Category theory for consistency between multilevel system modeling (MBSE) and safety (MBSA)

Julien Vidalie

université
PARIS-SACLAY

Category theory for consistency between multilevel system modeling (MBSE) and safety (MBSA)

Théorie des Catégories pour la cohérence des modèles multi-niveaux systèmes (MBSE) et sûreté de fonctionnement (MBSA)

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°573 interfaces : matériaux, systèmes, usages (INTERFACES)
Spécialité de doctorat: Ingéniérie des systèmes complexes
Graduate School : Sciences de l'ingénierie et des systèmes,
Référent : CentraleSupélec

Thèse préparée dans l'unité de recherche Laboratoire Quartz—ISAE Supméca, sous la direction de **Jean-Yves CHOLEY**, Professeur, le co-encadrement de **Faïda MHENNI**, maîtresse de conférences, du co-encadrant ou de la co-encadrante et le co-encadrement de **Michel BATTEUX**, Docteur

Thèse soutenue à Paris-Saclay, le 08 février 2023, par

Julien VIDALIE

Composition du jury
Membres du jury avec voix délibérative

Frédéric KRATZ Professeur des universités, INSA Centre Val de Loire	Président
Pierre SAQUI-SANNES Professeur des universités, ISAE-SUPAERO	Rapporteur & Examinateur
Abdelfattah MLIKA Professeur des universités, Université de Sousse	Rapporteur & Examinateur
Claude BARON Professeure des universités, LAAS-CNRS	Examinatrice
Antoine RAUZY Professeur des universités, Norwegian University of Science and Technology	Examinateur

THESE DE DOCTORAT

NNT : 2023UPAST020

Summary

List of Figures

List of Tables

ACKNOWLEDGEMENTS

> Write down everything that happens in the story, and then in your second draft make it look like you knew what you were doing all along
>
> Neil Gaiman

La thèse n'est pas un exercice de tout repos. Elle confronte le doctorant à des défis techniques et scientifiques, mais également humains, face auxquels il se sent parfois assez seul. En plus des difficultés inhérentes à ce travail, le contexte difficile qui l'a accompagné, avec notamment l'apparition de la pandémie du COVID, n'ont pas fait de mon doctorat une période facile. Cependant j'ai toujours pu compter sur mon entourage, proche et professionnel pour me soutenir face aux difficultés rencontrées. Il convient donc de remercier tous ceux qui ont, d'une manière ou d'une autre, contribué à la réussite de ces travaux.

Cette thèse a été accueillie par l'IRT SystemX. Je suis reconnaissant à cet institut ainsi qu'à ses membres pour la confiance qui m'a été accordé en me proposant ces travaux. Elle a été financée dans le cadre du projet S2C, grâce à la participation des partenaires industriels et académiques du projet ainsi que le soutien de l'ANR. Cela m'a permis de travailler dans un environnement professionnel de grande qualité.

J'ai ainsi eu la chance de côtoyer l'équipe S2C, Anouk Dubois, Hanane Fadiaw, Stephen Creff, Sylvain Champion, Colin Poubel, qui ont été des collègues formidables, m'ont énormément appris, et avec lesquels j'ai également partagé de bons moments de convivialité.

L'IRT SystemX accueille de nombreux thésards, qui constituent aujourd'hui une communauté unie. Lorsque je suis arrivé les doctorants avaient encore relativement peu de contacts entre eux en dehors des projets. Pascal Un et

Clarisse Lawson ont alors pris l'initiative de nous réunir et de créer l'association DOCSX qui a permis le rapprochement de doctorants de projets differents ayant des spécialités diverses. Je tiens à les remercier pour cela. Je remercie également l'ensemble des doctorants de l'IRT pour tous les bons moments passés ensembles, pour toutes les discussions scientifiques et personnelles. Merci également d'avoir supporté et répondu à mes sollicitations pour le séminaire des doctorants de l'association pour l'organisation duquel je pense que ma succession est assurée. Je pense en particulier à Maria Hanini, Victor Pellegrain, Tjark Gall, Mariana Reyes, Emmanuel Meunier avec qui nous avons constitué deux bureaux successifs de l'association. J'ai également eu la chance de rencontrer Natkamon Tovanitch. Thank you, my friend, for all the conversations we had. Thank you for having me discover Bo Bun, for showing me that even through the difficulties the Ph.D. was not impossible to finish and thank you of course for the beautiful cup that I preciously keep.

Durant cette thèse, j'ai aussi été accueilli par le laboratoire Quartz à l'ISAE Supméca. Si j'ai été moins présent en ce lieu, j'y ai cependant également fait de belles rencontres. Je remercie donc toute l'équipe IS2M, qui m'a accueilli, et qui, pour certains membres de l'équipe avaient également participé à ma formation d'ingénieur avant la thèse. Mes travaux s'appuient sur l'utilisation de la théorie des catégories. Ce sujet est apparu dans l'équipe au travers du travail mathématique de Stéphane Dugowson. Ce sont les discussions menées avec ce dernier qui m'ont permis d'avoir une meilleure compréhension de la théorie des catégories et de son potentiel. Les conversations avec Régis Plateaux, Olivia Penas, Nourhene Abdeljabbar et Mouna Fradi m'ont également beaucoup aidé. De plus j'ai reçu, tout au long de ces trois années l'aide précieuse de Christel Compagnon dans toutes mes démarches administratives. Par son travail au laboratoire et sa personnalité elle apporte un lien fort entre les personnels de Supméca, notamment par l'organisation des séminaires d'hiver « Les fermes du Vercland », dont je n'ose imaginer le coût en temps. L'évocation de ce séminaire me permet également d'avoir une pensée pour toutes les personnes que j'apprécie beaucoup à Supméca, que j'ai eu la chance de rencontrer grâce à ces événements, notamment Salem qui m'a appris à skier.

Je remercie toutes les personnes qui ont anonymement permis à ce travail de thèse de s'effectuer dans les meilleures conditions : Les personnels de ménage, les membres des services informatiques, RH et toute autre fonction support que j'aurai maladroitement oublié, les conducteurs de bus et de trains, les personnels du restaurant inter entreprise. Sans toutes ces personnes les chercheurs ne seraient pas en mesure de travailler et le contenu de ce manuscrit n'aurait jamais vu le jour.

J'ai été également accompagné par de nombreux amis que je ne saurai assez remercier pour leur soutien et le temps passé ensemble. Alexandre, Donatien, Cédric, vous avez été des piliers pour moi sur lesquels j'ai toujours

pu m'appuyer dans les moments difficiles, j'ai la chance d'avoir pu compter sur vous pour de nombreuses relectures et autres conseils scientifiques, mais également pour des défis sportifs ou ludiques. Michel, en sus d'être notre expert en informatique, tu es un ami sur lequel on peut toujours compter et je t'en suis extrêmement reconnaissant. Louis, Johanna, Cindy, Lucas, Julie, Charles, merci d'être de si bons amis. Valoux, Eloïse, Marie, Arnaud, Romain, Quentin, Maxime, Cécile, merci pour tous les bons moments passés sur discord ainsi que nos quelques entrevues IRL, vous avez été les soleils de l'année 2022, qui a été la plus difficile de ma vie, je ne pense pas que j'aurai pu aller jusqu'au bout de ces travaux sans vous.

Je tiens également à remercier les membres de mon jury de thèse, Pierre de Saqui-Sannes, Abdelfattah Mlika, Claude Baron, Antoine Rauzy et Frédéric Kratz d'avoir accepté d'évaluer mes travaux.

Je ne saurai exprimer toute la reconnaissance que j'ai envers mes encadrants, Michel Batteux et Faïda Mhenni, ainsi que mon directeur de thèse Jean-Yves Choley. Vous m'avez fait confiance en me chargeant de ce travail de thèse. J'ai énormément appris grâce à vous, aussi bien sur le plan scientifique qu'humain. J'espère faire honneur dans la suite de ma carrière à l'opportunité que vous m'avez offerte ainsi qu'à la qualité de votre encadrement.

Je réserve les derniers mots de ces longs remerciements à ma famille. J'ai la chance d'avoir deux sœurs formidables, Camille et Marie, que j'aime énormément. Mes parents ont toujours été un soutien sans faille et c'est grâce à eux que j'ai pu avancer dans mes études jusqu'à être accepté en thèse. Mes nombreux cousins, cousines, oncles et tantes que je vois malheureusement moins souvent, mais avec qui je partage toujours des moments de qualité. Ce sont votre présence, vos encouragements, vos conseils, qui m'ont permis de devenir l'homme que je suis aujourd'hui, et de mener à bien ce beau projet qu'est la thèse.

INTRODUCTION

1 Need for models synchronization and a formal framework

During recent decades, systems have become more complex as they evolved towards having multiple features due to the rise of mechatronic design. This complexity leads to a need to create numerous models during system design [22]. These models are used to represent the product for the sake of diverse modeling intents, thus allowing the analysis of specific properties.

Among these models, we find systems engineering models used to apprehend complexity in the system. Systems engineering encompasses many models, including requirement and architecture, safety, and multi-physics models. These models are written by different people, using various tools and formalisms. This induces an independence between the models that can lead to problems with consistency between the models. As several models represent one system, we fear some models may contain errors, making them incompatible with the designed system. Such inconsistencies are concerning in the case of safety-critical systems and their safety models, and manual reviews of the models is time expensive. These systems include cars, trains, aircrafts, nuclear plants and others. An undetected error in the safety model could lead to not considering some of the system's potential failure events. Such errors would then result in the system being unsafe for the user. When systems have to be certified to be accepted into the market, e.g., for aircrafts, the safety assessment methodology needs to be accepted by safety authorities. This requires great trust in the safety assessment process and, therefore, the consistency of the models.

This thesis takes place in the S2C (System to Safety Continuity) project at the Institut de Recherche Technologique (IRT) SystemX. The S2C project is a four-year project launched in April 2019, in collaboration between IRT SystemX

(Palaiseau) and IRT St-Exupéry (Toulouse). The project involves other partners, which are institutional (DGA), academic (IRIT, LAAS-CNRS, ONERA, Supméca), and industrial (Airbus Defence and Space, APSYS, Dassault Aviation, LGM, Liebherr, MBDA, Samares Engineering, Thales). The project seeks to address continuity between the System Engineering and the Safety Assessment processes. It is divided into four tasks:

- Task 1: Global Process and lifecycle
- Task 2: Consistency methodology at a given system level
- Task 3: Consistency methodology between system level
- Task 4: Safety modeling methodology

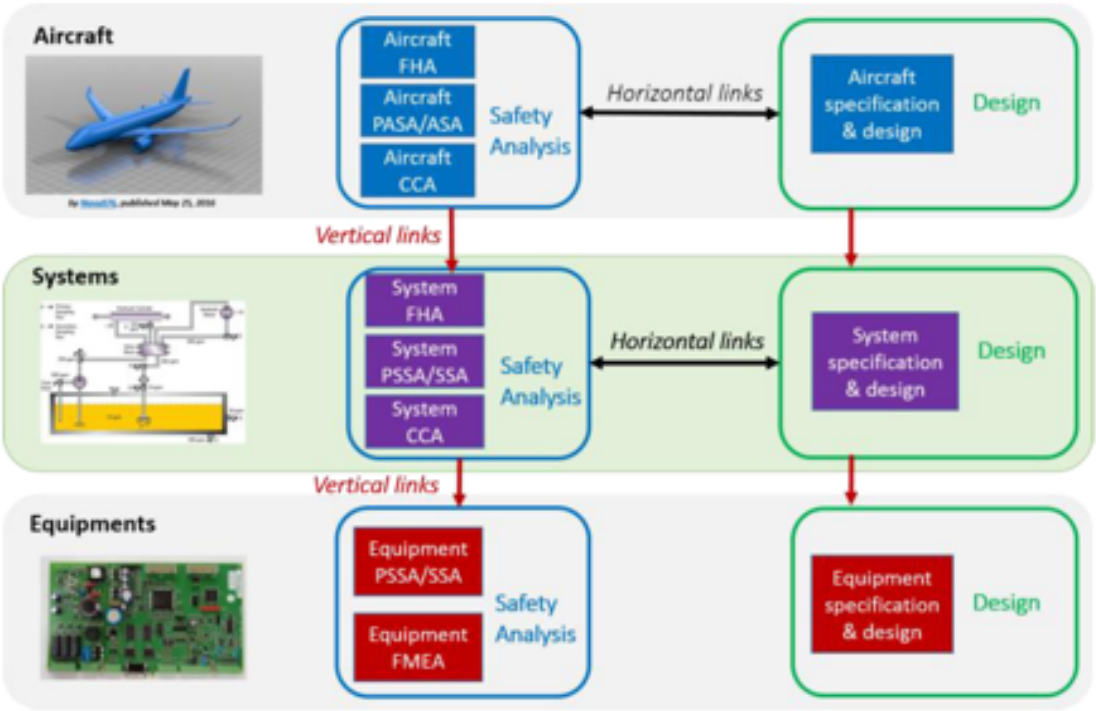


Figure 1: Positioning of the S2C project (Illustration from the S2C project contractual definition)

This thesis is in relation to tasks 2 and 3. Whereas the project encompasses all System Engineering and Safety Assessment disciplines, the thesis focuses on Model-Based System Engineering and Model-Based Safety Assessmment. These disciplines address the increase in complexity of systems, and even though they are not yet widely adopted in industry, their use is rising.

Dassault Aviation's Falcon 7X is an excellent example of this phenomenon, as its electric commands were the first aeronautic system to be certified using MBSA tools [112].

The systems engineering community proposes different methodologies such as model federation [54] or proof theory approach [45] to assert and maintain consistency between these models to face the challenge of model consistency. Amongst these methodologies, we also find synchronization methodologies [79]. [79] defines synchronization methodologies with three steps:

1. Abstraction: The models are translated to a common formalism.
2. Comparison: The models are compared in this common formalism.
3. Concretization: The comparison results are carried back to the original models.

Although these methodologies exist and provide means to minimize the number of contradictions between the models, there is no formal definition of consistency widely accepted in the community. We primarily understand consistency as the absence of contradiction between the models, but this is not a criterion that single-handedly allows a methodology's efficiency to be demonstrated. Thus, this thesis proposes a formal definition of a binary consistency relation and a mathematical framework within which we can specify a synchronization methodology: the SmartSync methodology.

This thesis work is hosted by the IS2M (Ingénierie des Systèmes Mécatroniques et Multi-physiques) team in Quartz Laboratory at ISAE-Supméca. Before working on this thesis, I was an engineering student in ISAE Supméca; at the end of my term there, I underwent a six months internship at Safran Tech within the Complex System Engineering team.

During this internship, I worked on the topic of MBSE to MBSA continuity from a different angle. Using Safran's Cameo System Modeler plugin, we developed an extension that allowed us to use the MBSE model. It could also be overloaded with safety artifacts, to generate AltaRica models. Although we had a working solution at the end of my internship, some points made me unsatisfied with this tool and led me to continue the work with this thesis. The first point is that this plugin heavily depended on Safran's Architecture grid. This plugin could only be used with a model developed by Safran Engineers, using the Safran plugin and the Cameo System Modeler tool. MBSE models come with various tools and architecture grids, so we only tackled a specific case of the problem. The second point is that, although the MBSA model we obtained was consistent with the architecture by construction, the safety analysts would heavily modify it to make it fit to their specific need. The MBSE and MBSA models are meant to evolve and for engineers to iterate on them. This means we had consistent models at the beginning of the process, but we had no way to verify

that they remained consistent. This is why we need synchronization, to get the models consistent in the first place and keep them consistent for the next steps.

2 The S2ML+Cat mathematical framework

In this work, we propose to use category theory to specify model synchronization through a mathematical framework. SmartSync [12] is a synchronization methodology that was developped by a team that includes the supervisors of this thesis, which uses S2ML (System Structure Modeling Language [14]) as a comparison language.

The mathematical framework associated to SmartSync defines S2ML models as quintuples, in this work we intend to give a new theoretical perspective over this methodology, through another theory. This is why we propose a new mathematical framework, where we explore the idea to specify SmartSync and S2ML models through category theory.

In S2ML, the structure of the model is represented using three concepts: blocks, ports, and connections. These concepts are sufficient to represent any structural model [100]. To define the categorical framework, we provide a categorical representation of the S2ML concepts of block, port, and connection. From these mathematical definitions, we are able to build a categorical representation of models. We then provide the proof that we can associate these representations to the original S2ML mathematical framework. Thus, we are able to propose a definition of consistency between these categorical specifications of structural models in the context of SmartSync. We apply this definition to the SmartSync methodology through the study of model consistency in the design workflow of an autonomous critical system, a fixed-wing blood delivery drone.

Synchronization methodologies are based on the abstraction, comparison, and concretization steps. In this work, we give a formal meaning to the comparison step of SmartSync through the mathematical framework. To that intent, we provide a way to specify structural models as categories by representing model elements as category objects and their relationships as morphisms. Between such mathematical objects, we can define consistency relations, allowing us to identify the models' common skeletons. Therefore, we are able to detect differences between the models. Compared to the SmartSync methodology, we add a formal layer that allows mathematical proofs and computations. This layer does not affect the end user but it allows to demonstrate the exact comparison results that will be output by the methodology, not only for one case study but in the general case. This layer is illustrated in Fig. 2.

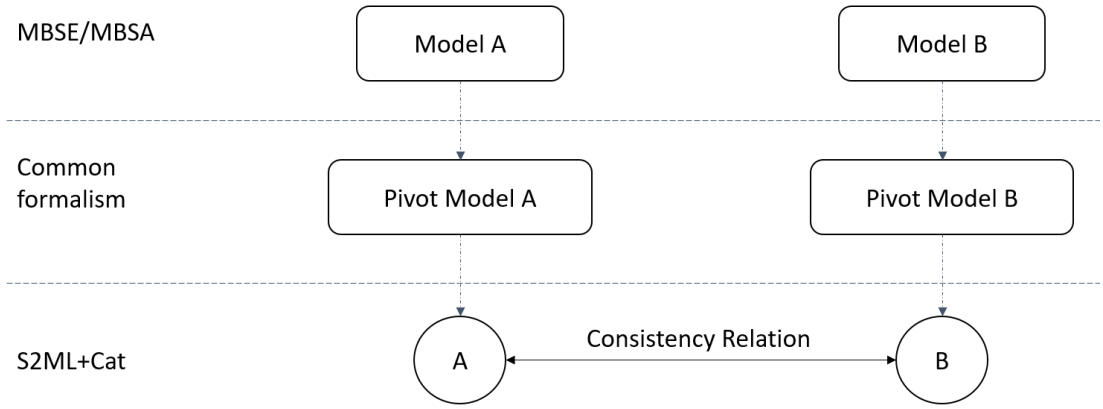


Figure 2: Approach of the comparison with S2ML+Cat

3 Summary of the thesis

The rest of this thesis is composed of five chapters:

- **Chapter 1 "State of The Art"**
  This first chapter is a state-of-the-art of the notions related to the thesis. It introduces System Engineering and Safety assessment major concepts. It describes the existing work around models and their consistency. Finaly, it introduces category theory.

- **Chapter 2 "A case study: The landing gear"**
  In this second chapter, we describe the modeling of a landing gear system. MBSE and MBSA models are designed with the SysML formalism and AltaRica 3.0 modeling language respectively. We then compare and synchronize them using the SmartSync methodology. This case study is then used as a working example to explain the notions we introduce in the following chapters.

- **Chapter 3 "MBSE and MBSA System Models Differences"**
  In this third chapter, we study the differences between MBSE and MBSA models. We propose a typology of these differences, which allows us to distinguish between inconsistencies and other kinds of differences that should not be eliminated. We also study the feasibily and relevance of structural consistency between MBSE and MBSA state machines.

- **Chapter 4 "The S2ML+Cat framework"**
  This fourth chapter proposes a mathematical framework, S2ML+Cat, based on category theory. This framework allows for formal representation of structural models, based on the S2ML language. We define binary

consistency relations between two structural models. We also discuss the possibility of using the S2ML+Cat framework to simplify the synchronization whenever previous versions of the models have already been synchronized, and a change history exists.

- **Chapter 5 "Application example: The blood delivery drone"**
  This fifth and final chapter presents the modeling of a second case study, the blood delivery drone. We use this case study to apply the mathematical framework from chapter 4. We design MBSE, MBSA, multiphysics, and scenario models of the drone. We then operate a SmartSync comparison of each domain-specific model with the MBSE model. Finally, we express these comparisons within S2ML+Cat to explain how the SmartSync methodology can be modeled with S2ML+Cat.

CHAPTER 1

STATE OF THE ART

Before we dive into the depths of model comparison and mathematical analysis, it is helpful to remind the basic and less basic concepts used in this thesis.

This chapter is divided into four sections. First, we discuss systems engineering, followed by safety assessment. We then focus on models and their comparison, especially with model synchronization. We finally give an introduction to the bases of category theory.

1.1 Systems Engineering

The subject of this thesis is consistency between Model-Based Systems Engineering (MBSE) and Model-Based Safety Assessment MBSA models. In this first section of the state of the art we introduce Systems Engineering. Systems Engineering is commonly defined as a multidisciplinary field that focuses on the design of complex systems.

1.1.1 History

Systems engineering emmerged through the work of NASA (National Aeronautics and Space Administration) and USAF (United States Air Force) in the 60s [73]. New approaches were developed in the context of the Apollo program to offer possibilities of an in-depth, systematic study of the systems.

The Apollo 1 disaster was one of the events that enlightened the need for better study and traceability over complex systems design. The integration of designs from the Mercury and Gemini space programs led to the use of a hatch design that made it difficult to exit the module. At the same time, no possibility of a pre-flight incident was taken into account in the safety studies [5]. During a ground test, an incident caused a fire in the module, leading to the death of

flight commander Gus Grissom, astronaut Edward White, and astronaut Roger Chaffee. A system approach could have avoided this disaster by considering the astronauts' needs in the capsule.

Although the beginning of Systems engineering is often associated with the Apollo program, the term was, in fact, first used by Bell Telephone Laboratories in the 1940s [107].

The International Council on Systems Engineering (INCOSE) was founded in 1990. This organization promotes international collaboration in Systems engineering through publications, formation and certification of systems engineers, and centralization of the knowledge over Systems engineering.

The Association Française d'Ingénierie Système (AFIS) was founded in 1998 with similar objectives. The AFIS works with INCOSE to propagate System Engineering in France.

Together, INCOSE and AFIS maintain the SE Handbook [63] and SEBoK (Systems Engineering Body of Knowledge) [5].

1.1.2 Definition of Systems engineering

Multiple definitions of Systems engineering have been given over time by INCOSE, AFIS, and other members of the System Engineering community. A first and basic definition is the following one, coming from the INCOSE [63].

> "An interdisciplinary approach and means to enable the realization of successful systems"

This definition was completed in the 2012 edition of the SEBoK [64].

> "Systems Engineering (SE) is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on holistically and concurrently understanding stakeholder needs; exploring opportunities; documenting requirements; and synthesizing, verifying, validating, and evolving solutions while considering the complete problem, from system concept exploration through system disposal."

The INCOSE official definition, however, was recently updated [5].

> "A transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods."

As a recap, systems engineering has the role of fedarating all engineering disciplines in order to design systems that meet all stakeholder requirements [79].

1.1.3 Design cycle, the V-Model

The V-Model is omnipresent in System development. It is said to have emerged in the 1960's [117], but only became an industry-standard in the 1980's [23]. The V-Model is built as a more robust alternative to the waterfall model [103].

The idea of the V-Model is to have two separate branches in the iterative design process. The first branch is a partitioning phase, in which requirements and a system are specified. The second branch is an integration phase, in which parts are integrated and validated. Both branches are linked through a verification and validation process. These branches can be seen as descending and ascending respectively, as depicted in Fig. 1.1.

