[i]][k]
            if(m != n)
            {
                coexist[m,n] = 1
                coexist[n,m] = 1
            }
        }
    }
}

rownames(coexist) = rownames(precond)
colnames(coexist) = rownames(precond)

write.csv(coexist, "MatCoexistence.csv")

# Write and export the implication matrix

implic <- matrix(0, nrow(precond), nrow(precond))

for (i in c(1:(nrow(precond))))
{
    for (j in c(1:nrow(precond)))
    {
        if((i != j)&(length(setdiff(authStates[[i]],authStates[[j]])) == 0))
        {
            implic[i,j] = 1
        }

    }
}

rownames(implic) = rownames(precond)
colnames(implic) = rownames(precond)

reducedImplic <- transitive.reduction(implic)

write.csv(implic, "MatImplications.csv")
###############################################

    # DEFINE MODES STRUCTURE

###############################################

# Generate modes englobing each scope (scope modes)

modesScope <- vector("list", length(scopes))
modScoMat <- matrix(0L, nrow = length(scopes),ncol = ncol(precond))
modNames <- vector()
mode(modScoMat) <- "integer"
colnames(modScoMat) <- colnames(precond)

for(i in c(1:length(scopes)))
{
    mode <- integer(ncol(precond))
    for(j in c(1:ncol(precond)))
    {
        if (any(c(scopes[[i]][,j])==1))
        {
            mode[j] = 1
        }
    }
    modesScope[[i]] <- mode
    modScoMat[i,] <- mode
    modNames <- c(modNames, paste("Mode Scope", i))
}
```

```
rownames(modScoMat) <- modNames

# Remove eventual duplicates in scopes modes

modScoMatRaw <- modScoMat
modScoMat <- modScoMat[!duplicated(modScoMat),]

# Calcute the implication matrix for scope modes

impMod <- getImplicMatrix(modScoMat)

# Simplify the implication matrix using transitivity

redImpMod<- transitive.reduction(impMod )

# Calcute matrix of scope modes and abstract modes

constTreeMod <- getStructureMode(modScoMat,redImpMod)

# Calculate the implication matrix for the full list of modes

impStruct <- getImplicMatrix(constTreeMod)

# Simplify the implication matrix using transitivity

redImpStruct <- transitive.reduction(impStruct)

# Plot the graph of the structure of modes

redImpStructG <- as.directed(graph.adjacency(redImpStruct))
laySugi <- layout_with_sugiyama(redImpStructG, hgap=80)
plot.igraph(redImpStructG, layout = laySugi$layout)

###############################################

    # CASE STUDY: SCOPE 6

###############################################


modScoSixPrec <- modScoMat[c(7:12),]
impScoSixPrec <- getImplicMatrix(modScoSixPrec)
redImpScoSix <- transitive.reduction(impScoSixPrec)
constTreeScoSixMod <- getStructureMode(modScoSixPrec,redImpScoSix)
impStructScoSix <- getImplicMatrix(constTreeScoSixMod)
redImpStructScoSix <- transitive.reduction(impStructScoSix)

redImpStructScoSixG <- as.directed(graph.adjacency(redImpStructScoSix))
laySugi <- layout_with_sugiyama(redImpStructScoSixG, hgap=80)
plot.igraph(redImpStructScoSixG, layout = laySugi$layout)


###############################################

    # CASE STUDY: USE CASES MODES

###############################################

# Remove eventual duplicates in use case modes

precond <- precond[!duplicated(precond),]

# Calcute the implication matrix for use case modes

impUseCase <- getImplicMatrix(precond)

# Simplify the implication matrix using transitivity

redImpUC <- transitive.reduction(impUseCase)

# Calcute matrix of use case modes and abstract modes
```

```
constTreeUC <- getStructureMode(precond,redImpUC)

# Calculate the implication matrix for the full list of modes

impStructUC <- getImplicMatrix(constTreeUC)

# Simplify the implication matrix using transitivity

redImpStructUC <- transitive.reduction(impStructUC)

# Plot the graph of the structure of modes

redImpStructUCG <- as.directed(graph.adjacency(redImpStructUC))
laySugi <- layout_with_sugiyama(redImpStructUCG, hgap=80)
plot.igraph(redImpStructUCG, layout = laySugi$layout)




# Get the list of types of states values constraining the scope modes

modScoPart <- partition.matrix(modScoMat, colsep=stateSep)

redStaTyp <- match(filterStateDep(modScoPart),colnames(precond))

modScoMatDep <- modScoMat[,redStaTyp]

# Calcute the implication matrix for scope modes

impMod <- getImplicMatrix(modScoMat)



# Trace the graph of implications between scope modes

redImpModG <- as.directed(graph.adjacency(redImpMod))
laySugi <- layout_with_sugiyama(redImpModG, hgap=80)
plot.igraph(redImpModG, layout = laySugi$layout)

#
modScoSixPrec <- modScoMat[c(7:12),]
impScoSixPrec <- getImplicMatrix(modScoSixPrec)
redImpScoSix <- transitive.reduction(impScoSixPrec)

redImpScoSixG <- as.directed(graph.adjacency(redImpScoSix))
laySugi <- layout_with_sugiyama(redImpScoSixG, hgap=80)
plot.igraph(redImpScoSixG, layout = laySugi$layout)
```