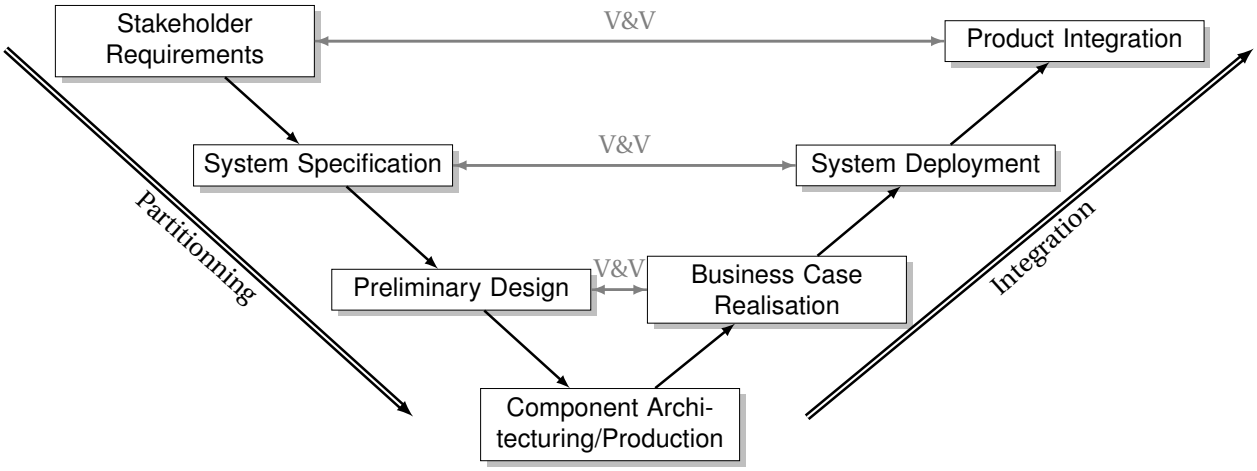


Figure 1.1: The V-Model

The V-Model is an iterative approach, as Verification & Validation (V&V) shall be operated after each step and may require going back to previous steps.

1.1.4 Standards

The foundations of systems engineering are defined in standards. The IEEE 1220 *IEEE Standard for Application and Management of the Systems Engineering Process* "defines the inter-disciplinary tasks that are required throughout a system's life cycle to transform stakeholder needs, requirements, and constraints into a system solution" [34]. It is a product-oriented standard that applies to systems such as transportation and information systems. This standard describes eight processes linked together in an iterative process, with requirements analysis and validation, functional analysis and verification, synthesis and design verification being operated in parallel with trade studies and assessments of requirements, functional and design, as illustrated in Fig. 1.2.

The ISO/IEC/IEEE 15288 *Systems Engineering—System Life Cycle Processes* [1] is focused at processes and lifecycle. It replaced the previous MIL-

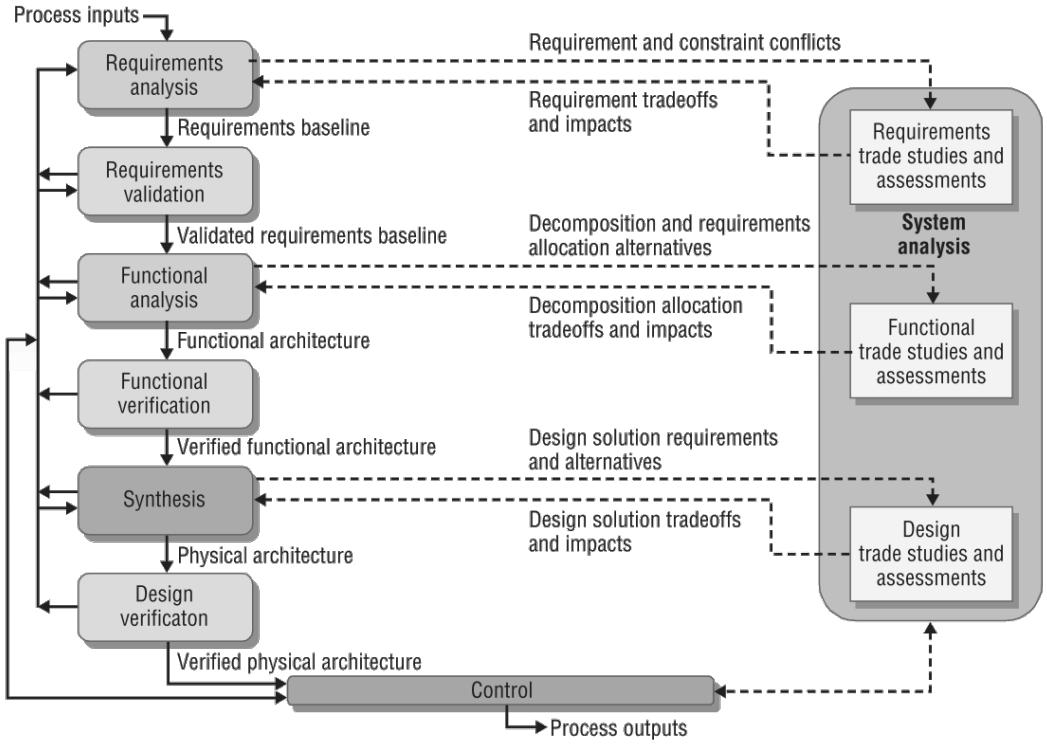


Figure 1.2: IEEE 1220 systems engineering process. SEP decomposes into eight subprocesses with their associated tasks, general process flows, and activities. [34]

STD 499A standard that was canceled on February 27th, 1995; this standard was an essential influence in the IEEE 15288 and IEEE 1220 [105]. The life cycle processes are organized in four sections:

- Agreement processes
- Organizational Project-Enabling Processes
- Technical Management Processes
- Technical Processes

These processes are illustrated in Fig 1.3.

Together, IEEE 15288 and IEEE 1220 provide the practices for the sound design of a system solution.

EIA 632 *Processes for engineering a system* [51] describes the process and activities that shall be operated to engineer a system [83]. The standard describes thirteen processes to design a system that are reported in five sections:

- Technical Management
- Acquisition & Supply

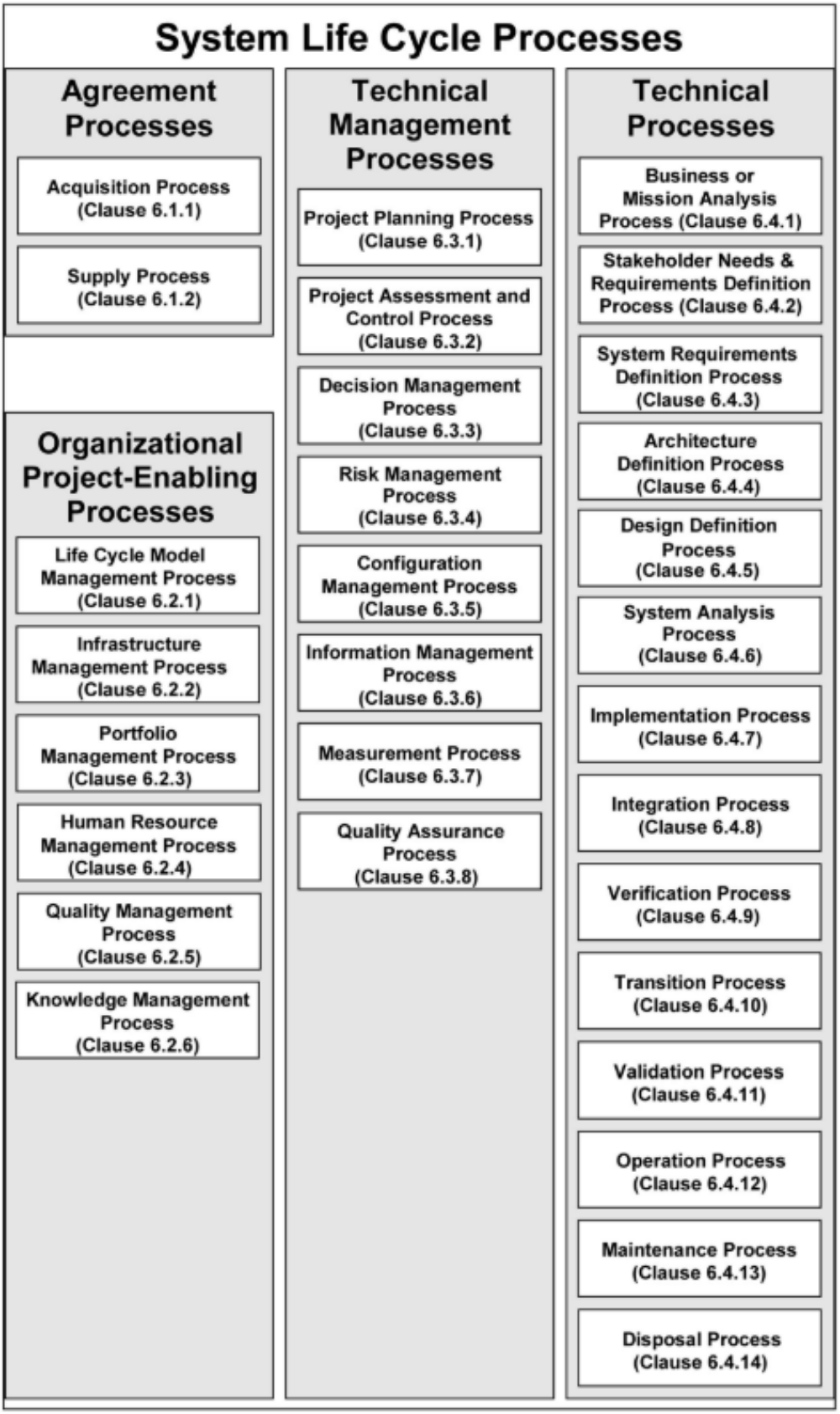


Figure 1.3: IEEE 15288 system life cycle processes [70]

- System Design
- Product Realization
- Technical Evaluation

These processes are described in Fig. 1.4.

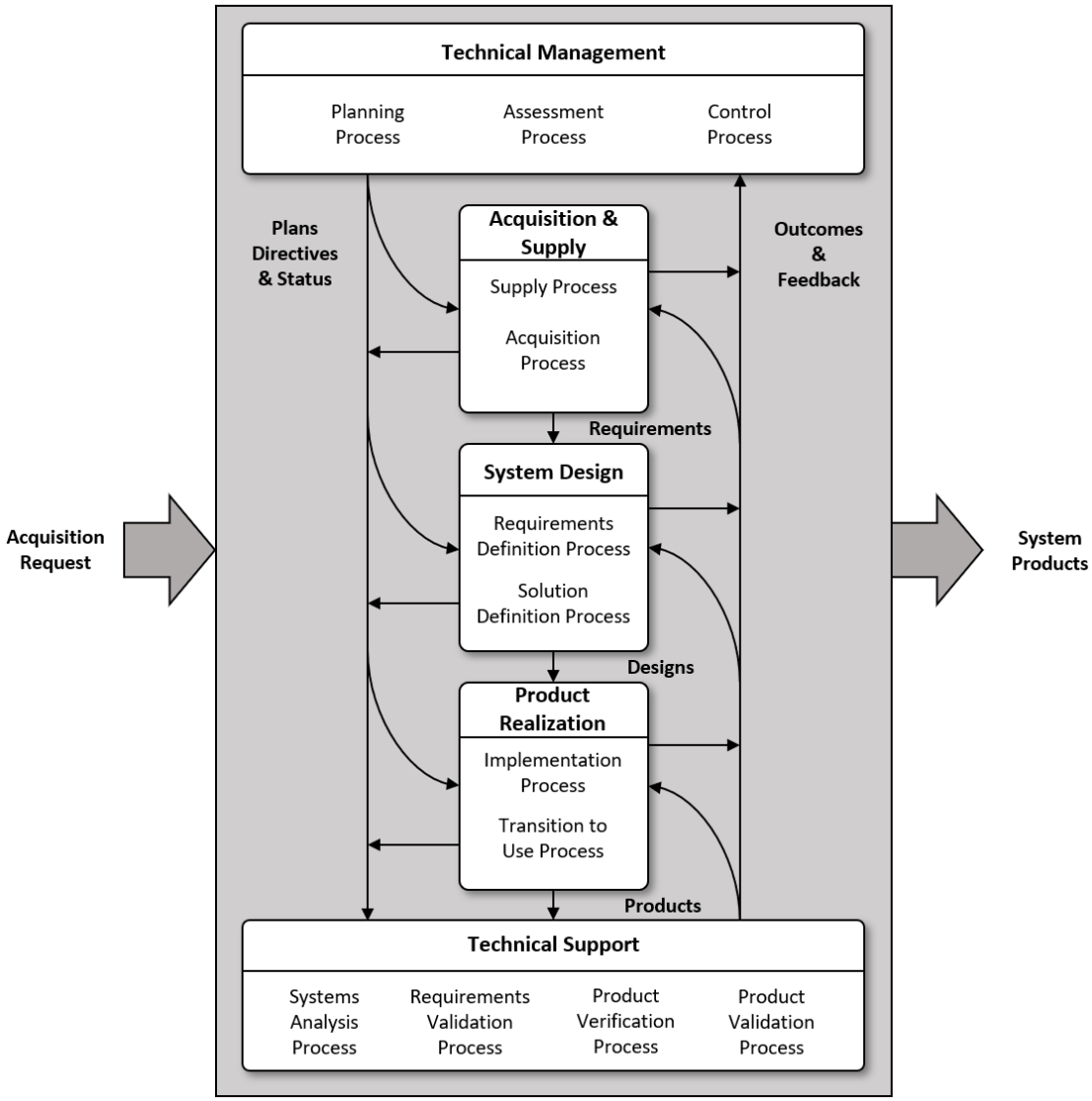


Figure 1.4: EIA 632 Process [83]

1.1.5 Models

Models are made for a purpose; we will further discuss what this implies in 1.3, but there is a point to be made here about why we create models in System

Engineering.

We choose which model to use depending on why and what we want to represent. Depending on the motivation for their creation, we can identify three kinds of models [79]:

- Models built for communication purposes: These representations are meant to be used as means to give information over a system to other people. They can be used for intents such as traceability with tools like SysML [91].
- Models buit for computation purposes: Computation models, such as Modelica, are used to represent a particular system behavior and compute indicators.
- Models built for generation purposes: We can also use models that can generate a solution to a problem. An example of a generation model is code generation.

1.1.6 System Architecture

With the rise of programming and micro-informatics, system architectures became a critical topic in the nineties [74]. Although in the first time the discipline was first developped for software, it was then extended to complex systems.

One proposition for a definition of system architecture is given in [31]:

> "System architecture is an abstract description of the entities of a system and the relationships between those entities."

System architecture is defined around three principal points of view, described by AFIS [30] and INCOSE [5]:

- Operationnal point of view

  Represents the capture of stakeholder needs for the system in its operational context during its lifecycle. It identifies the stakeholders and external systems, use cases and operationnal scenarios.

  The Black box view defines the boundary and environment of the system as well as its goal. In this view the system is defined through its inputs and outputs. Its internals are not specified, as the designers are not yet interested in what the system does or how it is made.

- Functionnal point of view

  Shows the functional requirements through functional modes and functional structures.

  White box view that defines the system and its operation at a high level of abstraction. In this view the system functions are defined, the designers describe what the system does.

- Physical point of view

  Shows the way physical requirements are impemented through material configurations, organs, and components allocated to functions.

  White box view defines subsystems and components as well as their structure in a concrete way, allowing to implement the functions. In this view, the components of the system are defined, the designers describe how the system is made.

1.1.7 Architecture frameworks

The points of view in System Architecture are characterized by architecture frameworks [5]. Architecture frameworks are defined in ISO/IEC/IEEE 42010 [69] as:

> "Conventions, principles, and practices for the description of architectures established within a specific domain of application and/or community of stakeholders."

Some of the most well-known frameworks are the DoDAF, DAF, NAF, CESAM grid, Arcadia, but other researchers have also proposed such methodologies. These frameworks describe how systems should be designed

Department of Defense Architecture Framework (DoDAF)
The United States Department of Defense developed the DoDAF to provide an architecture framework for the design of architectures in US Department of Defense projects. This includes both weapons development and other kinds of military technologies.

The DoDAF revolves around viewpoints [33] illustrated in Fig. 1.5:

- All Viewpoint: Describes the important aspects of the architecture across the different viewpoints.
- Data and Information Viewpoint: Describes the data flows in the system's architecture.
- Standards Viewpoint: Describes the system's normative environment, which applies to requirements, systems engineering and services.
- Capability Viewpoint: Describes the system's requirements, capabilities, and timings.
- Operational Viewpoint: Describes the operational scenarios, activities, and requirements for the system's capabilities.

- Services Viewpoint: Describes services and their interconnections for providing and supporting the system's functions.
- Systems Viewpoint: Describes the subsystems that provide and support the functions. Describes their interfaces, ressources, and functionalities.
- Project Viewpoint: Describes the relations between the requirements and the project implementation

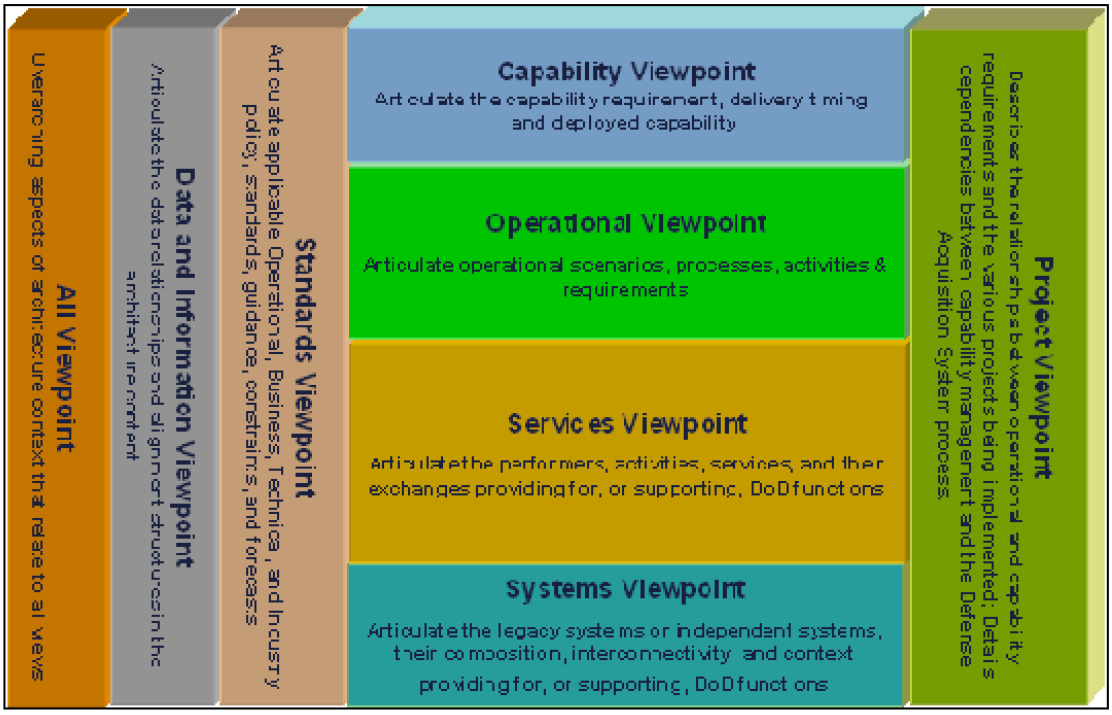


Figure 1.5: DoDAF viewpoints [33]

CESAMES Systems Architecting Method

The Center of Excellence on Systems Architecture, Management, Economy, and Strategy (CESAMES) proposed the CESAMES Systems Architecting Method (CESAM). This methodology is based on the operational, functional, and constructional visions, as illustrated in Fig. 1.6.

Each of these views is based on requirements, expectations over the system, and descriptions that answer the requirements.

- Constructional vision: The modeling begins with understanding the stakeholder's needs, the environment, missions, and scenarios the system will face.

- Functional vision: Functional requirements of the system are declared ("The <component of the system> shall <do something>"). Functional diagrams are drawn to describe the functions that the system should have to comply with the functional requirements.
- Operational vision: Constructional requirements are declared ("The <component of the system> shall <be something>"). Technical diagrams are drawn to describe the component that should constitute the system, and their interactions, to comply with the constructional requirements and operate the system's functions.

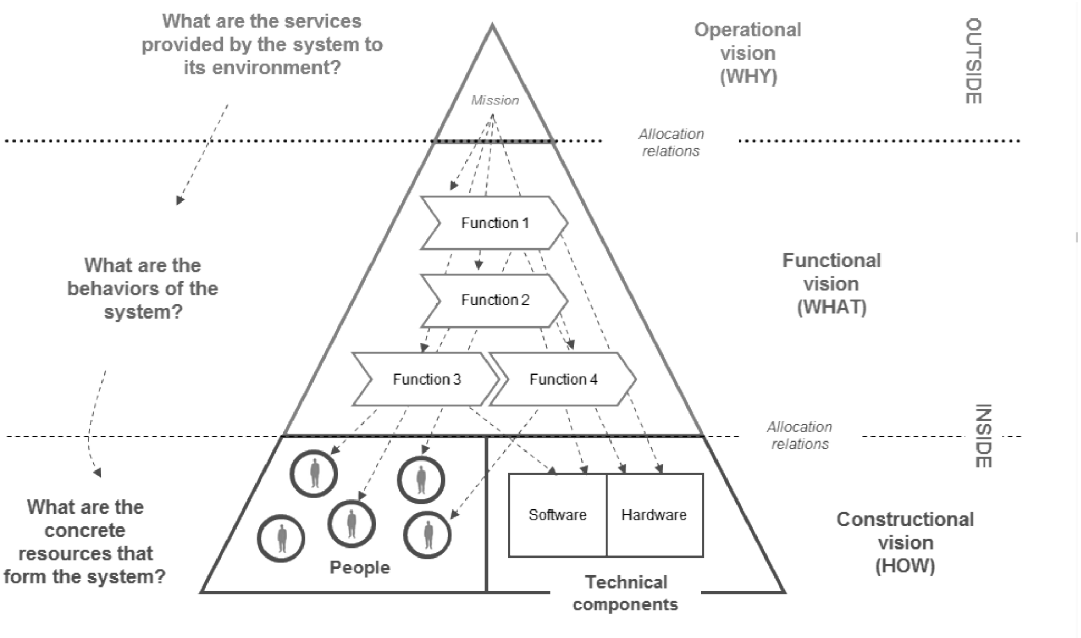


Figure 1.6: Pyramid of the CESAM architecture framework [73]

Arcadia Method

The Arcadia methodology is an architecture framework specially designed for the Capella tool. This methodology revolutes around the three viewpoints, each associated to specific diagram [102]:

- Operational needs: The Operational Architecture Blank diagram represents the allocation of Operational Activities to Operational Entities.
- System Analysis: Dataflow diagrams represent the system to show functional chains and the information dependancy between them.
- Logical architecture: Fonctions are represented in architecture diagrams and their relations. Scenario diagrams can also be used to represent functional scenarios.

- Physical architecture: Subsystems and Components are represented in architecture diagrams, with tree diagrams representing system breakdowns.

These viewpoints are modeled in an iterative process and coupled through performance, safety, and cost viewpoints, with bridges towards these disciplines.

An overview of the methodology is given in Fig. 1.7.

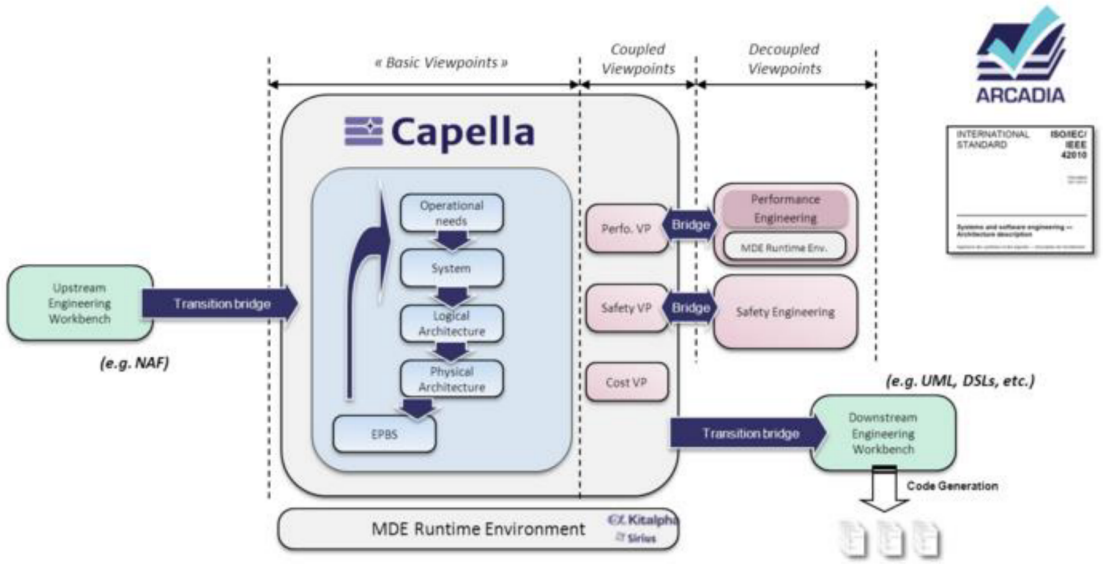


Figure 1.7: Capella big picture [102]

Some other examples

We can find other examples of architecture frameworks proposed by researchers and software editors that do not call themselves Architecture frameworks but rather Model-Based System Engineering methodologies. A more complete survey of these frameworks can be found in [42].

Rational Harmony for System Engineering is a SysML-based methodology that encompasses the V-Cycle [60]. It revolves around three main processes:

- Requirements analysis: An analysis of the process input. The stakeholder requirements are analyzed and refined, and a "Stakeholder Requirements Specification" is created.

- System functional analysis: The functional system requirements are transformed into a description of system functions. This analysis is based on use cases.

- Design synthesis: A physical architecture that can perform the functions is developed. Design synthesis is a top-down approach that begins with

architectural analysis, then architectural design (with iterations between these two first steps at each level of decomposition), and finally, detailed architectural design.

The Object-Oriented Systems Engineering Method (OOSEM) is a top-down methodology that uses SysML [42]. This methodology, presented in section 9.4 of the INCOSE System Engineering Handbook [61] includes six activities: Analyze stakeholder needs, Define system requirements, Define logical architecture, Synthesize candidate allocated architectures, Optimize and evaluate alternatives and Validate and verify the system. It aims to support the analysis, specification, design, and verification of systems [49]

Mhenni et al. [88] describe a methodology for the design of mechatronic systems using SysML diagrams. First, black-box analysis is performed to identify a set of requirements. The global mission of the system is defined, along with its lifecycle, context, external interfaces, the services it provides, and its functional scenarios. This leads to requirements specification and traceability. Then a White-box analysis is performed to design the system's functional, logical and physical architecture.

The magic grid framework [7] was proposed by the editor NoMagic as guidance to using its tool MagicDraw. The system is modeled through Requirements, Behavior, Structure, and Parameters at the Problem, Solution, and Implementation levels. This is done through the black box (the system's internals are not specified) and white box (the system's internals are specified) perspectives and with operational, functional, and physical points of view.

Knowing these frameworks allows us to understand how complex systems are designed. Although we have described a few of them, we do not compare them, as the work presented in this thesis does not rely on one of them but seeks to be agnostic of the architecture framework.

1.1.8 Model-Based System Engineering

The usual approaches for Systems engineering have been using documents to describe the system's requirements, environment, architecture, and other views. Although those documents are sometimes created in tools such as Excel or Doors, they are textual and rely on natural language.

The model-based approach in Systems engineering originates in model-based software development [42]. Software developers use such approaches to represent software architecture, with tools such as the UML notation [90]. This language was derived into the SysML notation [91], which system engineers use to describe complex systems. Another MBSE notation is CAPELLA, which is associated with the Arcadia architecture framework.

The model-based Systems engineering tools such as SysML or CAPELLA use diagrams to represent different system views. Structural and behavioral as-

pects are represented, with diagrams such as requirement diagrams allowing for representation of the requirements, activities diagrams allowing for representation of the functions, and block definition and internal block diagrams allowing for representation of the architecture. The MBSE languages present a lack of semantics [115], which causes ambiguities without using a third-party methodology. Therefore architecture frameworks (see section 1.1.7), provide guidelines to create systems.

Although we often use MBSE for such diagrammatic models, Systems engineering encompasses various disciplines. Therefore, some domain-specific languages (DSL), such as languages for multi-physics or safety assessment, can also be interpreted as part of MBSE.

1.1.9 Harel Statecharts

The MBSE models can include a behavioral description of the system. In the case of SysML, part of this behavior is represented through Harel Statecharts.

As specified in [91], the SysML concept of State Machine is derived from the state machines found in the UML language [90]. These State Machines are mostly based on Harel's Statecharts, a broad extension of conventional state machines and state diagrams. A complete description of the Statechart formalism can be found in [56], with a description of a semantic for the formalism in [57].

Statecharts are a visual formalism that extends the usual state machines and state diagrams. In the same way as traditional state machines, a Statechart is composed of states that are connected through transitions, each transition having a source and a target state.

In addition, the Statechart formalism provides a way of expressing depth in the model with composite states. A composite state is a state that can encapsulate other states, as pictured in figure 1.8. The composite state can be entered by entering one of the encapsulated states. It can also be entered by being the target of a transition, in which case an initial state has to be declared and will be entered. The composite state can be the source of a transition, thus leaving any internal state that the Statechart was in if the transition is fired.

Finally, Statecharts have orthogonal states that are composite states with multiple "orthogonal components" such as, when entering the orthogonal state, the Statechart enters one state in each orthogonal component. An example of orthogonality is found in figure 1.9. In this example, an orthogonal state `Y` encapsulates states `A` and `D`, when `Y` is entered, state `A` and `B` are entered and therefore states `B` and `F` are entered as they are the initials states of respectively `A` and `D` — this is shown by them being the target of the transition with the black point as its source.

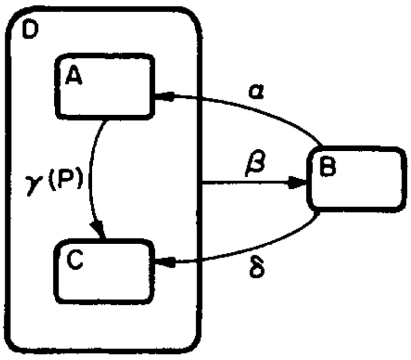


Figure 1.8: State chart with composite state D encapsulating states A and C [56]

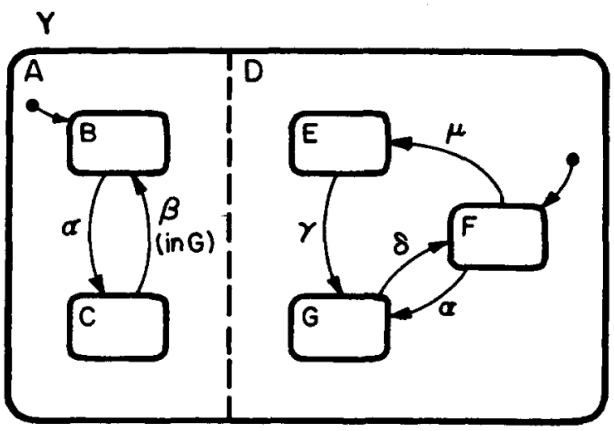


Figure 1.9: State chart with orthogonal state Y encapsulating states A and D[56]

1.2 Dependability

The subject of this thesis is consistency between Model-Based Systems Engineering (MBSE) and Model-Based Safety Assessment MBSA models. This second section of the state of the art introduces dependability, which MBSA is a part of. Dependability is an engineering field that aims at making systems reliable and safe.

1.2.1 History

Dependability emerges from the necessity to assess the reliability and safety of a product. This need is not new, but the techniques used to carry on analyses have widely evolved in the last century.

Until the renaissance, it was believed that a chain's weakest link was its main defining element. Dependability would therefore be simply conduced by rein-

forcing that element. Many events during the late nineteenth and early twentieth centuries led to the emergence of safety analyses.

The InterSection magazine of November 2004 [40] lists some of these events; we sum them up, along with a few other events, in table 1.1.

1842	Versailles rail accident
1903	Parisian metropolitan fire
1912	Sinking of the Titanic
until 1930's	Intuitive approach: reinforce the weakest link
1930's	Beginning of air transports
	Statistic approach with failure rates
1940's	Murphy's law : "Anything that can go wrong will go wrong."
1950	Advisory Group on the Reliability of Electronic Equipment (AGREE)
	Mean Time Between Failures(MTBF)
1960's	Failure modes and Effects Analysis (FMEA)
	Fault trees
1967	Appolo 1 disaster
1970's	Risk assessment
1986	Tchernobyl accident
1990's	Model-Based Safety Assessment (MBSA)

TABLE 1.1 Chronology of events during the emergence of Dependability

1.2.2 Definition

Alain Villemeur [114] defines dependability by

> "la sûreté de fonctionnement consiste à évaluer les risques potentiels, prévoir l'occurrence des défaillances et tenter de minimiser les conséquences des situations catastrophiques lorsqu'elles se présentent."
>
> "Dependability consists in evaluating potential risks, predicting failure occurrences and trying to minimize consequences of catastrophic situations when they happen."

Another definition, given by IEC 60300-3-4 [67] is:

> "Dependability is the ability to perform as and when required. A dependable item is one with justified confidence that it operates as desired and satisfies agreed stakeholder expectations."

While agreeing with the latter definition, Marvin Rausand and Arnljot Høyland [97] consider dependability as a characteristic of the system that can be measured through:

- Mean Time To Failure (MTTF)
- Number of failures per time unit (failure rate)
- The probability that the item does not fail in a time interval (0, t] (survival probability)
- The probability that the item is able to function at time t (availability at time t)

A dependable system will not fail more often than it is acceptable to fail. To achieve the dependability of the system, analyses are operated to compute chosen dependability parameters and estimate how often it will fail. Redesigns can be made to improve the system. In the following parts of this section, we detail how this process is operated.

1.2.3 Concepts

Avizienis and Laprie [76] proposed a dependability taxonomy, organised with the three following concepts :

- Attributes are system parameters that can be assessed to evaluate their dependability. They include availability, reliability, safety, integrity, maintainability, and confidentiality.
- Means are methods and technics meant to optimize attributes; they aim at making the system more dependable.
- Threats are events that affect the system's dependability. They include faults, errors, and failures.

These concepts and their implications are summed up in Fig. 1.10.

As these concepts are mainly safety oriented, it is essential to note the distinction between safety and security. Safety is the system's resilience to accidental harm, whereas security is its resilience to malicious harm [85]. This distinction is essential, as this results in two engineering fields, which, although

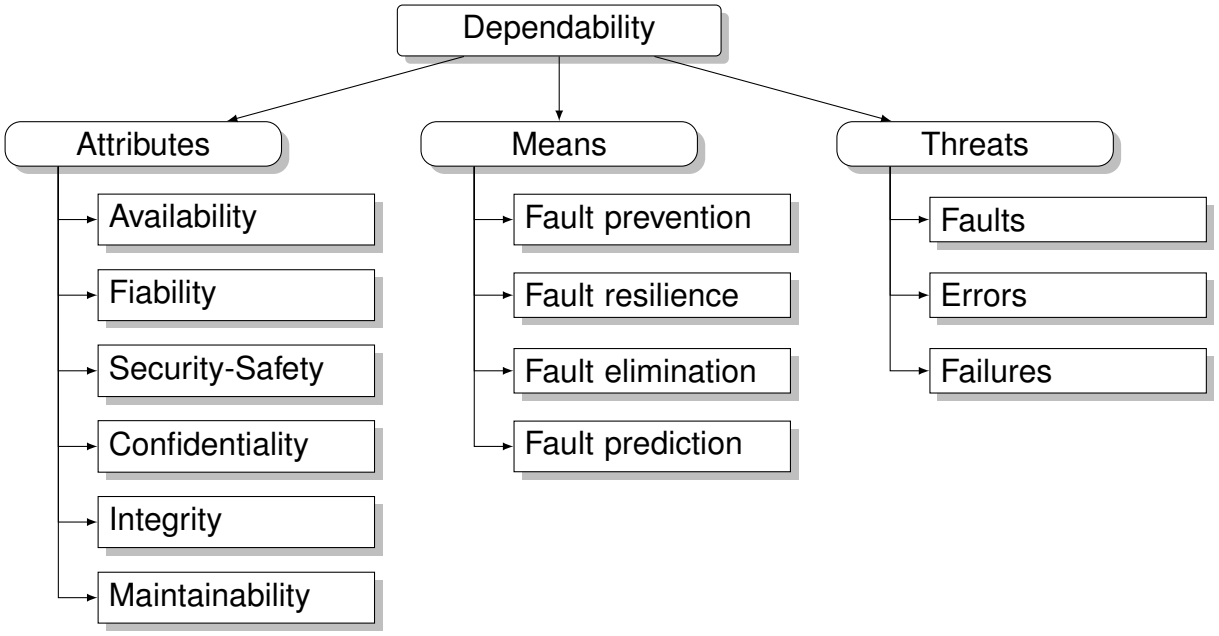


Figure 1.10: Taxonomy of dependability [76]

they share many resemblances, are separate and usually operated through different tools and teams [46]. We have used the term Threat in the taxonomy, as it is widely used in the literature, but this word can be misleading as it implies an attacker. A better word would be Hazard, which is the safety equivalent of a threat, i.e., a dangerous situation [11]. In the same way, in the safety vocabulary, we talk about failures for events where the system functions are not fully functional. In contrast, the corresponding security concept would be an incident.

In this thesis, we are interested in the systems' safety and will not consider security concepts.

1.2.4 Standards

The ISO/IEC 61508 [71] describes the general methodology used for the safety assessment process of electrical/electronic/programmable electronic systems (E/E/PES). This standard considers that since zero risk cannot be reached, the probability of failure should be known and lowered as much as possible.

To achieve functional safety, the IEC 61508 uses three safety lifecycles [19]:

- Overall Safety Lifecycle
- E/E/PES Safety Lifecycle
- Software Safety Lifecycle

The E/E/PS Safety Lifecycle and Software Safety Lifecycle are parts of the Overall Safety Lifecycle. They include a definition of safety requirements specific to E/E/PES and Software, their validation, and the design and integration of the E/E/PES and Software parts of the system.

The Overall Safety Lifecycle is illustrated in Fig. 1.11. The idea behind this lifecycle is to make the approach so that a check is made after each phase to confirm that the required outputs have been produced [26]. This is achieved through methodical planning of the functional safety activities, with inputs and outputs defined for each phase.

The approach preconized by the IEC 61508 is to develop the system and its functions and then operate a Preliminary Risk Analysis (PRA) over the system. During this PRA, Safety Integrity Levels (SIL) are attributed to each possible failure of the system.

A SIL is a criticality level for a failure of the system. It is attributed depending on the level of danger associated with the failure and corresponds to a target failure measure. Target failure measures are the measured probability that the failures will happen per hour.

These target failure measures are defined depending on the demand mode. Low demand mode is when the frequency of demands for the function is less than once per year and less than twice the proof-test frequency (meaning that the system is checked between each function use). High demand or continuous mode is when the frequency of demands is more than once a year or twice the proof-test frequency.

Description of the SIL can be found in table 1.2. IEC 61508 does not explicitly state the safety integrity levels requirements for safety functions, as they widely depend on the applicative context.

The different fields of engineering use either the IEC 61508 standard or specific standards that are derived from IEC 61508; these standards are detailed in table 1.4.

Level	**Target failure measure in low demand mode**	**Target failure measure in continuous or high demand mode**
4	$\geq 10^{-5}$ to $< 10^{-4}$	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-4}$ to $< 10^{-3}$	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-3}$ to $< 10^{-2}$	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-2}$ to $< 10^{-1}$	$\geq 10^{-6}$ to $< 10^{-5}$

Table 1.2: Levels of SIL in IEC 61508

In the aeronautic context, DO-178B [104], summarized in [110], declares five levels of criticality called Design Assurance Level (DAL). They are equivalents of the IEC 61508's SIL. These five levels are detailed in table 1.3.

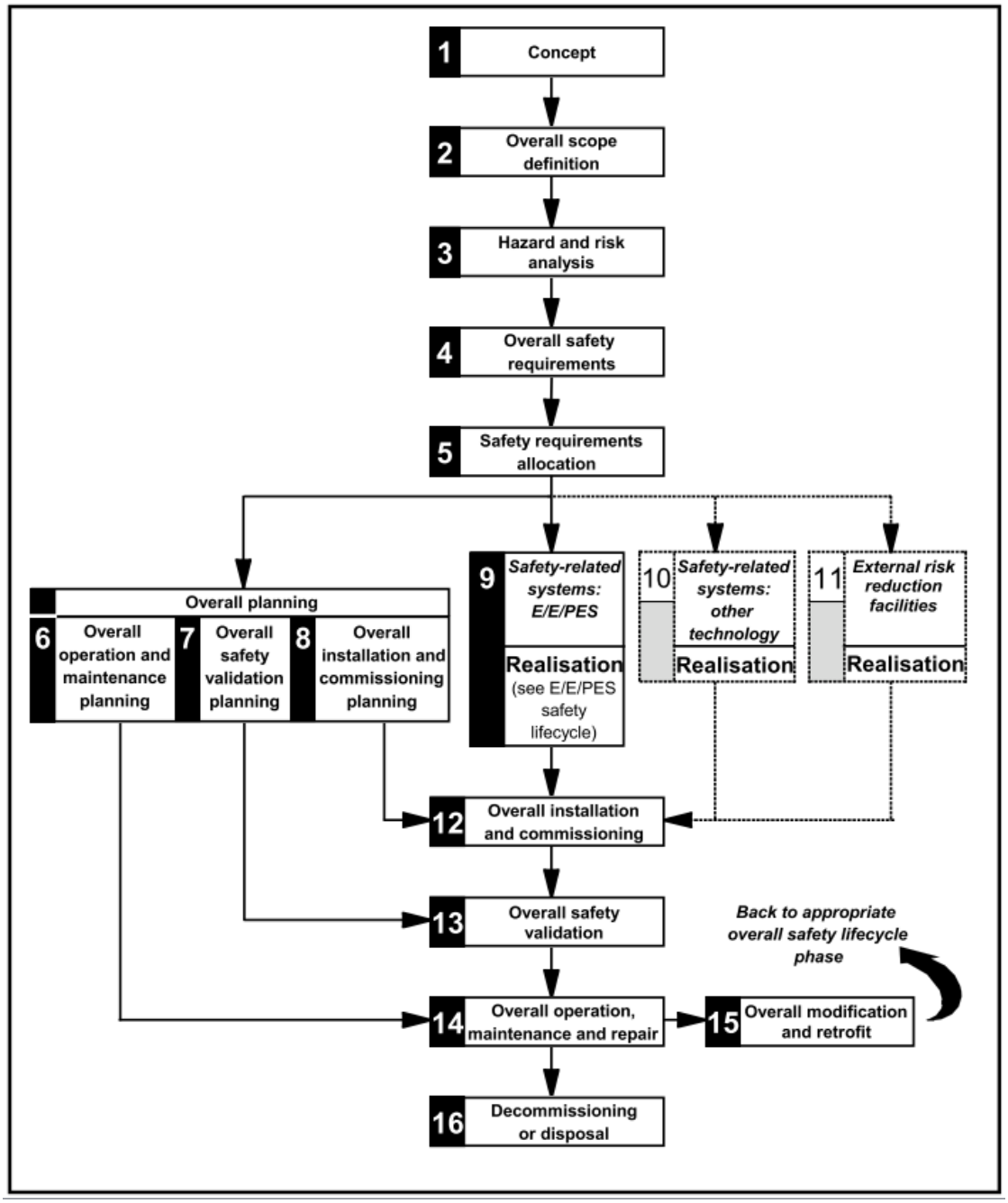


Figure 1.11: Overall safety lifecycle of the IEC 61508 series [71]

Level	Name	Failure Rate
A	Catastrophic	$10^{-9}/h$
B	Hazardous	$10^{-7}/h$
C	Major	$10^{-5}/h$
D	Minor	$10^{-3}/h$
E	No Effect	n/a

Table 1.3: Levels of DAL in DO-178B

A DAL level is attributed to a failure mode through its effects. ARP 4761 [62] defines these effects as such:

- Catastrophic: All failure condition that prevent continued safe flight and landing
- Hazardous: Large reduction in safety margins or functional capabilities. Higher workload or physical distress such that the crew could not be relied upon to perform tasks ccurately or completely. Adverse effects upon occupants.
- Major: Significant reduction in safety margins or functional capabilities. Significant increase in crew workload or in conditions impairing crew efficiency. Some discomfort to occupants.
- Minor: Slight reduction in safety margins. Slight increase in crew workload. Some inconvenience to occupants.
- No Effect: The failure does not affect the flight safety

[21] does a survey of the categories used for criticality in diverse industries. We detail the equivalents of SIL and DAL in table 1.4.

Industry	Criticality Levels	Standard
Automotive	Automotive Safety Integrity Level (ASIL)	IEC 61508 [71], ISO 26262 [68]
Railway	Safety Integrity Level (SIL)	EN 50129 [6]
Nuclear	Plant Condition Categories (PCCs)	IEC 61226 [65], IEC 61838[66]
Aeronautics	Design Assurance Level (DAL)	DO-178B [104]
Space	3 categories	ECSS Q30 [43], ECSS Q40 [44]

Table 1.4: Criticality levels in diverse industries

1.2.5 Tools and methodologies for safety assessment

1.2.5.1 Tools for safety assessment

Failure Modes and Effects Analysis (FMEA):
The failure modes and effects analysis is one of the first systematic methodologies for safety assessment. It was first introduced by the US military in MIL-P-1629 [113], later revised in MIL-P-1629A [8].

The FMEA analyzes each system's function with the attribution of every failure mode. These failure modes include:

- Loss of the function
- No answer to solicitation
- Partial loss of the function
- Intempestive triggering of the function

The effects of each failure are declared in the FMEA table, and criticality is attributed to them depending on the danger they represent. This criticality is usually part of one of the norms cited in 1.2.4, such as SIL, ASIL, PCCs or DAL levels.

Fiability diagrams:
A fiability diagram is an alternative vision to the fault tree. It focuses on the system and its states rather than events. A fiability diagram is composed of [92]:

- An entry E, a diagram body, and an exit S;
- A flux transmitted from E to S, going through the diverse paths;
- Failures of some entities block the flux at the failed component.

If there is no path from E to S, the system is failed; otherwise, it is functional.

Fault trees:
Fault tree analysis (FTA) uses logic trees that allow to compute causality in the occurence of diverse events and a failure condition. The Bell Telephone Laboratories first introduced it in the development of the Minuteman missile in the 1960's ([116] cited by [118]).

Fault trees are composed of events, leafs of the tree, and logical gates that lead to the tree's root. The tree's root represents a failure condition of the system that the fault tree focuses on. Fault trees allow us to split the failure conditions

into detailed events [109]. Events of the tree represent basic events of the system, such as component failures. An example of a generic fault tree is shown in Fig. 1.12. This fault tree is equivalent to the logical proposition:

$$FailureCondition \equiv (Event1 \vee Event2) \wedge (Event3 \wedge Event4)$$

Failure rates are attributed to the events; in this case, an exponential law of parameter $\lambda = 10^{-6}$, i.e., for $t > 0$, the probability of a failure at time t is:

$$f(x) = \lambda e^{-\lambda t} = 10^{-6} e^{-10^{-6} t}$$

Through the logical gates, the fault tree allows to compute which combinations of events lead to the failure condition and with which rate; these combinations are called cuts.

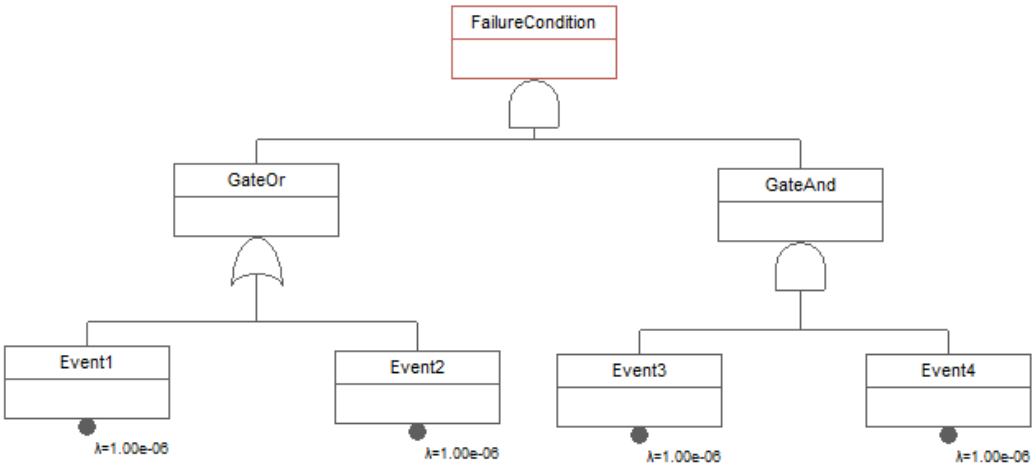


Figure 1.12: Exemple of a small fault tree for a battery

A cut that no longer leads to the failure condition being true if any event is removed is called a minimal cut. For the the fault tree from figure 1.12, the minimal cuts are listed in table 1.5. We call order of a cut the number of events in this cut.

Cut order	**Event set**	**minimal cut**
3	{Event1, Event3, Event4}	yes
3	{Event2, Event3, Event4}	yes
4	{Event1, Event2, Event3, Event4}	no

Table 1.5: Minimal cut sets for the fault tree from Fig. 1.12

Markov Chains:

Markov chains represent the diverse possible states of the system and their relations. They implement the notion of reconfiguration, i.e., the ability to repair the system after a failure. They also allow the evaluation of fiability, through probabilistic computations.

Andrei Markov introduced Markov chains in 1906 as an extension of the Bernoulli model [50]. They are sequences of random variables within which the probability of moving to the next step only depends on the present state and not on the previous ones.

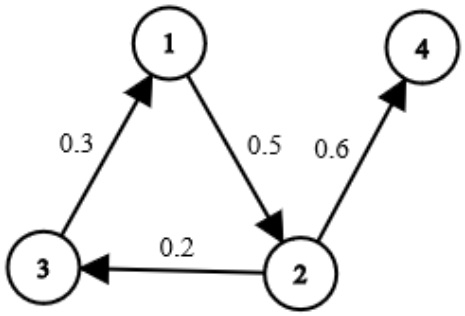


Figure 1.13: example of a graph representing a Markov chain

We can describe a Markov chain through an oriented graph with weighted edges. The graph in Fig. 1.13 represents a Markov chain with four states; the weights of the edges are the probability of moving onto the next state. We can also see this Markov chain as the matrix from Fig. 1.14.

$$\begin{pmatrix} 0.5 & 0.5 & 0 & 0 \\ 0 & 0.2 & 0.2 & 0.6 \\ 0.3 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1.14: Simulation matrix for the Markov chain from Fig. 1.13.

Petri nets:

Petri nets were first introduced in Carl Adam Petri's Ph.D. thesis [94] in 1962. A Petri net is an oriented graph with two types of vertices, places and transitions. The edges of the petri net can run from a place to a transition or from

a transition to a place. A place can contain tokens that are consumed when a transition is fired from that place and created when a transition is fired to that place.

Mathematically, a Petri net is a tuple $N = (P, T, F, W, m_0)$ where:

- P and T are disjoint sets of places and transitions
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, i.e., the set of arcs of the oriented graph.
- $W : ((P \times T) \cup (T \times P)) \rightarrow \mathbb{N}$ it the arc weight mapping
- $m_0 : P \rightarrow \mathbb{N}$ is the initial marking, i.e., the initial distribution of tokens.

A transition can only be fired if all incoming arcs have weights lower than the number of tokens in their source places. Fig. 1.15 shows a Petri net with Working and Failed places and Failure and Repair transitions. The working place contains a token, meaning that the failure transition with a weight of 1 can be fired.

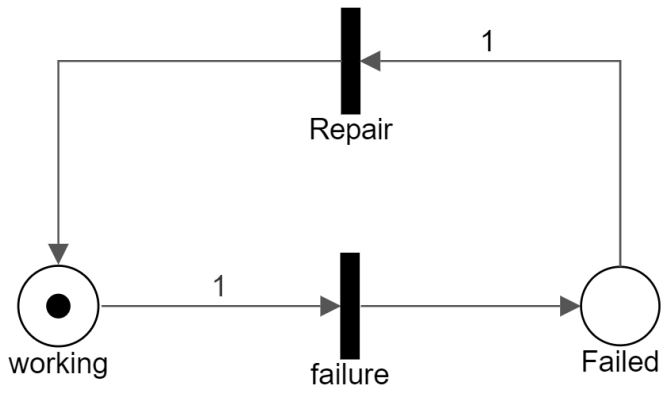


Figure 1.15: Petri Net for a Working/Failed component.

Since the 80's petri nets have been used as behavioral models for Monte-Carlo simulations [92]. Stochastic Petri nets are Petri nets within which delays are allocated to the transitions. They can be used to model the dysfunctional behavior of components.

Model-Based Safety Assessment:

Similarly to MBSE, the field of safety analysis is faced with the challenge of complexity. The traditional approaches, such as Petri nets, reliability block diagrams, event trees, and fault trees, are models that are very close to mathematical equations and lack structure or lack expressivity power. They also are

very abstract compared to the system specification. The model-based safety assessment (MBSA) approaches aims at answering these limitations.

There are two very different approaches to MBSA [80]:

- Defining safety assessment through the extension of the models used in system development.

  Examples of such approaches are [59] where the SysML model is used to produce a FMEA or [29] where the SysML model can be used to produce fault trees. Saqui-Sannes et al [106] also propose a methodology where they use a software called TTool along with SysML to enrich the MBSE model with safety and security properties and check whether they are compliant or not with the requirements. Lai et al. [75] make a literature review of these approaches. Such approaches will produce safety artifacts that are consistent with the architecture by construction. However, they require overloading the systems engineering model with artifacts with different modeling intents than the original System Engineering. Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [93] is a MBSA tool that is based on simulink.

- Performing safety assessment based on dedicated models.

In this work, we focus on dedicated MBSA models.

The MBSA models have a high expressivity, they represent systems with a point of view closer to their architecture. The gain on expressive power is obtained by going from Boolean formalisms to state or event formalisms. There are three different kinds of model-based risk and safety assessment formalisms [16], including specialized profiles of MBSE tools such as [87], approaches that extend fault trees or reliability block diagrams such as dynamic fault trees [35] and model-based safety and reliability assessment modeling languages. In this work, we are interested in MBSA modeling languages, such as SAML [53], Figaro [25], and especially AltaRica 3.0 [15].

AltaRica 3.0 is the third version of the AltaRica language; it is based on the mathematical framework of guarded transition systems (GTS) [98], which are state automata.

Guarded Transition Systems The AltaRica 3.0 language [15] is designed around the concept of Guarded Transition Systems (GTS). The GTS formalism generalizes Blocks diagrams and Petri nets, as described in [98].

A GTS can contain variables, more precisely state variables to represent the states of the system, and flow variables to represent elements exchanged between components of the system; events, transitions and assertions.

An event allows for the triggering of a transition, which will change the value of state variables. E.g. if a variable `s` is defined over a set of values `{OK, KO}` and initiated to `OK`, then we can define an event `failure` associated with a transition

changing the value of state from OK to KO. The domain of the variable s is declared using the keyword domain. We can also declare Real, Boolean, or other common types as values for state variables or parameters. For example we declare Booleans input and output with reset value true. This is illustrated in figure 1.16.

Finally, assertions are used to constrain the variables of the system. For example, in Fig. 1.16 we set up the output value to be equal to input if the system state is OK and false if the system state is KO.

```
domain systemstate {OK, KO}
block system
  systemstate s (init = OK);
  Boolean input, output (reset = true);
  event failure;
  transition
    failure : s == OK -> s := KO;
  assertion
    output := if s == OK then input else
                                false;
end
```

Figure 1.16: example of the AltaRica3.0 representation of a GTS

It is possible to compose GTS. In that way, we can define multiple components in a system, each associated with one GTS. The composition of two GTS is defined as their free product as depicted in [98] and results in a GTS. AltaRica 3.0 models with multiple components are shown in section 2.2.3.

1.2.5.2 Safety Assessment methodology

In the aeronautics field, a system's dependability must be thouroughly examined to comply with certification authorities' guidelines. The product must be safe, as passengers are unlikely to survive a crash, and a plane can carry tens or even more than a hundred people. Thus, aircraft manufacturers carry out very thorough safety analyses of their systems.

The ARP4761 [62] provides recommended practices for the safety assessment of an aircraft. These practices are widely followed by aircraft and aircraft engine manufacturers and are currently the most exacting safety analyses carried out on engineered systems. Other engineering fields, such as the automotive and railway industries or the nuclear industry, also carry out safety studies using part of the tools we cite.

Functionnal Hazard Assessment (FHA):

FHA is the first step of safety assessment. Systems functions and risks related

to them are listed. Failure conditions are identified, and a severity is allocated to each, depending on its effects. A maximal failure rate requirement is then allocated to each failure condition according to its severity. This process results in an FMEA table.

Preliminary System Safety Assessment (PSSA)
PSSA is the second step of safety assessment. It is an iterative process within which engineers assess the system's conformity with the safety requirements of the FHA. Based on the FHA results, fault tree analysis is usually used to compute which sets of events can result in failure conditions.

System Safety Assessment (SSA):
SSA is the third and final step of the Safety assessment. It is a validation approach for the design, similar to PSSA. SSA differs from PSSA by being a bottom-up approach, as opposed to the top-down PSSA. By allocating occurrence rates to events in the FTA, the occurrence rates of the failure conditions can be computed. These failure rates are compared to the safety requirements to assess the system's safety.

Common Cause Analysis (CCA):
CCA is conducted in parallel with FHA, PSSA, and SSA. It aims to evaluate the architecture's sensitivity to events with common causes or modes. CCA is achieved through particular risk, zonal safety, and common cause risk analyses.

1.3 Models

Because we are interested in models, and especially models consistency, they are the topic of this third section of the State of The Art. In this section we begin with a definition of what a model is and an introduction to models, and we then focus on model consistency and comparison.

1.3.1 Definition

A model is an abstraction of reality. Models can be of different nature. For example, they can be [79]:

- An abstract representation such as a mathematical description such as equation systems
- A graphical description such as diagrams or schematics
- An informatic model such as a language or an algorithm

In this work, many of the models we consider are conceptual or information models. An information model represents concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse [78]. We also work with some simulation models that allow us to represent the system and compute some of its indicators.

We explained in 1.1.5 that we can divide models into three kinds depending on their motivation: Communicate, compute and generate. These motivations drive the model designers' choice of which representation they will use. Communication will often use graphical or physical models, whereas computation and generation require more formal models such as mathematical or informatic ones.

1.3.2 Syntax and semantics

In order to design models unambiguously, we need languages that allow us to define them.

Such languages require a syntax, which is elements used to express concepts and a way to combine them to create structure in the language.

It also requires a semantic in order to give it meaning.

Syntax and semantics are the two pillars of formal languages used in modeling. David Harel and Behrnard Rumpe give one definition of syntax in [58]:

> "Depending on the language type, syntactic elements can be words, sentences, statements, boxes, diagrams, terms, models, clauses, modules, and so on."

The syntax is the language's words and the form that the program or model will have.

Semantics is more challenging to define. They attribute meaning to the things that are written using syntax. Of course, they are linked to syntax and cannot exist without it. Robert Floyd proposes an approach for the definition of semantics to assign meaning to programms [47]:

> "A semantic definition of a programming language, in our approach, is founded on a syntactic definition. It must specify which of the phrases in a syntactically correct program represent commands, and what conditions must be imposed on an interpretation in the neighborhood of each command."

Together, syntax and semantics provide rules to write a correct model. However, it is important to note that a model can be both syntactically correct and semantically wrong. Examples of that idea can be found in natural language.

> "The teapot ate my cat."

This sentence is grammatically correct; it follows the construction rules for a phrase in the English language. However, it is semantically incorrect, as it makes no sense since a teapot does not eat things.

Through complete syntax and semantics, we obtain a language that allows unambiguously expressing a set of concepts. In a perfect world, any language, whether written or graphic, should have complete syntax and semantics; we can call such a language "formal". However, that is not always the case, and many languages are not formal. For example, natural languages usually have incomplete semantics and result in possible ambiguities. Similarly, some modeling languages do not have complete semantics. Most MBSE languages, such as SysML, have incomplete semantics, and it is impossible to specify these models [115] formally.

1.3.3 Structural models and behavioral models

The MBSE and MBSA models are two different kinds of models that represent the same systems with two different modeling intents. The modeling intent is the primary constraint on the representation of the system in these models.

In [17], the authors state that models are composed of behavior and structure. This means modeling languages are composed of a mathematical framework and structural construction. Depending on the modeling intent, engineers use different models that allow for the representation of diverse behaviors. Because of the need for these diverse points of view over the behavior, and because we do not always obtain a model through the composition of two heterogeneous models, the diversity of models is irreducible [100].

Therefore we cannot avoid the creation of multiple models representing the same system and, therefore, the same structure, albeit with different behaviors. This leads to the problem of asserting consistency between these models. In this work, we are interested in comparing the structure of the models, as we want to show that the architectures of the systems represented by the models are the same.

1.3.4 Model consistency

Different approaches exist to assert consistency between heterogeneous models. The model federation [54] approach uses a database to store all data from a system's models. Within that database, the user can assert consistency and traceability between the models.

Proof theory is used by [45] in their inconsistency management approach. This methodology defines models as first-order logics, identifying inconsistencies through a computer-aided proof tool.

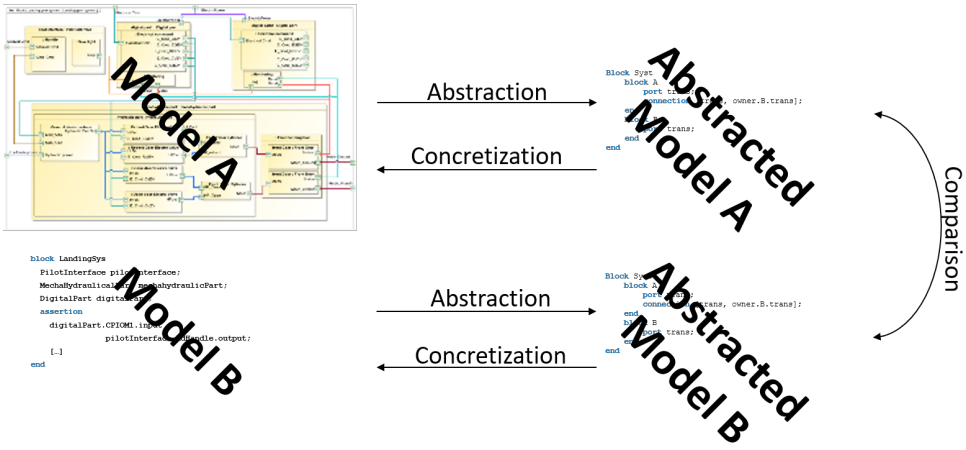


Figure 1.17: Model synchronisation approach.

Another approach to heterogeneous model consistency is model synchronisation [79]. The synchronization consists of a three-step process, which is illustrated in Figure 1.17:

- **Abstraction:**

  The models are translated into a common formalism.

- **Comparison:**

  The comparison is operated in the abstracted models. Writing the compared models in a standard formalism makes the comparison easier than with heterogeneous models.

- **Concretization:**

  The comparison results are carried back to the original models. Concretization can be achieved by correcting the models, annotations, or other means.

Since they are written in heterogeneous languages, it is impossible to compare the models as is. This is why they are translated to a common formalism that supports the artifacts required for comparison. In this common formalism, the models can be compared, i.e. we can find the correspondances and differences between them. Finally, as we want to have consistent models, the concretization step is meant to eliminate the inconsistencies, based on the comparison results.

The model synchronization approach is the base for different methodologies. In this work, we intend to specify them in a mathematical framework.

The SmartSync synchronization framework [13], is a synchronization methodology. Its authors provide a methodology and tools for synchronizing

the MBSE and MBSA models [12]. The comparison is operated on abstracted models written with the S2ML language, described in subsection 1.3.6.

SmartSync relies on the user to manually align model elements, as described in Section 5.2.1.

The directed graph methodology from [20] is also a synchronization methodology. It does rely on the abstraction of the models towards graphs. In the graphs, vertices represent parts and ports from SysML and blocks and classes from AltaRica; edges represent connectors from SysML and assertions from AltaRica. This framework allows for using graph search algorithms to traverse the graphs, and the use of concepts such as graph and sub-graph isomorphisms to compare the models.

The consistency links methodology [32] is another synchronization approach that creates links between MBSE and MBSA models. This methodology mainly focuses on the functional aspect of the system.

The three latest methodologies provide a structural comparison between Systems engineering and safety models. They rely on comparison models that follow the block, ports, and connection structure described in Section 1.3.3, with different ways to represent them.

1.3.5 SmartSync Methodology

The SmartSync approach is the one we use in this thesis's case studys. It is operated through the following steps [95]:

1. Transformation of models into S2ML (today is done manually, can be automated)
2. Basic comparison of S2ML Models (automatic), the tool provide a list of the model elements at the first system level
3. Analysis of the generated report and creation of a matching file from the report (manual)
4. Comparison of S2ML models with a matching file (automatic)
5. Analysis of the generated report and population of the matching file with new correspondences (manual)
6. Comparison with a populated matching file (automatic) and so on...

These steps are illustrated in Fig. 1.18.

If an element has no equivalent in the other model, the user can give it the attribute `forget` if he can justify why it is not an inconsistency.

The SmartSync tool takes two S2ML compiled models (in the XML format) as input and outputs a CSV comparison file. The user aligns the elements from

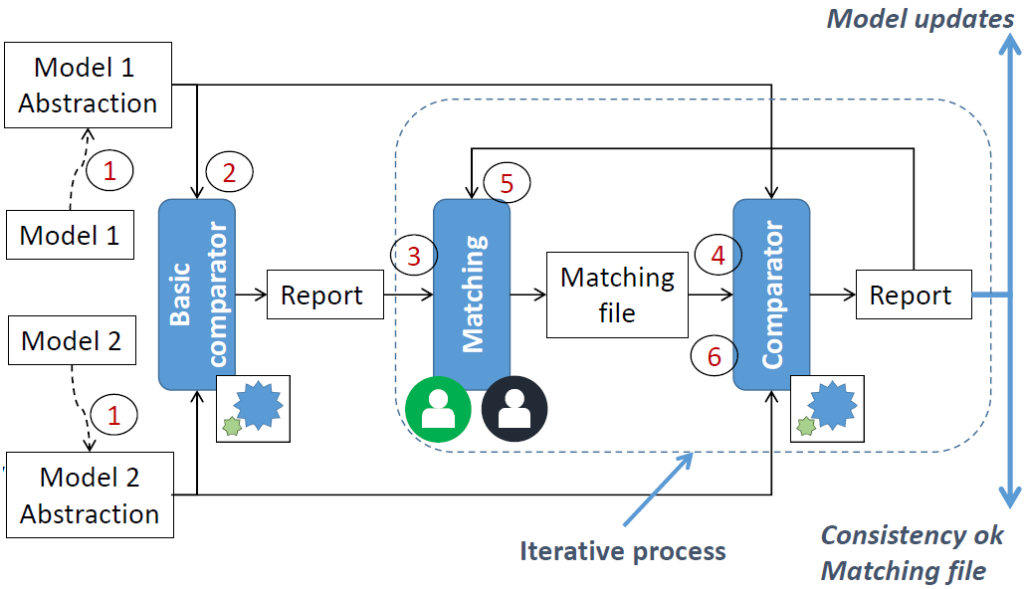


Figure 1.18: Model synchronization process using SmartSync [95]

both models in the CSV file, then re-iterates the SmartSync comparison with the two S2ML compiled models and the CSV file as an additional input. The SmartSync tool can then identify the children's elements and provides a new CSV comparison file to the user for the next iteration.

1.3.6 S2ML - System Structure Modeling Language

As its name subtly suggests, the System Structure Modeling Langage (S2ML) [14] is a modeling language dedicated to the representation of systems' structures. This language is the foundation of the S2ML+X paradigm [100]. The S2ML+X paradigm relies on the thesis (explained in 1.3.3) that models are composed of structure and behavior. S2ML is the structural basis for a set of modeling languages that implement different behaviors over a common structural paradigm. These languages include AltaRica 3.0 and SCOLA, which we use in this thesis. Although those languages are not derived from it, we can also see S2ML as a paradigm for other structuring languages, such as Lustre or Modelica. In this thesis, we demonstrate in Chapter 5 that we can consider synchronizing Modelica models with MBSE by translating them to S2ML.

The S2ML language is built around basic elements: blocks, ports, and connections. These three basic concepts, although very restricted, allow for a robust conceptualization of systems. Many languages can be interpreted through them, such as SysML, AltaRica, Modelica, and Lustre.

The S2ML models can be mathematically described as a quintuple $< P, C, B, \alpha, r >$ [18], where:

- P and B are sets of symbols called ports and blocks, respectively;
- C is a multiset of subsets of P called connections;
- α is a subset of $B \times (P \cup C \cup B)$ such as any element of $(P \cup C \cup B)$ that is associated with, at most, a unique element of B called its parent, and there exists a unique block $r \in B$ with no parent.

1.3.7 Multiphysics models

We explained in 1.3.3 that we need two different models when facing two different modeling intents. This results in diverse physical phenomena happening to the system being simulated using different models. This makes sense as, for example, the aerodynamics of an airfoil are a physical phenomenon entirely different from its deformation. Nonetheless, the deformation of an airfoil depends on the forces applied to it and, therefore, on the aerodynamic forces. Similarly, its aerodynamics depends on its geometry and might change when deformed. The multiphysics simulation domain seeks to answer this issue by providing means to correlate the computation of indicators that are interlinked [72].

Even though coupling models is an integral part of multiphysics, it requires computation times that are those of the simulations it uses. During the design of a system, it is often required to obtain quick results over its parameters, even if they will have a lesser precision. This results in the existence of means for presizing, which allows for simplification of the problem and multiphysics simulation. Some of these tools are simulink [84] and modelica [9]. They allow for simple multiphysics modeling of systems based on differential equations.

In the last case study of this thesis (Chapter 5) we use Modelica models of a drone in the consistency study. Through that example we show that SmartSync, and also the mathematical framework – S2ML+Cat – that we have designed in this thesis, allow for comparison of such models on top of the MBSE and MBSA ones.

1.4 Category theory

Category theory is the mathematical theory that we use to represent model consistency in this thesis. This last section of the State of The Art gives an introduction to category theory and to the categorical concepts used in the thesis.

1.4.1 History and interest

Category theory is a branch of mathematics that finds its origins in the work of Samuel Eilenberg and Saunders Mac Lane in the early 1940s. Together they introduced the concept of category in their 1945 paper [39]. Mac Lane later wrote "Categories for the Working Mathematician " [81] which is now widely reguarded as the reference book in category theory.

Category theory aims at unifying mathematics, especially by creating a bridge between Topology and Algebra. After category theory was used for many years to redefine existing concepts, Alexander Grothendieck built some completely new mathematical objects based on it [52]. Due to its high level of abstraction and because it took a long time before concrete applications were found, category theory has often been called "abstract nonsense" by the mathematic community [82].

Nowadays, category theory is often used in conjunction with logic. Olivia Caramello proposes the idea that category theory, and especially Grothendieck toposes, can be used as bridges to unify mathematics [27].

In this thesis, we finally used elements of category that are less complex than grothendieck's toposes, but the idea of using category theory for MBSE and MBSA model consistency stemmed from this translating potential that Olivia Caramello explored.

In this work, we use category theory to describe objects through their relations. We detail in this section some basic knowledge of category theory that is required to understand the thesis. A more complete yet well-written and easily understandable introduction to category theory can be found in courses and books such as [81], [111](for non mathematician scientists), [10], [101], [77], [96] (in french), [28](advanced notions around the grothendieck toposes), [89](from a programmer point of view), and many others.

1.4.2 Basic concepts: Categories and functors

A category is composed of objects and sets of morphisms between these objects. Morphisms are often also called arrows.

Definition 1.1. *Category [10]*
A category C is composed of:

- *A class Ob(C) of objects*
- *for $x, y \in Ob(C)$, the set $Hom_C(x, y)$ of morphisms from x to y*
  This set is called the Homset of x and y.

A category must respect the following rules:

- *Composition: If $f \in Hom_C(x, y)$ and $g \in Hom_C(y, z)$, then $g \circ f \in Hom_C(x, z)$.*

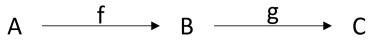


Figure 1.19: Model synchronization process using SmartSync [95]

- *Associativity: If f, g, h are morphisms such as* $g \circ f$ *and* $h \circ g$ *exist, then* $(h \circ g) \circ f = h \circ (g \circ f)$
- *Identity: for each* $x \in Ob(C)$*, there exists an identity morphism* $Id_x : x \longrightarrow x$

We can give a graphical representation of a category through an oriented graph. This representation is called a diagram over the category. Vertices represent the objects of the category, and edges represent the morphisms. A diagram does not have to be exhaustive over the category. Thus, we can draw a finite diagram over a category containing an infinity of objects. We often do not show identities and morphisms resulting from diagrams' composition, which allows better readability. Although there is a formal definition of what a diagram is, that can be found for example in Section 5.1 of [10], in this thesis we consider the diagrams in an intuitive way, as a graph within which the vertices are some objects of a category and the edges are some of the morphisms between these objects. An example of a diagram over a category that has three objects A, B and C, and two morphisms f: A $\longrightarrow$ B and g: B $\longrightarrow$ C can be found in Figure 1.19.

Just like other mathematical objects, we can link categories to eachother through applications called functors.

Definition 1.2. *Functor*
For two categories C, D, a functor F is an application $F : C \longrightarrow D$ *composed of:*

- *A function* $F_{Ob} : Ob(C) \longrightarrow Ob(D)$
- *For each* $x, y \in Ob(C)$*, a function* $F_{x,y} : Hom_C(x, y) \longrightarrow Hom_D(F_{Ob}(x), F_{Ob}(y))$

We can note F(x) or F(f) the images of an object x and a morphism f respectively. A functor $F : C \longrightarrow D$ *satisfies the following properties:*

- *identities are preserved by F, i.e. for* $x \in Ob(C)$, $F(Id_x) = Id_{F(x)}$
- *Composition is preserved by F, i.e. for* $x, y, z \in Ob(C)$ *and* $f \in Hom_C(x, y)$, $g \in Hom_C(y, z)$*, we have* $F(g \circ f) = F(g) \circ F(f)$

There exist applications between functors called natural transformations. Natural transformations are mappings for each object and morphism of a category between the images of their images by two functors. In this aspect, they allow transforming a functor into another functor. They are essential in defining the equivalences of categories.

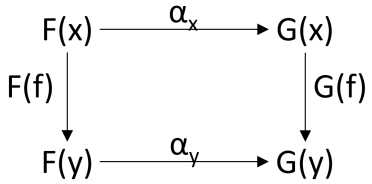


Figure 1.20: Illustration of a natural transformation

Definition 1.3. *Natural transformation*
Let C and D be two categories.
Let $F : C \longrightarrow D$ *and* $G : C \longrightarrow D$ *two functors.*
A natural transformation $\alpha : F \longrightarrow G$ *is composed, for each* $x \in Ob(C)$ *of a morphism* $\alpha_x : F(x) \longrightarrow G(x)$ *in D, such as for each* $y \in Ob(C)$*, for each* $f \in Hom_C(x,y)$*, we have:*
$G(f) \circ \alpha_x = \alpha_y \circ F(f)$
This corresponds to saying that the diagram over the category D given in figure 1.20 commutes.

1.4.3 Useful concepts

We have defined the basic concepts of category theory; we will now focus on some properties of interest to this work.

In the same way as functions can be injective, surjective, or both (bijective), there are similar concepts for functors between categories.

Definition 1.4. *Full and Faithful functors*
Let $F : C \longrightarrow D$ *be a functor.*
We say that F is Full if for any $x, y \in Ob(C)$, $F_{x,y} : Hom_C(x,y) \longrightarrow Hom_D(F(x),F(y))$ *is a surjection. We say that F is Faithful if for any* $x, y \in Ob(C)$, $F_{x,y} : Hom_C(x,y) \longrightarrow Hom_D(F(x),F(y))$ *is an injection.*

We can also define the concept of equivalence between two categories.

Definition 1.5. *Equivalence of categories*
Let C and D be categories.
An equivalence of categories is a functor $F : C \longrightarrow D$ *such as there exists:*

- *a functor* $G : D \longrightarrow C$
- *natural transformations* $\alpha : Id_C \longrightarrow G \circ F$ *and* $\alpha' : Id_D \longrightarrow F \circ G$

The pullback is also an interesting concept for our purpose. Prerequisites for this notion are cones and limits, so we shall define them first. Before defining these prerequisites, one must note that what is called a "triangle" in a

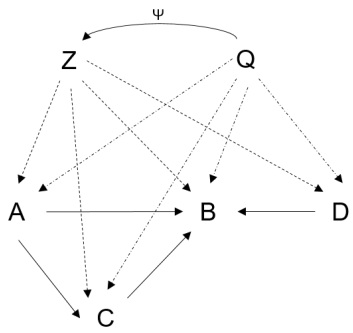


Figure 1.21: Illustration of a cone (Q) and the limit (Z)

diagram corresponds to three objects, with morphisms between them. For example we would have objects A, B and C, and morphisms f from A to B, g from B to C, and h from A to C. We say that the triangle commutes if $g \circ f = h$.

Definition 1.6. *Cone*
Let C be a category.
Let there be a diagram over C.
A cone over this diagram is an object Q with morphisms from Z to every object of the diagram, such as any newly formed triangle commutes.

A limit is a universal cone, i.e., a cone such as, for any other cone, a unique path exists from the limit to the other objects in the diagram through the cone.

Definition 1.7. *Limit*
Let C be a category.
Consider a diagram over C.
A limit on this diagram is a cone Z such as for any other cone Q we have a unique morphism $\psi : Q \longrightarrow Z$ such as all triangles including ψ commute.

Cones and limits are illustrated in Fig. 1.21. The considered diagram is constituted of the A, B, C and D objects and morphisms between them. Z and Q, assuming that compositions of morphisms of the diagram with the dotted arrows commute, are cones. Q is a limit of the diagram if for any cone Z, there is a unique morphism $\Psi : Q \longrightarrow Z$ that makes everything commute. It means that a limit is a universal cone, because for each cone there is a unique way to obtain this cone from the limit, through the ψ morphism.

Definition 1.8. *Pullback*
Let A, B and C three objects of a category.
Let f and g two morphisms from A and B to C.
A pullback in a category is the limit $Ax_C B$ of the diagram constituted of A, B, C, and g and f, illustrated in Fig. 1.22.

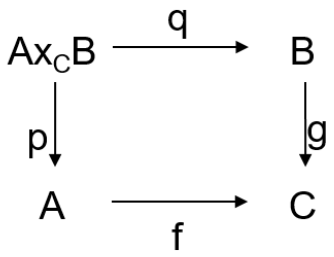


Figure 1.22: Structure of the pullback

The pullback is an object with morphisms towards two other objects that respects constraints with a third object. This can be interpreted as the biggest object that respects the constraints.

1.4.4 Use of category theory in Systems engineering

Because category theory is a tool that encompasses abstraction levels and interactions between objects, some approaches have used this mathematical framework to represent of complex systems.

Fundamental mathematical approaches give highly theoretical representations of systems. Schultz [108] represents dynamic systems using the concept of sheaves, which are functors from a category to the category of sets with compelling topological properties (The category of sets is the category whose objects are all the sets, and the morphisms are the applications between sets). This allows to represent interactions between multiple dynamic systems. Andrée Ehresmann [38], [37] uses category theory to represent systems in a way that allows for straightforward representation of hierarchy levels, the evolution of the system, and interactions between multiple systems.

Some approaches are closer to the Systems engineering point of view. For instance, [41] uses ontologies and category theory to represent MBSE models. Authors in [2] and [3] establish a category-based meta-model of Systems engineering and safety assessment.

CHAPTER 2
A CASE STUDY: THE LANDING GEAR

This chapter introduces the landing gear case study, which we use as a working example in the thesis. The first section presents the landing gear system and explains its use. We then present the modeling of the case study with MBSE and MBSA tools. Finally, we discuss the synchronization process of the models using the SmartSync methodology. This example is used in chapter 3 and chapter 4 to exemplify our proposed concepts.

2.1 The Landing Gear System

2.1.1 Motivations and approach

In this thesis, especially in chapter 4, we present concepts with high abstraction. In order to give a down-to-earth image of these concepts to the reader, we will use the example of a landing gear system. In this chapter, we present this case study, introduced in the literature by Boniol and Wiels [24], and the models that we have built with the assistance of an internship trainee at the ISAE-Supmeca Quartz Laboratory.

In order to provide a simple yet representative case study, we modeled the MBSE and MBSA points of view of the Landing Gear. We operated their comparison using the SmartSync methodology and tool. The MBSE model was created with Mohamed-Sami Kendel using the SysML tool MagicDraw. In parallel, I wrote the MBSA model using the AltaRica 3.0 modeling language.

We use this system as a demonstration of the MBSE and MBSA modeling methodologies and an application example to help understand the thesis. In Chapter 5, we use a second case study, a blood-delivery fixed-wing drone, to demonstrate the S2ML+Cat methodology introduced in this thesis.

2.1.2 Presentation of the system

The landing gear case study was described in [24] and served as a benchmark for techniques and tools for the assertion of system behavior. This system is a standard aircraft landing gear composed of three gears (front, rear-left, and rear-right). Figure 2.1 illustrates the global architecture of the system. The system is composed of the system pilot interface, its mechanical and hydraulic parts, and its digital control part. A more detailed depiction of the mechanical and hydraulic part of the system is found in Figure 2.2.

This system is also relevant to an MBSA study since aeronautic systems are required to comply with CS25 regulations – an aeronautic recommended practice describing safety analyses that are authorized to be completed on aircraft equipment for certification.

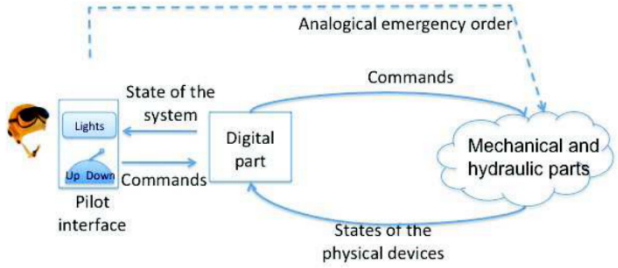


Figure 2.1: Global Architecture of the Landing Gear System [24]

For this work, two models of this system were created. The first one aims at modeling the system architecture, made with the Cameo System Modeler tool implementing the SysML language [48]. The second is a safety analysis view created using the OpenAltaRica tool implementing the AltaRica 3.0 modeling language [16]. To comply with the aeronautic certification requirement, these models must be separated and made by separated people. This allows safety analysis to independently verify the compliance of the architecture described in the MBSE model. In this work, we reproduced a realistic workflow with models written by two different people and did not eliminate differences before the final review. With this protocol, we want our workflow to present realistic differences and not avoid inconsistencies by not having independence between both models. The creation of the MBSA model was based on the MBSE model, and we aim to detect differences that occurred in this creation.

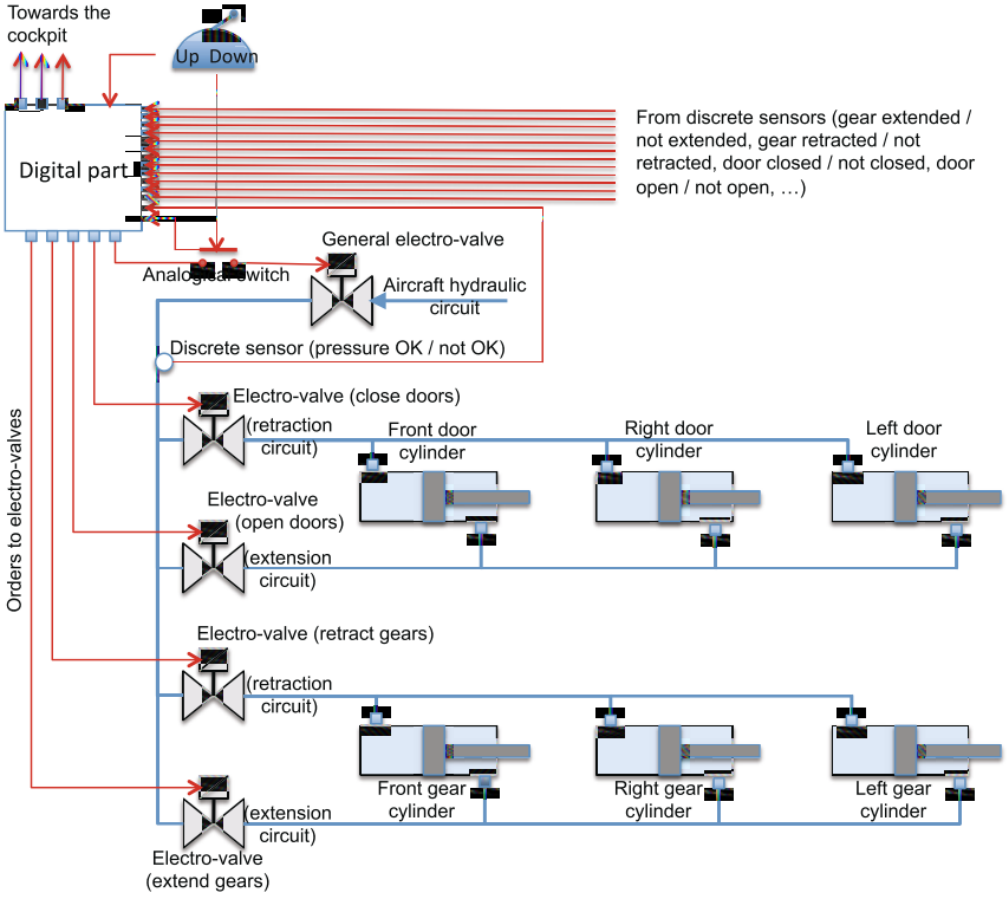


Figure 2.2: Architecture of the Hydraulic part [24]

2.2 Modeling the Landing Gear

2.2.1 MBSE Modeling

2.2.1.1 Methodology

In our study, the MBSE modeling was realized following the SysML methodology described in [86]. This methodology first focuses on a black box analysis of the system describing requirements, system context, lifecycle, and operational scenarios. Then some white box views of the system represent its functional and physical structure in addition to its behavior.

Our interest is in the structural and behavioral features of our models. They are contained within the white box views of the system, which is why for synchronization of the MBSA model, we only focus on these views.

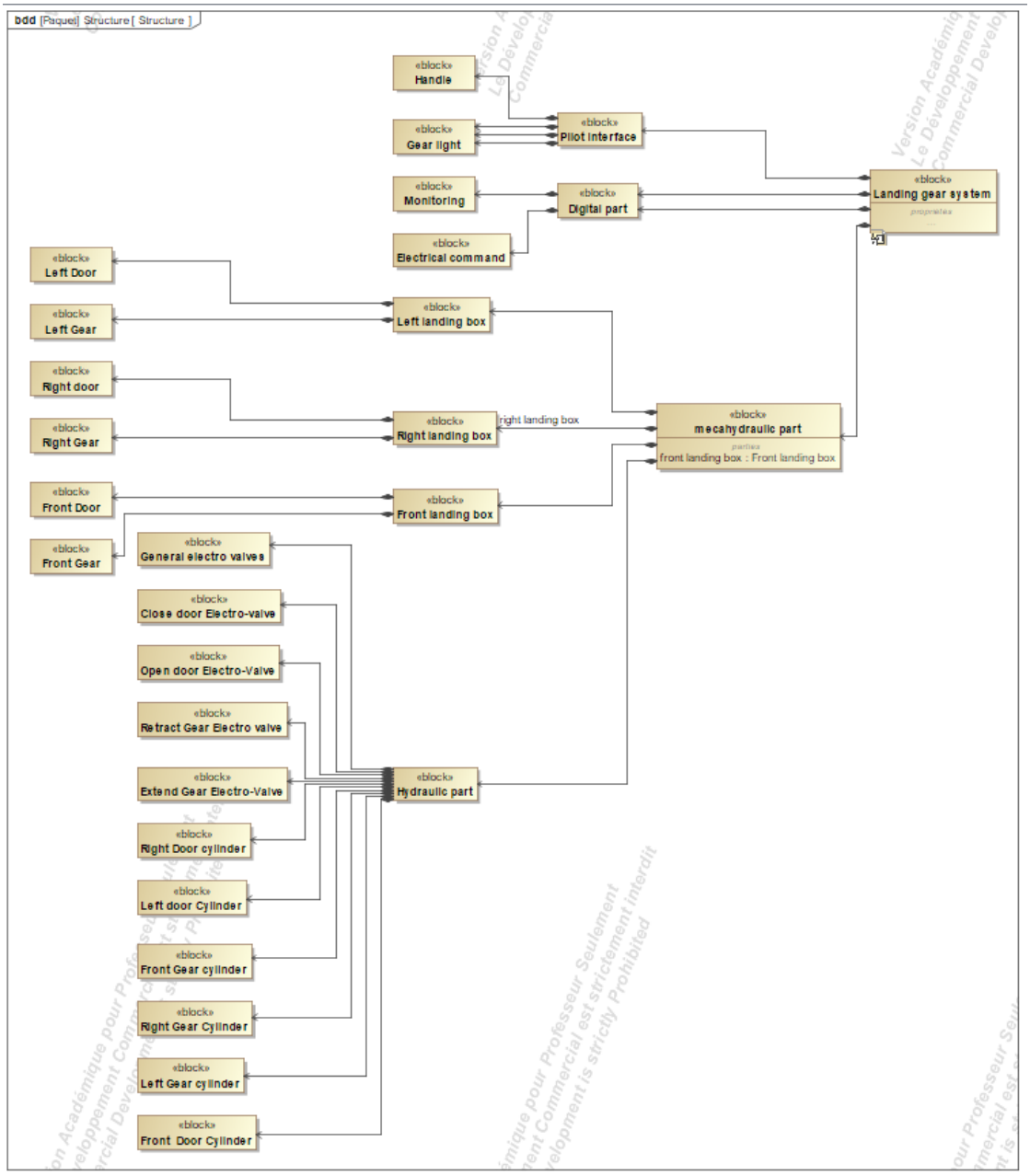


Figure 2.3: Block Definition Diagram of the Landing Gear System

2.2.1.2 Modeling

The system architecture is modeled around its three main subsystems, which can be observed in the Block Definition Diagram (BDD) shown in Fig.2.3.

This BDD shows the system's breakdown structure. Arrows in the diagram represent composition links, meaning that one block (or system) is composed of the components to which the arrow points. This view of the system shows us its structure but does not entirely define its architecture, as connections and flows between the components are not given. They are specified in the Internal Block Diagram (IBD) in Fig. 2.4.

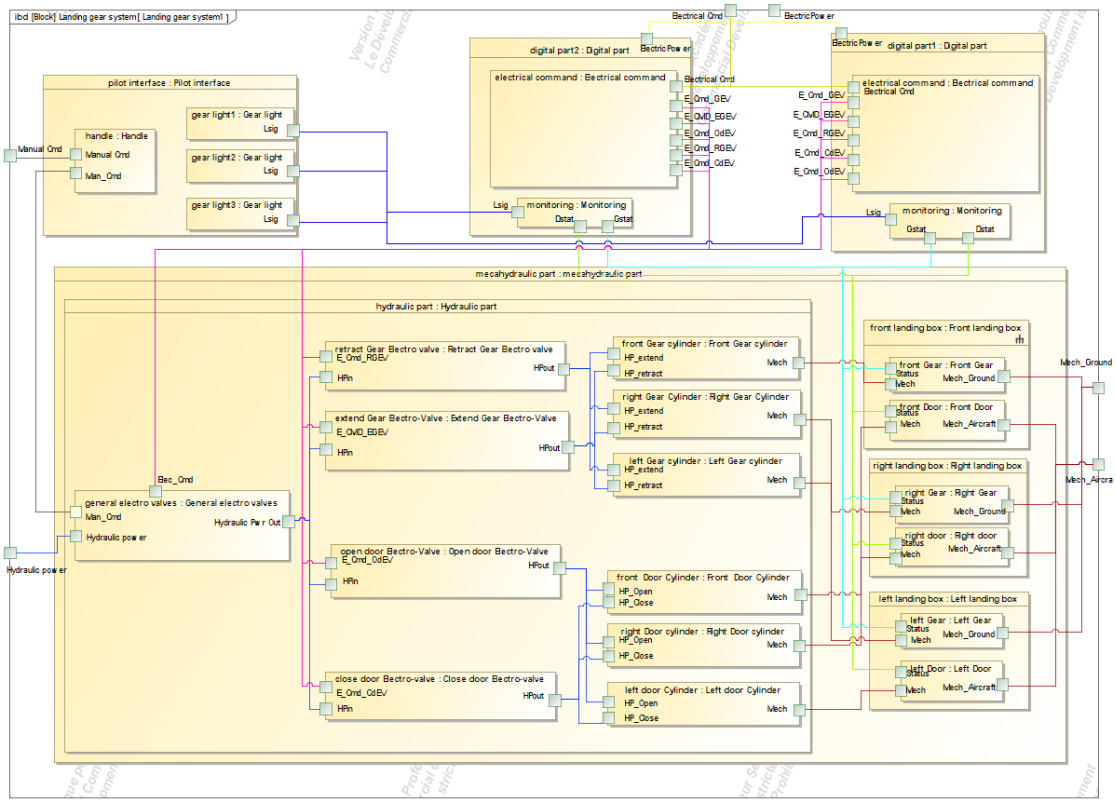


Figure 2.4: Internal Block Diagram of the Landing Gear System

In this IBD, we can observe the relations between components. The connections' names give us information about their type of flows. Finally, they can also contain typed variables. It is important to note that this model does not intend to simulate our system but is rather a communication tool over it and a means for traceability. Therefore, naming on this model is highly important and carries much information compared to usual simulation models where the content of variables, flows, and other quantitative values are the essential information carried by the model.

The two views we previously described are the ones that will be interesting in the context of MBSE/MBSA synchronization for the scope of this work.

2.2.2 MBSA Modeling

2.2.2.1 Methodology

The MBSA model we made for this work was created using the OpenAltaRica platform, based on the AltaRica 3.0 modeling language. It represents the system through its structure and dysfunctional behavior.

2.2.2.2 Modeling

The MBSA modeling was achieved using the article from Boniol [24] presenting the system as a reference document and based on the MBSE modeling. We here considered the MBSE model as a specification document of the system and expect to verify its compliance with safety requirements through MBSA modeling. Except for a few differences, which will be talked about in section IV, the model has a very similar structure to the one presented in the SysML IBD and BDD in Fig. 2.3 and Fig. 2.4 respectively.

```
domain nrpState {OK, KO}
class NonRepairableComponent
  nrpState s (init = OK);
  parameter Real lambda = 1.0e-5;
  event failure (delay = exponential (lambda));
  transition
    failure: s == OK -> s := KO;
end
```

Figure 2.5: Class NonRepairableComponent in AltaRica 3.0

In AltaRica 3.0, we represent the system by a main "block"; a container that will be considered and simulated by the AltaRica 3.0 compiler. The components of our system are described in classes that are instantiated in the main block. Fig. 2.7 gives the code of the main block representing the system, and Fig. 2.8 and Fig. 2.9 describe the class ElectroValve and the class Cylinder respectively which are components of the landing gear system. Instances of those components are linked in the system to allow for retraction and extension of the landing gear. Assertions such as the one presented in Fig. 2.6 allow for the input of the cylinder to be at all times equal to the value of the output of the electrovalve.

```
digitalPart.CPIOM1.input:= pilotInterface.udHandle.output;
```

Figure 2.6: example of an assertion in AltaRica 3.0

All components in our system are extending the NonReparaibleComponent class presented in Fig. 2.5. This class describes the state machine for the failure of a component. Delays characterize those failure events. This allows the execution to compute the time after which the transition shall be fired. Delays are described using probability distributions such as, in our case study, an exponential distribution. They also allow computation of the probability of an event happening for the generation of fault trees in the case of static systems.

We derive all system components from this general class, specializing this class by adding new variables that represent the ports of our components and assertions that represent connections between these variables. As an example Fig. 2.6 represents such a connection. It means that the output value of the udHandle component of the pilot interface is given to the input of the CPIOM1 component of the digital part.

This means that during the execution, the value of the output of the udHandle component, which is the handle used by the pilot to actuate the landing gear, will be affected to the input of the first CPIOM unit of the digital part. Assertions can also be used to give values to variables based on component state or other information.

The interest of having this formal representation of the system rather than using a notation such as SysML is that it allows for formal computation over the system safety. Thanks to this model, we can compute minimal cut sets of the system with their probabilities. We can also execute a stochastic simulation of the system with failures and identify propagation paths of the failures. This would not be possible if there were any ambiguity in the representation of the system. This is not an issue for human communication with the MBSE model.

```
block LandingSys
  PilotInterface pilotInterface;
  MechaHydraulicalPart mechahydraulicPart;
  DigitalPart digitalPart;
  assertion
    digitalPart.CPIOM1.input := pilotInterface.udHandle.output;
    [...]
end
```

Figure 2.7: Block Landing System in AltaRica 3.0 (some assertions were hidden for clarity)

```
class ElectroValve
  extends NonRepairableComponent (lambda = 1.0e-6);
  Boolean input, output, order (reset = false);
  assertion
    output := if s == OK then input and order else false;
end
```

Figure 2.8: Class ElectroValve in AltaRica 3.0

```
class Cylinder
  extends RepairableComponent;
  Integer input (reset = 0);
  Boolean output (reset = 0);
  assertion
    output := if s == OK then input else false;
end
```

Figure 2.9: Class Cylinder in AltaRica 3.0

2.2.3 State Machines

2.2.3.1 Gear lights state machine

The first state machine example that we will consider represents the status lights for the landing gear. This example was first introduced as part of the landing gear reference architecture in [24], structural comparison of MBSE and MBSA models for this architecture can be found in section 2.3.

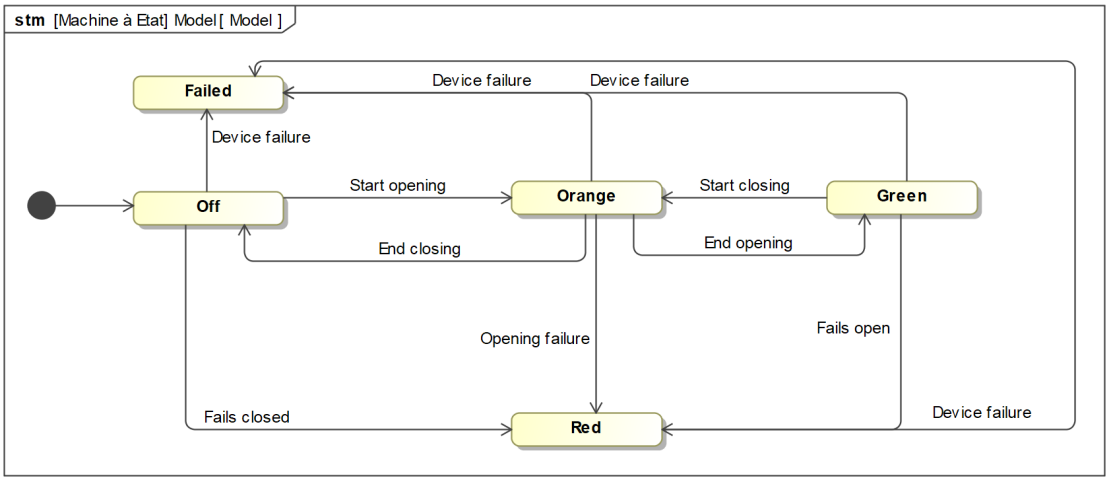


Figure 2.10: SysML state machine of the gear light behavior

Three lights indicate to the pilot the status of all three gears using the following code:

```
block systemLight
  GearLight light;
end

domain GearLightState {off, green, orange
                       , red, failed}
class GearLight
  GearLightState s (init = off);
  event openingSequenceStarts, openingSequenceEnds,
     openingSequenceFails, closingSequenceStarts,
     closingSequendEnds, closingSequenceFails, failsopen,
     failsclosed, failsred, failsorange, failsgreen, failsoff;
  transition
    openingSequenceStarts: s == off -> s := orange;
    openingSequenceEnds: s == orange -> s := green;
    openingSequenceFails: s == orange -> s := red;
    closingSequenceStarts: s == green -> s := orange;
    closingSequendEnds: s == orange -> s := off;
    closingSequenceFails: s == orange -> s := red;
    failsopen: s == green -> s := red;
    failsclosed: s == off -> s := red;
    failsred: s == red -> s := failed;
    failsorange: s == orange -> s := failed;
    failsgreen: s == green -> s := failed;
    failsoff: s == off -> s := failed;
end
```

Figure 2.11: AltaRica 3.0 model of the gear light use case

- Green light: "Gear locked down";
- Orange light: "Gear maneuvering";
- Red light: "Landing gear system failure";
- No light: "Gear locked up".

The MBSE view of this state machine is given in figure 2.10 and the MBSA view ing figure 2.11.

2.2.3.2 Redundant system

the second case study we will use is a generic redundant system composed of two components: the main unit that is normally used and a spare unit that is used when the main one is down.

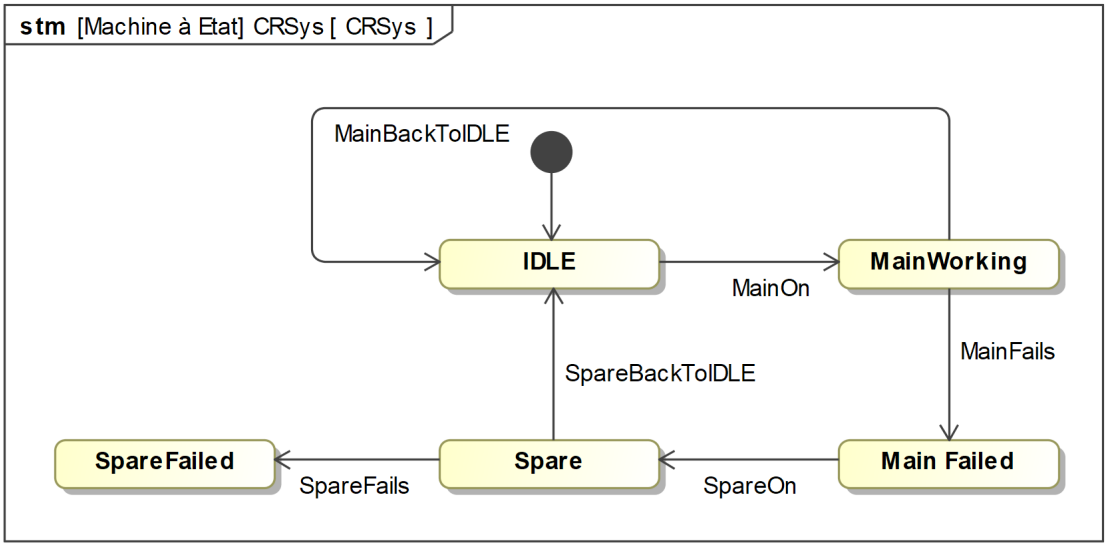


Figure 2.12: SysML state machine of the redundant system

The MBSE models the system's behavior through a general state machine that shows the system's global behavior, which is given in figure 2.12. The MBSA code for the system is represented in figure 2.13.

It can be noticed that the MBSA model instantiates two components that represent the main and spare units of the system. These components both embed a GTS that has "WORKING", "FAILED" and "STANDBY" states and transitions from WORKING to FAILED, from STANDBY to WORKING, and from STANDBY to FAILED. Compilation of this model will result on the composition of both these state machines. Note that those state machines are very different in their structures. The first one shows the system's global behavior, and the second computes global behavior by the composition of the behaviors of the two components.

2.3 Synchronization of the models

Once the MBSE and MBSA models are built, we want to assert consistency between them. This assertion is done using the SmartSync methodology and tool (see section 1.3.5).

This consistency assessment between the MBSE and MBSA models of the landing gear results in the study presented in section 3.1. We will also use this consistency assessment as our working example in Chapter 4.

```
domain UState {WORKING, FAILED, STANDBY}

class Unit
  UState vsState (init = WORKING);
  Boolean vfDemanded (reset = false);
  parameter Real pFailure = 1.0e-5;
  parameter Real pFailureOnStart = 1.0e-4;
  event evFailure (delay = exponential(pFailure));
  event evStart (delay = Dirac(0.0),
   expectation = 1 - pFailureOnStart);
  event evFailedToStart (delay = Dirac(0.0), expectation =
     pFailureOnStart);
  transition
    evFailure: vsState == WORKING -> vsState := FAILED;
    evStart: vfDemanded and vsState == STANDBY -> vsState :=
       WORKING;
    evFailedToStart: vfDemanded and vsState == STANDBY -> vsState
       := FAILED;
  Boolean vfInput, vfOutput (init = false);
  assertion
    vfOutput := vfInput and vsState == WORKING;
end

block System
  Unit Umain;
  Unit Uspare(vsState.init = STANDBY);
  Boolean vfOutput (reset = false);
  assertion
    Umain.vfDemanded := true;
    Uspare.vfDemanded := Umain.vfOutput == false;
    Umain.vfInput := true;
    Uspare.vfInput := true;
    vfOutput := Umain.vfOutput or Uspare.vfOutput;
    observer Boolean oTE = vfOutput == false;
end
```

Figure 2.13: AltaRica 3.0 model of the redundant system use case

2.3.1 Translation to S2ML

The first step for consistency assessment of the MBSE and MBSA models is to translate them to the S2ML formalism. Although it should be possible in the future to automate the translation, it currently needs to be done manually.

SysML to S2ML translation:

As we intend to compare the architecture of the landing gear, we are not interested in the whole SysML model. We only need to translate the content of the IBD (Fig 2.4) and BDD (Fig 2.3).

We translate the different model elements from SysML to the corresponding concepts in S2ML, following the correspondence table 2.1.

SysML	S2ML
partproperty	block
port	port
connection	connection

Table 2.1: Mapping between concepts for translation between SysML and S2ML

As an example, we can study the translation of the `PilotInterface` partproperty, which is illustrated in Fig 2.14.

In SysML this block contains four other partproperties : the `Handle` and `Gearlight1`, `Gearlight2` and `Gearlight3`. The `Handle` contains two ports, `Manual_Cmd` and `Man_Cmd`, and each `Gearlight` contains a port `Lsig`.

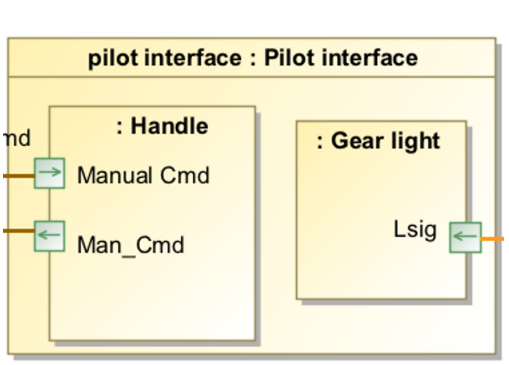
We translate the `PilotInterface` to a S2ML block `PilotInterface`, which contains the four blocks `Handle` and `Gearlight1,2,3`. The block `Handle` contains ports `Manual_Cmd` and `Man_Cmd`, and each `Gearlight1,2,3` a port `Lsig`.

The complete translated S2ML model can be found in Appendix B.2.

Because the SmartSync tool does not consider the connections, we do not include them in this translation. However, examples of translation of the connections for SysML and AltaRica 3.0 can be found in Chapter 5, along with some leads as to how connections could be included in the comparison.

AltaRica 3.0 to S2ML translation:

Contrary to the SysML model, we translate the whole AltaRica model to S2ML. We, however, eliminate some elements, such as domains, state variables, events, and transitions. In this case for simplicity we do not translate the safety specific elements, we do it in the second case study presented in chapter 5. Ideas for translations and comparison of state machines between




```
block Pilot_interface
    block Handle
        port Manual_Cmd;
        port Man_Cmd;
    end
    block Gearlight1
        port Lsig;
    end
    block GearLight2
        port Lsig;
    end
    block Gearlight3
        port Lsig;
    end
end
```

(**a**) (**b**)

Figure 2.14: The SysML pilot Interface (**a**) and its translation in the S2ML model (**b**).

SysML and S2ML can be found in section 3.2. In the same way, we use correspondence table2.2 to manually translate the model elements from the AltaRica 3.0 model to S2ML.

AltaRica 3.0	**S2ML**
block	block
variable	port
assertion	connection

Table 2.2: Correspondance of concepts for translation between SysML and S2ML

Although the AltaRica 3.0 model's translation is more direct as AltaRica 3.0 and S2ML share the same structure [100], it requires more work. The original AltaRica 3.0 model contains classes that are instantiated as blocks; therefore, we need to instanciate the model to translate it to S2ML for comparison (or to translate it to S2ML and then instanciate the S2ML model).

The AltaRica 3.0 and its S2ML translation can be found in Fig 2.15. As previously claimed, we can observe that the AltaRica 3.0 model contains classes that are not yet blocks.

We instanciate the AltaRica 3.0 code to the code found in Fig 2.16. The instanciation process consists of instantiating all the blocks declared in the model that refers to classes.

As an example, the declaration `"Gearlight frontGearLight"` in class PilotInterface becomes a block `"frontGearLight"` that contains the Booleans `input` and `output`, as well as an assertion from the class GearLight. Because this class extends the class `NonRepairableComponent`, it also contains the state variable `nrpState s`, the event `failure` and the transition `failure`. Once the

```
domain nrpState {OK, KO}
class NonRepairableComponent
    nrpState s (init = OK);
    event failure (delay = Dirac(0));
    transition
        failure: s == OK -> s := KO;
end

class PilotInterface
    GearLight frontGearLight;
    GearLight leftGearLight;
    GearLight rightGearLight;
    UpDownHandle udHandle;
    AnalogicalSwitch analogicalSwitch;
end

class GearLight
    extends NonRepairableComponent;
    Boolean input, output (reset = false);
    assertion
        output := if s == OK then input else
             false;
end

class UpDownHandle
    extends NonRepairableComponent;
    Integer output (reset = 0);
    Integer position (reset = 1);
    assertion
        output := if s == OK then position
             else 0;
end

class AnalogicalSwitch
    extends NonRepairableComponent;
    Boolean recentChange (reset = false);
    Boolean output (reset = false);
    assertion
        output := if s == OK then
             recentChange else false;
end
```

```
block pilotInterface
    block frontGearLight
        port input;
        port output;
    end
    block leftGearLight
        port input;
        port output;
    end
    block rightGearLight
        port input;
        port output;
    end
    block udHandle
        port output;
        port position;
    end
    block analogicalSwitch
        port recentChange;
        port output;
    end
end
```

(**a**)

Figure 2.15: The AltaRica 3.0 pilot Interface (**a**) and its translation in the S2ML model (**b**).

instanciation of the model is complete, we can translate it to S2ML in a similar way to the SysML model.

The block `frontGearLight` from the AltaRica 3.0 model is translated to a S2ML block `frontGearLight` that contains a port input and a port output. As we explained before, because they are not relevant to the comparison, we do not transcript the state variable, event, and transition.

We could translate the assertion "`output := if s == OK then input else false;`" to a connection "`connection [input, output](type = "assertion")`". Because we want to keep things simple in this example, and the SmartSync tool does not yet consider connections, we choose not to translate the connections.

The complete translated model can be found in annex B.3.

```
block PilotInterface
    block frontGearLight
        nrpState s (init = OK);
        Boolean input, output (reset = false);
        event failure (delay = Dirac(0));
        assertion
            output := if s == OK then input else false;
        transition
            failure: s == OK -> s := KO;
    end
    block leftGearLight
        nrpState s (init = OK);
        Boolean input, output (reset = false);
        event failure (delay = Dirac(0));
        assertion
            output := if s == OK then input else false;
        transition
            failure: s == OK -> s := KO;
    end
    block rightGearLight
        nrpState s (init = OK);
        Boolean input, output (reset = false);
        event failure (delay = Dirac(0));
        assertion
            output := if s == OK then input else false;
        transition
            failure: s == OK -> s := KO;
    end
    block udHandle
        nrpState s (init = OK);
        Integer output (reset = 0);
        Integer position (reset = 1);
        event failure (delay = Dirac(0));
        assertion
            output := if s == OK then position else 0;
        transition
            failure: s == OK -> s := KO;
    end
    block analogicalSwitch
        nrpState s (init = OK);
        Boolean recentChange (reset = false);
        Boolean output (reset = false);
        event failure (delay = Dirac(0));
        assertion
            output := if s == OK then recentChange else false;
        transition
            failure: s == OK -> s := KO;
    end
end
```

Figure 2.16: The instanciated AltaRica 3.0 Pilot Interface

2.3.2 Comparison with SmartSync

Once the translation to S2ML is complete, we can compare the models using the SmartSync platform. In this thesis, all comparison work done using SmartSync was done with the SmartSync platform v0.0.1, hence some shortcomings of the tool that will arise in this example, which may be corrected in future versions.

A comparison using the SmartSync tool is done iteratively, through the process described in section 1.3.5 and illustrated in Fig. 1.18. A more down-to-earth version of these steps can be expressed as follows.

- The S2ML Compiler is used to compile the SmartSync textual models to compiled XML files.
- The XML files are given as inputs to the S2ML Comparator, and an output file is specified
- The SmartSync comparator outputs a CSV file containing three columns:

  Type: This column contains the model element type, which can be a block, a port, or a connection

  Model1: This column contains the model element name for the first XML model input

  Model2: This column contains the model element name for the second XML model input
  This file contains the model elements at the first level of abstraction of the model, i.e., direct subsystems, ports that are contained by no other block than the main block, and connections that are contained by no other block than the main block. They are not yet aligned.
- The user aligns each element of column Model1 to its corresponding element in column Model2. The attribute `"forget"` can be given to elements that do not have a counterpart when the user can justify why they do not.
- The processed CSV file is given as an input to the SmartSync tool, along with the XML models
- The SmartSync tool outputs a new CSV file containing the same columns with children elements of the ones that have already been aligned.
- The user processes the CSV file as before and iterates with the SmartSync tool until no new model element is found.
- Model elements that have no counterpart and were not assigned the `"forget"` attribute in the last processed CSV file are considered inconsistent.

When we first input the XML models for the landing gear system, the SmartSync Tool outputs the table 2.3. In this example, we choose Model1 as the MBSE model and Model2 as the MBSA model.

Type	Model1	Model2
	main.Landing_gear_systeme	main.landingSys
port	main.Landing_gear_systeme.ElectricPower	
port	main.Landing_gear_systeme.Electrical_Cmd	
port	main.Landing_gear_systeme.Hydraulic_power	
port	main.Landing_gear_systeme.Manual_Cmd	
port	main.Landing_gear_systeme.Mech_Aircraft	
port	main.Landing_gear_systeme.Mech_Ground	
block	main.Landing_gear_systeme.Digital_part1	
block	main.Landing_gear_systeme.Digital_part2	
block	main.Landing_gear_systeme.Pilot_interface	
block	main.Landing_gear_systeme.mecahydraulic_part	
block		main.landingSys.digitalPart
block		main.landingSys.mechahydraulicPart
block		main.landingSys.pilotInterface

Table 2.3: First output file of the SmartSync comparison

We notice that the MBSE model contains ports at the system's frontier, whereas the MBSA model does not. These ports correspond to interfaces between the system and its environment, for example, its power input. In this case, we deemed it normal not to consider them in the safety study and gave them the attribute "`forget`". In a "real-world" safety study, it cool be argued that we need to consider the use cases where they fail. This highlights that the engineers need to operate this part of the model comparison. This choice cannot be automated.

We can align the MBSE model's "`Pilot_Interface`" and "`mecahydraulic_part`" to the MBSA model's "`pilotInterface`" and "`mechahydraulicPart`".

We note that the "`Digital_part1`" and "`Digital_part2`" from the MBSE model only correspond to the "`digitalPart`" of the MBSA model. This is because the MBSA model encapsulates CPIOM1 and CPIOM2 (Core Processing & Input/Output Modules) inside the Digital part at a lower abstraction level. In contrast, the MBSE model considers redundancy at this abstraction level. The SmartSync tool currently does not allow to consider such exceptions; we suggest that this could be overcome by adding a keyword "`skip`" that would allow skipping to the children elements. The mathematical framework introduced in chapter 4 would allow to specify such a feature.

For the rest of this example we considered the digital part out of the scope of the comparison. We could also correct one of the models to change the level of abstraction and operate the comparison or create standalone models of the DigitalPart1/DigitalPart2 and CPIOM1/CPIOM2 and operate the comparison over these models.

This first step of comparison results in the processed table 2.4.

Type	Model1	Model2
	main.Landing_gear_systeme	main.landingSys
port	main.Landing_gear_systeme.ElectricPower	forget
port	main.Landing_gear_systeme.Electrical_Cmd	forget
port	main.Landing_gear_systeme.Hydraulic_power	forget
port	main.Landing_gear_systeme.Manual_Cmd	forget
port	main.Landing_gear_systeme.Mech_Aircraft	forget
port	main.Landing_gear_systeme.Mech_Ground	forget
block		main.landingSys.digitalPart
block	main.Landing_gear_systeme.Digital_part1	
block	main.Landing_gear_systeme.Digital_part2	
block	main.Landing_gear_systeme.Pilot_interface	main.landingSys.pilotInterface
block	main.Landing_gear_systeme.mecahydraulic_part	main.landingSys.mechahydraulicPart

Table 2.4: Processed file for the first step of comparison

We iterate this process for each abstraction level of the models; this corresponds to four iterations. The final result table can be found in appendix C.7.

Some model elements have the mention MCB as their type. This means the tool output "`Missing Corresponding Block`" when we aligned them because they were aligned to a block in another branch of the Product Breakdown. In the same way as before, even though SmartSync cannot currently handle this case, our mathematical framework can do so by reaching back to the closest common ancestor of both blocks. Therefore this case could probably be handled in future versions of SmartSync.

2.3.3 Results of the comparison and actions taken on the models

After the comparison's fourth iteration, we detected several inconsistencies. Twenty elements of the MBSE model have no counterpart in the MBSA model. Twenty-three elements of the MBSA model have no counterpart in the MBSE model. These inconsistent elements can be found at the bottom of the table in appendix C.7. They are the ones with empty cases for either the Model1 or Model2 element.

We will not analyze all the inconsistencies and solutions here but only focus on one example of a misunderstanding resulting in inconsistencies. An analysis of the differences (not to be confused with inconsistencies) between the models we found during the comparison can be found in section 3.1.

The MBSE model describes separate electro-valves for actuation of the front, right, and left landing gears, each time one for extension and one for retractation. The MBSA model describes one extension electro-valve, and one

retractation electro-valve, that operate all three gears. The same description is done for the electro-valves used for opening and closing the gear doors.

The original paper by Boniol and Wiels [24] describes the hydraulic architecture with one extension valve and one retraction valve for all three gears (see Fig. 2.2), which allows for the synchronized use of the gears. During our design of the models, we considered this article as the specification of the system. Therefore we deem that there is an error in the MBSE model.

We correct the MBSE model by removing the valves and replacing them with unique opening/closing Gear Electro-valves and opening/closing Door Electro-valves.

2.3.4 Need for consistency assessment and a mathematical frame

The comparison we operated between the MBSE and MBSA models for the landing gear shows that many inconsistencies can appear in the design of a system's models.

In this case, 43 model elements are deemed inconsistent after the comparison. Considering that this is not a real-life system, we can consider that these are small models, and we did not consider connections in the comparison.

The inconsistencies we detected highlight the need for consistency assessment between MBSE and MBSA. Synchronization methodologies can also have flaws, as we have shown with SmartSync that we had difficulties encompassing some differences in the abstraction of the models. We also understand that SmartSync does not consider the connections and that, in general, synchronization methodologies have a scope of inconsistencies that they address.

This thesis aims to contribute to synchronization methodologies by providing a mathematical framework based on category theory. With this framework, we intend to make it possible to formally demonstrate the efficiency of a synchronization methodology by showing which elements are compared based on which criteria. In this context, in Chapter 4, we propose the S2ML+Cat framework, built based on the concepts of ports, blocks and connections of S2ML. In Chapter 5, we exemplify this framework with the SmartSync tool, to show that our framework is compatible with the SmartSync methodology.

CHAPTER 3

MBSE AND MBSA SYSTEM MODELS DIFFERENCES

In this chapter, we study the differences between MBSE and MBSA models. We want to introduce the notion of inconsistency in a way that allows us to separate it from other differences. For that purpose, we propose a typology of the differences we observed in comparing the landing gear models.

We also remark that the models contain structural and behavioral aspects, and our consistency assessment has only concerned the structural aspect. Therefore we study the possibility of a consistency assessment over the state machines contained in both MBSE and MBSA models.

3.1 Typology of the MBSE and MBSA Models Differences

MBSE and MBSA are two parts of the system's design, that serve two different purposes. Thus, differences occur between those models. Whether for lack of communication between the teams or more fundamental reasons linked to the nature of those models.

These differences could lead to models representing two distinct and different systems instead of the same ones. We call these differences inconsistencies. Synchronization between MBSE and MBSA models is thus necessary to ensure consistency. It is also important to note that not all differences between the models are inconsistencies.

This section proposes a typology of the differences between MBSE and MBSA models. Through this typology, we understand better the mechanisms that lead to differences. We also clarify the distinction between inconsistencies and the rest of the differences.

3.1.1 Model comparison

We reviewed both MBSE and MBSA models together and searched for all differences between them. This review, described in Section 2.3, allowed us to identify types of differences. We do not cite all the differences here, but we list all the different kinds we encountered.

Naming differences:

The first differences that occur when reviewing the models are in the names of the elements of the model. The names can vary between both models. For example, the handle from the pilot interface is called "Handle" in the MBSE model and "udHandle" in the MBSA model.

Naming differences result from the system engineer and the safety analyst calling the same elements differently.

They can happen because of their different technical backgrounds or naming practices (upper/lower cases, shortcuts...). Other naming differences come from the naming rules in both tools. These rules can be either restrictions or simply "best practices". For example, the pilot interface subsystem is called "Pilot interface" in the MBSE model and "pilotInterface" in the AltaRica 3.0 model. This difference results from the differences in uses of the two models: AltaRica 3.0 is a formal modeling language that allows the computation of reliability or safety indicators. Thus objects are represented in one character sequence that follows strict rules and cannot contain a space. SysML is used to declare and communicate. As a consequence, there are no restrictions on model elements names.

It also happens that some elements of both models can sometimes not be named, whereas their counterpart in the other model is. Assertions in AltaRica 3.0 are unnamed. They serve the purpose of simulation, defining rules for calculating variables' values. AltaRica 3.0 variables must be named. On the contrary, connections in the SysML model can be named. For example, using the type of flow or actions they convey. However, the engineer often does not name the ports they connect; they are automatically named p1, p2, pk by the modeling tool. Such differences result from the different intents of the two models and the modeling habits of both engineers.

Variable types differences:

The system engineer and the safety analyst do not have the same view on the system. Therefore they do not represent system variables in the same way.

Variable types are sometimes not assigned to the ports in the SysML model but rather to the connections, depending on the engineer's modeling habits. The system engineer wants to prove that the system answers requirements that can be linked to those values. Thus, MBSE variables are usually typed as their physical unit. Because they are mainly interested in knowing whether they are

nominal or dysfunctional and not by their accurate values, safety analysts often type variables with booleans or discreet domains such as [OK, KO] or even domains with multiple values such as [OK, KO, ERRONEOUS]. Thus, most variables in the AltaRica model are typed with discrete values.

Structural differences:

We also observe diverse structural differences between the models.

The first one is that some subsystems are specified at different levels of abstraction. For example, the arborescence for the first digital part is "`Landing gear System/Digital Part`" in the MBSE model and "`LandingSys/digitalPart/CPIOM1`" in the MBSA model (note that there is also a significant naming difference).

In the same way, the arborescence for the electro-valve that brings hydraulic pressure to the door extension cylinders is "`Landing gear system/mecahydraulic part/Hydraulic part/Open door Electro-Valve`" in the MBSE model and "`LandingSys/mechahydraulicPart/hydraulicSys/DoorHydraulicSys/extensionDoorElectroValve`".

We have spoken in section 2.3.3 about an inconsistency in the hydraulic part representation, which we suppose solved here.

Although the naming differs, subsystem levels between those paths match, apart from the "DoorHydraulicSys" level in the MBSA model that does not match any subsystem in the MBSE model. This difference originates from the Safety analyst grouping parts differently. He may want only to consider the failure of a group of parts rather than every unique part. It could also happen that the MBSA model only specified "DoorHydraulicSys" without modeling the electro-valve inside it.

Some components are also specified at different places in the Product Breakdown Structure. This is the case with the cylinders in our comparison. The system engineer considered them part of the hydraulic parts, whereas the safety analyst put them in the landing sets along with the gears/doors they are connected to. Such differences could be caused by a different point of view over the system or by a modeling error. In this case, we considered the models to be consistent. Finally, in the Internal Block Diagram, we observe some connections to the outside of the system that are not considered in the AltaRica model. For example, it is the case of the "Electric Power" input. The system engineer must represent the interactions between the system and its environment. However, even though this connection has a tangible impact on the system, it was not considered relevant for the safety analysis of the landing gear system and thus not modeled with AltaRica. This difference can be either considered a modeling error or not, depending on whether that connection has an impact or not on safety analysis.

Although it is not the case in our models, some connections could have

been placed between ports that do not necessarily exist in the MBSE or MBSA representations of the system. This could either occur by modeling error or because those values are irrelevant to one or the other modeling intent.

3.1.2 Types of differences

We are interested in understanding how differences arise between MBSE and MBSA models. Although we need to identify inconsistencies between the models, as they are a source of errors in the design, the models cannot be identical. The absence of differences would mean that they contain the same exact information. Whatever use can be achieved with one would also be achievable with the second. There would be no need for two separate models. To understand these differences, we analyze their sources.

From the comparison, we deduce three main types of differences in our model related to the cause of the differences between the models. Along with the kind of differences that we observed, these types can be found in Table 3.1. Some of those differences result from modeling errors and lead to the models describing different systems; we call them inconsistencies. However, we also note differences that are not problematic and sometimes necessary.

Differences due to Modeling tools and practices:

The first type of difference is related to differences caused by the different modeling tools and practices. Examples of this are different naming rules, connections between ports, or assertions that relate to variables carrying names differently. This type of difference could be handled by modeling practices or rules in some instances, for example, the implementation of naming rules in SysML similar to those in formal modeling languages. It could also be considered in the comparison by not taking into account names that have no counterpart or by cleverly comparing them, for example, connections names in SysML to variables names in AltaRica. Note that the naming differences may be reduced through the company quality process.

Differences due to modeling intent:

The second type of differences that we denote are differences linked to modeling intent. For example, we observed differences in abstraction between both models, as illustrated with the electro-valve. It is also the case with some variables and connections with physical unit types in MBSE and boolean/discrete types in MBSA. These differences are necessary for the system engineer and safety analyst to work in a sound modeling environment. Their existence is the reason for having two separate models rather than modeling all information in one global model.

Type	Observed differences
Due to Modeling tools and Practices	Different naming rules
	Different name meaning
	No naming counterpart
	Different abstractions of subsystems
Due to Modeling Intent	Different variable types
	Different model arborescence
	Different interactions/connections in and with outside the system
Due to Modeling Errors	Wrong model arborescence
	Wrong Variable Values/Types
	Wrong connections between system components

Table 3.1: Typology of the differences between MBSE and MBSA system models

Differences due to modeling errors:
The third type of differences is differences caused by modeling errors. Model synchronization aims to eliminate those differences. They can be different naming, wrong values, incorrect links between components, etc. We believe these inconsistencies to be more difficult to recognize from the second type of differences (modeling intent) than the first (modeling practices and tools). For example, the differences in structure that we observed can often be interpreted as inconsistencies or differences due to modeling intent, depending on whether we estimate them to be relevant.

We also raise another interesting way to classify differences. We encountered differences between the models related to a particular system element, such as a naming difference or a variable type difference. We also encountered differences related to the model's structure, such as abstraction differences, different placement of an element in the Product Breakdown Structure, or faulty connections between component ports/variables. These two types of differences are also interesting because they have different interpretations in a mathematical framework around the models.

3.1.3 Discussion

This typology shows that differences between MBSE and MBSA models can be sorted by their causes. These sources of differences are the use of different modeling tools and practices, different modeling intents, and, finally, modeling errors.

Differences due to modeling intents and standards are essential to the models since they are the reason two models are used instead of one and should be preserved. On the contrary, inconsistencies due to modeling errors should

be eliminated from the model. Engineers might also want to reduce differences due to different modeling tools and practices as much as possible. For example, the unification of naming conventions between systems engineering and safety assessment could be used to reduce naming differences.

3.2 Structural state machine consistency

Behavioral consistency is one of the difficulties raised by the need for synchronization between system architecture and safety analysis. The system architecture and safety models represent the system behavior but with different intents, making these representations very different. The SysML notation, one of the most widely used MBSE languages, does provide multiple diagrams that allow for behavioral modeling. One of these diagrams is state charts, which are refined state machines. Similarly, the AltaRica 3.0 language, an MBSA formalism, uses guarded transition systems (GTS) to represent the system's behavior. In this section, we study the structural comparison of these state machines. We do not intend to provide a behavioral comparison methodology for MBSE to MBSA consistency. Instead, we study the possibility of structurally synchronizing two artifacts representing behavior.

To operate this synchronization, we use a synchronization methodology similar to the SmartSync methodology, along with two mappings of concepts from state machines to the S2ML language.

3.2.1 MBSE and MBSA State Machines synchronization mappings

To structurally compare state machines, we suggest a methodology based on the model synchronization SmartSync platform. Thus we first abstract our models to S2ML models, then we compare these models. Finally, based on the comparison results, we decide the actions that have to be taken on the original state machines to ensure consistency.

In this section, we present two mappings of Statechart and AltaRica 3.0 state machines concepts to S2ML that we will then compare through case studys in section 3.2.2 and 3.2.3.

3.2.1.1 Primary Mapping

The first mapping of concepts is the basic vision of seeing a state machine as states linked through transitions. Therefore this mapping follows the correspondence presented in table 3.2. Note that in the case of statecharts, states are encapsulated in components but can also be encapsulated in other states.

SysML	S2ML	AltaRica 3.0
Component	Block	block
State	Block	State
Transition	Connection and port	Transition + event

Table 3.2: Primary mapping of concepts

To represent a transition, we create ports in both states that are the source and the target of the transition, and we then create a connection between those two ports encapsulated in the source state. For example, figure 3.1 represents a system `Syst`, with State `A` and State `B` linked by transition `"trans"`.

```
block Syst
  block A
    port trans;
  end
  block B
    port trans;
  end
  connection [A.trans, B.trans];
end
```

Figure 3.1: S2ML representation of a connection with the primary mapping

This conceptual mapping is basic. It does not convey all the properties that could be put inside either a SysML State Chart or an AltaRica 3.0. For this reason, we also suggest a more advanced mapping.

3.2.1.2 Advanced mapping

Because of the simplicity of the primary mapping, we have designed a more advanced mapping that conveys more properties of the state machines. This second mapping also modifies how we conceptually model states. This more advanced mapping is detailed in table 3.3.

Note that in this mapping, we represent states using ports. This is a better translation of the atomic concept of state, as ports are also atomic elements of the S2ML formalism, i.e., they cannot be decomposed into more model elements. Consequently, transitions are expressed by connections between the ports representing their source and target states. Thus the previous state `A` to state `B` `"trans"` transition is now written as illustrated in Fig. 3.2.

Because of this modification, although this new mapping is more complex, it also allows for more concise modeling of simple state machines. We can also easily represent depth in the state machines by representing composite states

SysML	S2ML	AltaRica 3.0
Component	Block	block
State	port	State
Transition	Connection	Transition
initial state	"init = true" attribute	initial state
final state	"final = true" attribute	—
sub-machine state	New block with inside states as ports	—
detailed historic	"dHist = true" attribute	—
succinct historic	"Hist = true" attribute	—
orthogonal state	two or more blocks	state machines in different components

Table 3.3: Advanced mapping of concepts

```
block Syst
  port A;
  port B;
  connection trans [A; B];
end
```

Figure 3.2: S2ML representation of a transition with the advanced mapping

as blocks. For example, the state machine in Fig. 3.3 can be modeled as shown in Fig. 3.4.

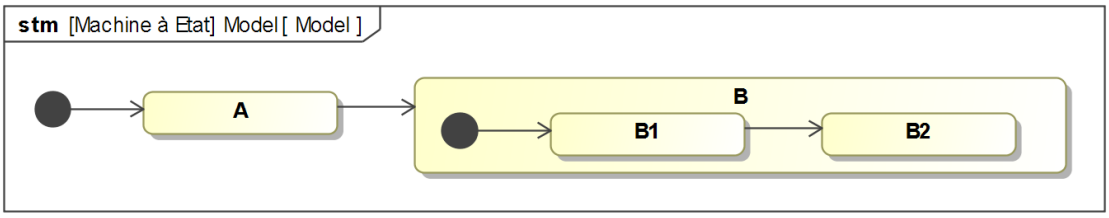


Figure 3.3: State machine with deep state

To model the transition entering state `B`, we write a connection between `A` and the initial state of the `B` sub-machine state `B1`. This also allows us to model cases where a transition would target a non-initial state of `B` as its target.

```
block Model
  port A (init = true);
  block B
    port B1 (init = true);
    port B2;
    connection [B1; B2];
  end
  connection trans [A; B.B1];
end
```

Figure 3.4: S2ML code of a state machine with a deep state using the advanced mapping

3.2.2 Gear Lights comparison

3.2.2.1 Primary mapping

To operate the comparison, we first translated the MBSE and MBSA state machines presented in section 2.2.3 to S2ML following the primary mapping from section 3.2.1.1. This translation results in the code shown in figure 3.5.

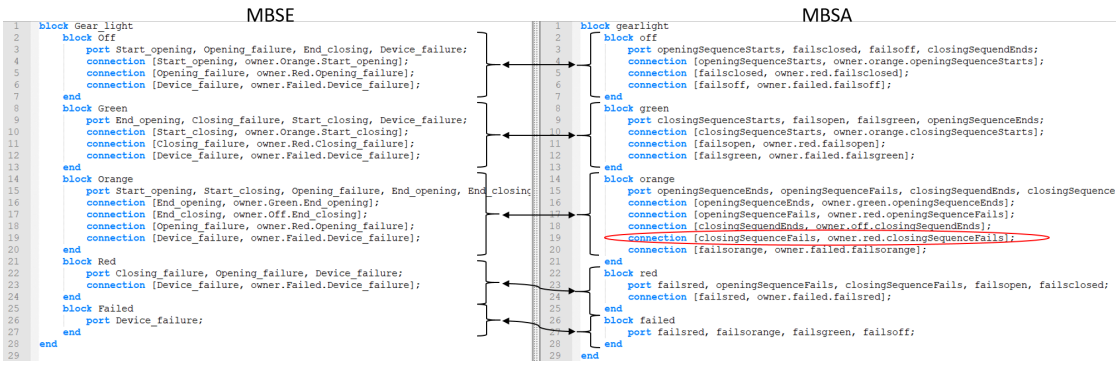


Figure 3.5: S2ML abstractions for the gear light state machines with primary mapping

We then compare these codes by producing a naming correspondence between the models. This correspondance is illustrated in table 3.4.

We then compare the states and transitions of the state machines. From this comparison, we observe some differences between our MBSE and MBSA models:

- The `orange` state in the AltaRica model has a transition `ClosingSequenceFails` that has no correspondent in the SysML model;
- The `Device_failure` transition in the SysML model is unique for all states while it is not in the AltaRica model;

MBSE	MBSA
Start_opening	openingSequenceStarts
End_opening	openingSequenceEnds
Start_closing	closingSequenceStarts
End_closing	closingSequendEnds
Closing_failure	failsopen
no correspondance	closingSequenceFails
Device_failure	failsorange failsred failsgreen failsoff
Opening_failure	openingSequenceFails failsopen

Table 3.4: Naming correspondence between MBSE and MBSA models for Gear lights behavior

- Naming differences: lower case or upper case first letter differences between namings in the models, no space between words in AltaRica model.

The comparison results provide us with knowledge of the differences between these state machines. Because these differences may not be inconsistencies, the system architect and safety analyst shall decide whether each of these differences is to be corrected or not. Therefore they must declare which actions are to be taken to correct them. In our case, we decide:

- A `Closing_failure` transition is added to the MBSE model;
- We consider the correspondence of multiple `Device_failure` transitions with different `FailsColor` transitions a formalism difference since the different naming in AltaRica corresponds to a same event which is the failure of the component. Therefore no further action is taken;
- Naming differences are deemed acceptable in the system as the names convey similar meanings.

3.2.2.2 Advanced mapping

In the same way, as we did before, we first translate the models to S2ML, now using the mapping from table 3.3. As a result, we obtain the code displayed in figure 3.6. The naming correspondence we deduce is the same that we have shown in 3.4. We can also observe the same differences as with the primary mapping. We then apply the same corrections to the original models.

MBSE

```
block gearLight
    port Off (init = True);
    port Orange;
    port Green;
    port Red;
    port failed;
    connection Start_opening [Off; Orange];
    connection End_opening [Orange; Green];
    connection Opening_failure [Orange; Red];
    connection Start_closing [Green; Orange];
    connection End_closing [Orange; Off];
    connection Fails_open [Green; Red];
    connection Fails_Closed [Off; Red];
    connection Device_failure [Off; Failed];
    connection Device_failure1 [Orange; Failed];
    connection Device_failure2 [Green; Failed];
    connection Device_failure3 [Red; Failed];
end
```

MBSA

```
block gearLight
    port off (init = True);
    port orange;
    port green;
    port red;
    port failed;
    connection openingSequenceStarts [Off; orange];
    connection openingSequenceEnds [orange; green];
    connection openingSequenceFails [orange; red];
    connection closingSequenceStarts [green; orange];
    connection closingSequendEnds: s == orange; off];
    connection failsopen [green; red];
    connection failsclosed [off; red];
    connection failsred [red; failed];
    connection failsorange [orange; failed];
    connection failsgreen [green; failed];
    connection failsoff [off; failed];
    connection closingSequenceFails: s == orange; red];
end
```

Figure 3.6: S2ML abstractions for the gear light state machines with advanced mapping

3.2.3 Redundant system comparison

3.2.3.1 Primary mapping

MBSE

```
block CRSys
    block IDLE
        port MainOn;
        port MainBackToIDLE;
        connection [MainOn; owner.MainWorking.MainOn];
    end
    block MainWorking
        port MainFails;
        port MainBackToIDLE;
        port MainOn;
        connection [MainBackToIDLE; owner.IDLE.MainBackToIDLE];
        connection [MainFails; owner.MainFailed.MainFails];
    end
    block MainFailed
        port SpareOn;
        port MainFails;
        connection [SpareOn; owner.Spare.SpareOn];
    end
    block Spare
        port SpareBackToIDLE;
        port SpareFails;
        port SpareOn;
        connection [SpareBackToIDLE; owner.IDLE.SpareBackToIDLE];
        connection [SpareFails; owner.SpareFailed.SpareFails];
    end
    block SpareFailed
        port SpareFails;
    end
end
```

MBSA

```
block System
    block Umain
        block WORKING
            port evFailure;
            port evStart;
            connection [evFailure; owner.FAILED.evFailure];
        end
        block FAILED
            port evFailure;
            port evFailedToStart;
        end
        block STANDBY
            port evStart;
            port evFailedToStart;
            connection [evFailedToStart; owner.FAILED.evFailedToStart];
            connection [evStart; owner.WORKING.evStart];
        end
    end
    block Uspare
        block WORKING
            port evFailure;
            port evStart;
            connection [evFailure; owner.FAILED.evFailure];
        end
        block FAILED
            port evFailure;
            port evFailedToStart;
        end
        block STANDBY
            port evStart;
            port evFailedToStart;
            connection [evFailedToStart; owner.FAILED.evFailedToStart];
            connection [evStart; owner.WORKING.evStart];
        end
    end
end
```

Figure 3.7: S2ML abstractions for the cold redundancy with primary mapping

The primary mapping for the redundant system use case results on the codes found in figure 3.7.

From this code, we can observe a fundamental difference between the models we compare. The MBSE model contains a system-level block called `CRSys` that corresponds to the MBSA model `System` block. However, in the MBSE model, the blocks representing states are directly contained in the `CRSys` block, whereas in the MBSA model, the states are contained in sub-blocks of the `System` block.

This is due to the difference in the level of abstraction described in subsection 2.2.3.2. As we describe the behavior at the system level in MBSE and the component level in MBSA, there is no direct correspondence between the elements described in both models. Since our translation methodology does not carry more information, it does not enable us to perform a comparison for this kind of state machine (including orthogonal states).

3.2.3.2 Advanced mapping

MBSE

```
block ColdRedundancy
    port IDLE (init = True);
    port MainWorking;
    port MainFailed;
    port Spare;
    port SpareFailed;
    connection MainOn [IDLE, MainWorking];
    connection MainBackToIDLE [MainWorking, IDLE];
    connection MainFails [MainWorking, MainFailed];
    connection SpareOn [MainFailed, Spare];
    connection SpareBackToIDLE [Spare, IDLE];
    connection SpareFails [Spare, SpareFailed];
end
```

MBSA

```
block System
    block Umain
        port WORKING (init = True);
        port FAILED;
        port STANDBY;
        connection evFailure [WORKING, FAILED];
        connection evStart [STANDBY, WORKING] (guard = "vfDemanded");
        connection evFailedToStart [STANDBY, FAILED] (guard = "vfDemanded");
    end
    block Uspare
        port WORKING;
        port FAILED;
        port STANDBY (init = True);
        connection evFailure [WORKING, FAILED];
        connection evStart [STANDBY, WORKING](guard = "vfDemanded");
        connection evFailedToStart [STANDBY, FAILED] (guard = "vfDemanded");
    end
end
```

Figure 3.8: S2ML abstractions for the cold redundancy system state machines with advanced mapping

In the same way, the translation using advanced mapping (figure 3.8) does not allow the direct creation of a correspondence table. However, this methodology allows comparison because we have a more refined representation of the state machine.

Through computation of the Cartesian product of both components' state machines, we obtain a new state machine representing the system's global behavior. This is done as follows:

- The initial state is a couple containing the initial states of both state machines, here (`Umain.WORKING`, `Uspare.STANDBY`);
- The transitions are those from either state machines with one of the two states as their source, here `Umain.evFailure` and `Uspare.evStart`;
- The target states of these transitions are couples composed of the original target state of the transition and the other unchanged state. E.g. for the `Umain.evFailure` transition the target state is (`Umain.FAILED`, `Uspare.STANBY`);
- We then iterate on the new states until we reach only existing states.

Considering the guard conditions to eliminate unreachable states, we finally obtain the state machine depicted in figure 3.9. Note that this comparison

is no longer structural, as we had to compute the cartesian product, which requires understanding of the execution of the state machine.

In this code, we name the state by combining the initials of the two states they contain, therefore (`Umain.WORKING`, `Uspare.STANBY`) is called `WS`. Transitions were also assigned arbitrary names to avoid redundancy. After this computational phase, we can establish a correspondence table between the MBSE and MBSA state machines, which is given in table 3.5.

On this basis, we can now observe a few differences between the MBSE and MBSA models:

- The MBSE `IDLE` state and transitions coming out from and back to this state do not exist in the MBSA model;
- The MBSA `FM2` transition that allows going directly from the `FS` state – where the main unit is failed, and the spare unit is in standby – to the `FF` state – where both units are failed – does not exist in MBSE.

```
block System
  port WS (init = True);
  port FS;
  port FW;
  port FF;
  connection FM [WS, FS];
  connection SS [FS, FW];
  connection FM2 [FS, FF];
  connection FS2 [FW, FF];
end
```

Figure 3.9: Product of the state machines for redundant system's components

We can correct these differences by :

- Initialising the main unit to `STANDBY` in the MBSA model and adding a transition going from `WORKING` to `STANDBY` in both units;
- Adding transition `Spare failed to start` to the MBSE model with source `MainFailed` and target `SpareFailed`.

3.2.4 Discussion

From these use cases, we can conclude that, although the first mapping is relevant for simple state machines, it does not provide enough information to compare more complex state machines, such as those with orthogonality. This lack is addressed by the second mapping, which is more advanced.

MBSE	**MBSA**
IDLE	
MainWorking	WS
MainFailed	FS
Spare	FW
SpareFailed	FF
MainOn	
MainBackToIDLE	
MainFails	FM
SpareOn	SS
SpareBackToIDLE	
SpareFails	FS2
	FM2

Table 3.5: Naming correspondence between MBSE and MBSA models for redundant system behavior

Nevertheless, this proves the limitation of structural comparison towards state machine consistency, as we cannot deal with different abstraction levels without considering the execution. In 3.2.3.2, we computed the product of the state machines attributed to the main and spare components. However, we also considered the guards to eliminate unreachable states that would pollute the comparison. Eliminating these unreachable states requires us to be aware of how the state machine is executed and can be more complicated than shown in our case study. Therefore we believe that structural comparison is insufficient to assert consistency between MBSE and MBSA state machines. Further work on comparison linked to their execution would be needed to supplement the structural aspect.

Finally, the reader should note that we only propose a way to synchronize MBSE and MBSA state machines. This is not a methodology for MBSE and MBSA behavior synchronization. On the one hand, MBSE formalism does provide tools other than state machines to model behavioral artifacts, such as use case diagrams. On the other hand, this methodology strictly compares state machines. However, the MBSA will usually model behavior with states that indicates whether the system is functionnal or not and complete the behavioral model with other indicators that denote the functional behavior. This also needs to be considered to achieve MBSE and MBSA behavioral synchronization.

CHAPTER 4

THE S2ML+CAT FRAMEWORK

In this chapter, we introduce the main contribution of the thesis: the S2ML+Cat mathematical framework. This framework allows to represent S2ML models, and by extension, structural models, through categories. In this framework, we define the concept of binary consistency relations, which are binary relations over the set of S2ML+Cat categories. The binary consistency relations respect some axioms, which makes it so that they imply the existence of some common structure between the related models. This implies that we can identify differences between the models thanks to these relations.

In the first section of this chapter, we present a simplified overview of the concept for this mathematical framework and the idea behind it. In the second section, we describe the mathematical concepts we use to represent S2ML models with categories, and the S2ML+Cat category, which is the universe of S2ML models. In the third section, we present some useful theorems and the definition of S2ML models equivalence. The theorems are either used for the definition of equivalence or the definition of a Consistency relation. The fourth section describes the concept of a binary consistency relation between S2ML models. The fifth section discusses the use of the S2ML+Cat category to simplify the comparison of new versions of previously synchronized models.

4.1 The S2ML+Cat idea

In the first section of this chapter we propose a general overview of our mathematical framework that will be detailed in the following sections. To do so, we present a simplified version of the framework, corresponding to the original idea we had during its conceptualization. However, this representation is incomplete, it will be further defined in Section 4.2 by giving formal definitions of all the mathematical objects.

The framework relies on a categorical representation of the S2ML models. This representation is not meant to replace the original algebraic definition of S2ML models that was presented in Section 1.3.6. It rather proposes an alternative, categorical point of view.

4.1.1 Simplified S2ML+Cat models

We propose to use categories to represent structural models such as those written in the System Structure Modeling Language (S2ML, see Section 1.3.6).

These models are structured around three concepts:

- Blocks: Blocks are containers that define the arborescence of the system.
- Ports: Ports are atomic elements, such as variables or states.
- Connections: Connections allow for establishing constraints over one or multiple ports. A simple example of a connection between two ports A and B of type integer would be "`A = B`".

We identify two types of relations between these model elements:

- Belonging relation: A block, port, or connection is contained in a block
- Reference relation: A port participates in a connection.

This allows us to define a metamodel category for structural models, illustrated as a graph in Fig. 4.1.

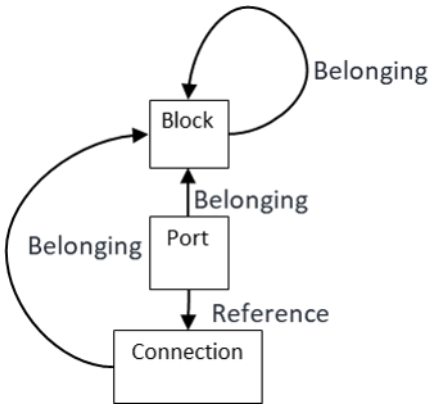


Figure 4.1: Simple representation of the metamodel for an S2ML model

This representation could be considered as a very simple category: The blocks, ports, and connections are the objects of the category, while the reference and belonging relations are the morphisms. However, in a category, morphisms can be composed. This means that if a port is referenced in a connection that belongs to a block, then there is a morphism in the category from

the port to the block. If the port does not belong to that specific block, the resulting morphism is neither a belonging nor a reference.

Although this situation can seem weird and unlikely, it is possible to define such a connection, as illustrated in Fig. 4.2.

```
block Syst
  block A
    block A1
        port p1;
    end
    block A2
        port p2;
    end
  end
  block B
    connection [main.A.A1.p1, main.A.A2.p2];
  end
end
```

Figure 4.2: S2ML representation of a connection situated in a block different from its ports

This means that the representation described here is not a category. Because we need the composition properties of category theory and because it allows for a more precise description of the models, we complete this definition in Section 4.2.

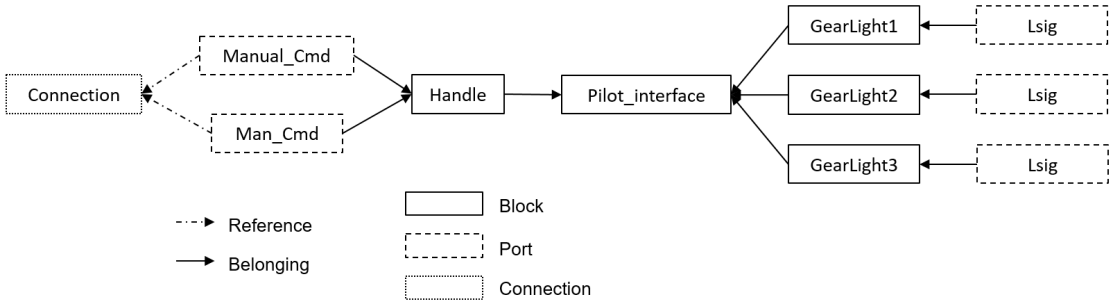


Figure 4.3: The pilot interface S2ML model translated from MBSE, represented using the simplified representation

Even though it is not a category, this simplification illustrates the mathematical framework's intent of using blocks, ports, and connections as objects and belongings and references as morphisms. We also tried to define all the concepts in a graph theory vision that would match this representation, which allowed us to implement similar theorems and proofs. However, the composition allowed by category theory simplifies the representation of levels of system abstraction, which we believe makes it more appropriate.

In the case of the landing gear, if we refer to the S2ML code for the MBSE representation of the `pilot_interface` part, presented in Fig 2.14 (In Section 2.3.1). We add the connection `"connection [Manual_Cmd, Man_Cmd]"` to the block `Handle`, for the sake of showing a connection. This represents a dependency between the input and output of the handle. We obtain the mathematical structure represented in Fig. 4.3.

MBSE	**MBSA**
Pilot_interface	pilotInterface
Pilot_interface.Handle	pilotInterface.udHandle
Pilot_interface.Handle.Manual_Cmd	pilotInterface.udHandle.position
Pilot_interface.Handle.Man_Cmd	pilotInterface.udHandle.output
Pilot_interface.GearLight1	pilotInterface.frontGearLight
Pilot_interface.GearLight1.Lsig	pilotInterface.frontGearLight.input
	pilotInterface.frontGearLight.output
Pilot_interface.GearLight2	pilotInterface.rightGearLight
Pilot_interface.GearLight2.Lsig	pilotInterface.rightGearLight.input
	pilotInterface.rightGearLight.output
Pilot_interface.GearLight3	pilotInterface.leftGearLight
Pilot_interface.GearLight3.Lsig	pilotInterface.leftGearLight.input
	pilotInterface.leftGearLight.output
	pilotInterface.analogicalSwitch
	pilotInterface.analogicalSwitch.recentChange
	pilotInterface.analogicalSwitch.output

Table 4.1: Correspondance table for the dictionary relation between the Pilot interface parts of the MBSE and MBSA models of the landing gear

4.1.2 Relation between the models

We mainly intend to compare models' structures, which is why the mathematical representation of the models heavily relies on this structure.

Our strategy to compare the structure is to identify a common skeleton between both models. Once we have identified this common skeleton, we can say that the model elements that are not in it constitute differences between the models that shall be reviewed.

Various criteria can be used to identify this common skeleton, which can have more or fewer constraints. For example, we can consider a dictionary relationship that would associate the model elements two by two between the models and not consider any connections or belonging relations. We create comparison models by eliminating everything that is not useful to the comparison. In the case of the dictionnary relation, we only consider the model elements' names, and allign them two by two. This results in the categories having

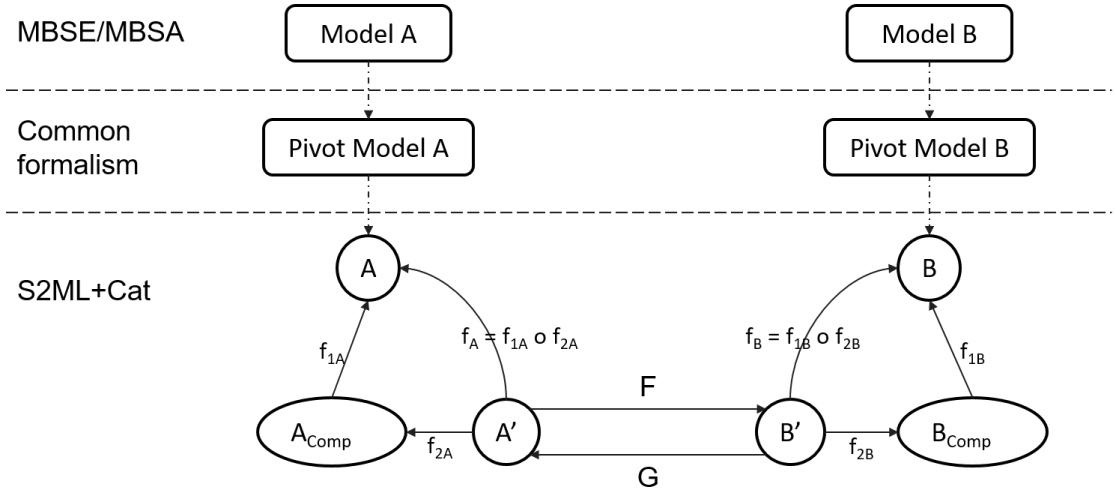


Figure 4.4: Structure for binary consistency relations

no morphisms since only the blocks and ports names will be considered, but neither the connections nor the belonging morphisms. Thus we obtain lists of blocks and ports. We can then create a comparison table associating the elements two by two.

If we apply this reasoning to the pilot interface, we obtain table 4.1.

This table can be interpreted as the representation of two applications between the models. If the representations we showed for S2ML models in Section 4.1.1 were really categories, those applications would be functors.

We will provide in the rest of the chapter definitions that allows to create a consistent mathematical framework based on these concepts. For now we only manipulated them for the sake of explication, but they are not yet complete.

Let us call A_{Comp} (resp B_{Comp}) the comparison "categories" for the MBSE (resp MBSA) models. We call A and B the "categories" for the original MBSE and MBSA models. We call A' and B' the "categories" constituted of only the elements associated with an element of the other model. We can define functors $F : A' \longrightarrow B'$ and $G : B' \longrightarrow A'$, where F associates each element of the first column of table 4.1 to the element of the same row in the second column, and conversely for G. We can define functors that inject A' in A_{Comp}, and A_{Comp} in A (idem for B', B_{Comp} and B). These functors are fully described in Section 4.2.4, in definition 4.7.

The objects that are not associated with an element of the other model constitute the differences we detect between the models.

This mathematical construction of creating comparison models (A_{Comp} and B_{Comp}) and compared models (A' and B') constitutes the consistency relation that we establish between the two models. This relation is illustrated in Fig. 4.4

If we consider the SmartSync comparison, which compares both arbores-

cence and elements of the model, excluding the connections, we obtain A_{Comp} and B_{Comp} categories depicted in Fig. 4.5

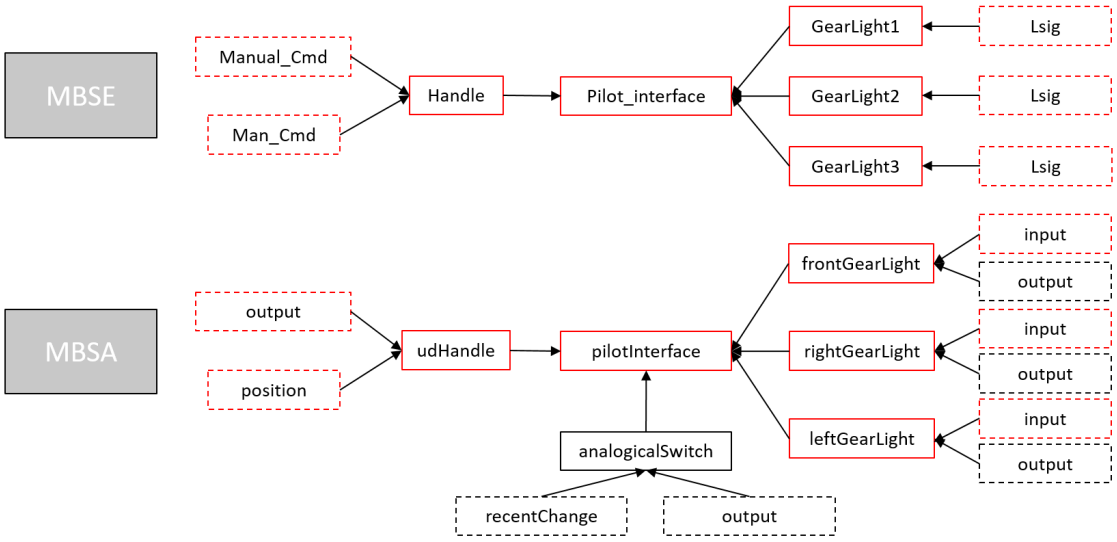


Figure 4.5: The pilot interface S2ML model translated from MBSE and MBSA, represented using the simplified representation, with their common skeleton in red

The red part of the categories corresponds to the A' and B' categories, as they are the common skeleton between both models.

In the rest of this chapter, we will define precisely how we represent S2ML models through categories and use this mathematical construction to define the concept of binary consistency relation between S2ML models.

4.2 Mathematical representation of a structural model

In this section, we assume that a model that carries structural information can be described with the following elements:

- Blocks
- Ports
- Connections

This is the case with the comparison formalisms used in the synchronization methodologies from subsection 1.3.4, although they are named differently in the Consistency links methodology. In [18], the S2ML models are formalized

with a quintuple composed of sets of ports and blocks, a multiset of connections, a composition relation, and a unique block with no parents regarding the relation.

Even though this description is rigorous, we propose a new representation inspired by category theory which allows for the definition of functors between the models. To give a categorical representation of S2ML models, we need to define four concepts corresponding to S2ML's ports, connections, blocks and models: Catports, Catconnections, Catblocks, and then Catmodels.

4.2.1 Catports, Catconnections and Catblocks

Definitions:

In this section, when we use set theory vocabulary, such as "a set", "a singleton", etc., we mean to use these notions with their naive set theory meaning. The ports are atomic elements of the model. They can be used to express variables in the model, states of a component, or other properties. We use symbols to represent this property; the symbol could be understood as the port's name.

Definition 4.1. ***"Catport"***
We consider a countable set of symbols P.

A Catport is a category CP where:

- *Ob(CP) is a singleton containing a symbol $p \in P$.*
- *The only morphism is the identity on p.*

Note that in this definition, the notation CP refers to "Catport" and not to the set P.

This definition means that to any $p \in P$, we can associate a Catport $Cat(p)$ with $Ob(Cat(p)) = p$ and the unique morphism $Id_p \in Hom(p, p)$.

In S2ML models, ports are interlinked through connections. We define connections as sets of ports.

Definition 4.2. ***"Catconnection"***
A Catconnection is a category CC where:

- *Ob(CC) is a non-empty, finite set of Catports.*
- *The only morphisms in CC are the identities.*

This means that if we have two Catports, $p1$ and $p2$, a connection between these two ports is the set $\{p1, p2\}$ along with the identities over these Catports.

Catports and Catconnections have no other morphisms than the identities. Such categories are called discrete categories. They can be seen as their object set although for the sake of homogeneity we make them categories.

We can now define the blocks with ports and connections, which are containers in the model. A block is a container that holds other blocks, ports, and connections.

Definition 4.3. ***"Catblock"***

A Catblock CB is defined as:

1. $Ob(CB)$ *is a finite set of Catblocks, Catports, and Catconnections*

2. *For each Catconnection* $C \in Ob(CB)$*, if a Catport* $P1 \in Ob(CB)$ *is such as* $P1 \in Ob(C)$*, then there exists two morphisms* $r_{P1,C} : P1 \longrightarrow C$ *and* $r'_{P1,C} : C \longrightarrow P1$ *in B, such as the component on objects of* $r_{P1,C}$ *maps* $p \in Ob(P1)$ *to* $P1 \in Ob(C)$ *and any morphism to the corresponding identity, and the component on objects of* $r'_{P1,C}$ *maps every object of C to* $p \in Ob(P1)$ *and every morphism to* Id_p*.*

3. *For each Catblock* $B_1 \in Ob(CB)$*:*

   - For each Catconnection $C \in Ob(B_1)$*, we have* $C \in Ob(CB)$ *and there is a morphism* $\alpha_{C,B_1} : C \longrightarrow B_1$ *in B that maps each object of C to* $C \in Ob(B_1)$ *and each morphism to the corresponding identity*

   - For each Catport $P \in Ob(B_1)$*, we have* $P \in Ob(CB)$ *and there is a morphism* $\alpha_{P,B_1} : P \longrightarrow B_1$ *in B that maps* $p \in Ob(P)$ *to* $P \in Ob(B_1)$ *and the identity on p to itself in* B_1

   - For each Catblock $B_2 \in Ob(B_1)$*, we have* $B_2 \in Ob(CB)$ *and there is a morphism* $\alpha_{B_2,B_1} : B_2 \longrightarrow B_1$ *that maps every object of* B_2 *to itself in* B_1 *and does the same with morphisms*

   For each X in $Ob(B_1)$*, there exist a unique Catblock* $B' \in CB$ *(possibly* B_1*), such as there is no Catblock* $B'' \in B'$ *such as* $X \in B''$*, we also have either* $B' = B_1$ *or* $B' \in Ob(B_1)$

4. *There are no other morphisms in the Catblock than those described here and the morphisms that derive from category theory axioms, i.e., identities and compositions.*

5. *for* $x, y \in Ob(CB)$*, we note* $Hom(x, y)$ *the set of morphisms from x to y in CB.*

Because Catblocks can contain Catblocks, this definition can seem like a Catblock may not be finite. However, the set of objects of the Catblock is finite, and if it contains a Catblock, then it contains everything that is in the Catblock. This means that we will always at some point find Catblocks that only contain Catports and Catconnections, and therefore any Catblock is finite and the definition allows to build Catblocks by induction.

Proposition 4.1. *A Catblock is a category.*

Proof. To prove that a Catblock is a category, as it includes the identities, we need to show that we can compose the morphisms, and that this composition is associative.

Let C be a Catblock, let $x, y, z \in Ob(C)$, we assume $x \neq y$ and $y \neq z$, as compositions with the identities are trivial.

Let $f \in Hom(x,y)$, $g \in Hom(y,z)$, with f and g morphisms defined in the points 2 and 3 of the definition.

If x, y, z are Catports or Catconnections, then we are in the case of the applications described in the second point of the definition. This means that f and g are functors, as Catports and Catconnections are categories; therefore, we can compose them.

Morphisms that have Catblocks as their source always have Catblocks as their target, therefore if y is a Catblock then z is a Catblock, and if x is a Catblock, then y and z are Catblocks.

We assume that z is a Catblock.

If y is a Catconnection, then x is a Catblock.

Therefore we have:

$Ob(x)$ is a singleton and the only morphism of x is the identity over $Ob(x)$'s only element, let us call it a. $f_{ob}(a) = x$ where $x \in ob(y)$, and $f_{a,a}(Id_a) = Id_x$.

As specified in point 3 of the definition, $g_{ob}(x) = y$ and $g_{x,x}(Id_x) = Id_y$.

Therefore $g_{ob}(f_{ob}(a)) = y$ and $g_{x,x}(f_{a,a}(Id_a)) = Id_y$.

We have fully constructed $g \circ f$ therefore it exists.

If y is a Catport then x is a Catconnection.

This means that for any $a \in Ob(x)$, $f_{ob}(a) = p$ where p is the single element of $Ob(y)$, and $f_{a,a}(Id_a) = Id_p$.

As specified in point 3 of the definition, $g_{ob}(p) = y$, with $y \in Ob(z)$ and $g_{p,p}(Id_p) = Id_y$.

Therefore, for any $a \in Ob(x)$, $g_{ob}(f_{ob}(a)) = y$ and $g_{p,p}(f_{a,a}(Id_a)) = Id_y$.

We have fully constructed $g \circ f$ therefore it exists.

The last case is if y is a block.

Then for any $a, b \in Ob(x)$, there are $a', b' \in Ob(y)$ such as $f_{ob}(a) = a'$, $f_{ob}(b) = b'$, and for any $h \in Hom(a,b)$ there is a $h' \in Hom(a', b')$ such as $f_{a,b}(h) = h'$.

If x is a Catport or a Catblock this is the identity, if x is a Catconnection this maps everything to x and Id_x.

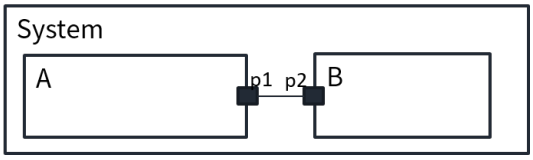


Figure 4.6: Example of a system model

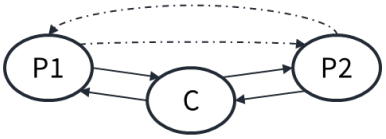


Figure 4.7: Structure of a connection with two ports in a block/model

As specified in point 3 of the definition, $g_{ob}(a') = a'$, $g_{ob}(b') = b'$ and $g_{a,b}(h') = h'$.

Therefore $g_{ob}(f_{ob}(a)) = a'$, $g_{ob}(fob(b)) = b'$ and $g_{a',b'}(f_{a,b}(h)) = h'$.

We have fully constructed $g \circ f$ therefore it exists.

Thus we have compositions in Catblocks.

Let $a, b, c, d \in C$, $f \in Hom(a,b)$, $g \in Hom(b,c)$, $h \in Hom(c,d)$.

We obtain associativity because f_{ob}, g_{ob}, and h_{ob} are set applications, therefore their composition is associative, and, for $x, y \in Ob(a)$, $f_{x,y}$,$g_{f_{ob}(x),f_{ob}(y)}$, and $h_{g_{ob}(f_{ob}(x)),g_{ob}(f_{ob}(y))}$ set applications, therefore their composition is associative.

Thus the composition of morphisms in Catblocks is associative since all their components are associative.

Therefore the Catblocks are categories.

□

This definition carries the properties of a block to the world of categories. Thus each point of the definition can be explained straightforwardly.

Considering a Catconnection C and a Catport $P1$ that the Catconnection refers to. If the Catport is contained in the same Catblock B as the connection, we define two morphisms $r_{P1,C} : P1 \longrightarrow C$ and $r'_{P1,C} : C \longrightarrow P1$ in B between the Catconnection and the Catport, and the Catport and the Catconnection respectively. These morphisms represent the relation between the Catport and the Catconnection. The Catport is part of the Catconnection, and the Catconnection refers to it. The typical structure of a Catconnection in a Catblock is depicted in Fig 4.7; it corresponds to the connection from Fig. 4.6. We do not assume that all ports of a connection are contained in the parent block of the connection, but we will eventually find an ancestor block containing all the connection's ports in a model. However, connections are usually going from one block to another. Thus there is no reason for all Catports to be contained in any Catblock that contains the Catconnection.

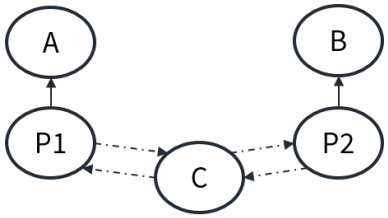


Figure 4.8: Diagram of the system block from Fig. 4.6

In Fig. 4.7, the plain arrows represent the $r_{P1,C}, r'_{P1,C}, r_{P2,C}$ and $r'_{P2,C}$ morphisms, and the dotted arrows represent their compositions. We have morphisms from the Catports to the Catconnection and from the Catconnection to the Catports. Therefore, we also have morphisms between Catports participating in the same Catconnection because of composition. In the same way, let us assume we have three Catports, P1, P2, and P3, and two Catconnections, C1 between P1 and P2 and C2 between P2 and P3; because of composition, we will have morphisms between P1 and P3, indicating an indirect Catconnection between these Catports.

The third point of the definition does translate the belonging relation in S2ML models. We define a morphism between a Catblock, a Catport, or a Catconnection and a Catblock that contains it. In the case of Catblocks, this morphism is the identity; it maps every object of the Catblock to its parent or ancestor. This means that everything in the Catblock is also contained in its ancestors. For Catconnections, this relation will absorb the content of the Catconnection to itself since the Catconnection conceptually only refers to the Catports rather than being a container that they are a part of. This also helps to disambiguate the difference between a Catport being in a Catconnection that belongs to a different Catblock and a direct belonging morphism from a Catport to a Catblock. The last property of that point expresses that a model element has a unique parent, either the Catblock itself or a Catblock within it. All other Catblocks that contain it are its ancestors and will contain this parent.

An example of the category depicting the system Catblock from Fig. 4.6 is depicted in Fig. 4.8. This category indeed contains everything that is contained in the system Catblock. In this case, this includes the A and B Catblocks, the P1 and P2 Catports, and the C Catconnection.

An example of a block with the Pilot Interface:

If we detail the category for the MBSE block pilot interface with this mathematical representation, with the added connection between the handle's ports, we obtain a graph similar to the one from Fig 4.3. The main difference is the absence of a "Pilot_Interface" block. In this representation, a Catblock does not contain itself. The second difference is in the Catconnection, where we have morphisms both from the Catports to the Catconnection and from the Catconnection to the Catports. This category is illustrated in Fig. 4.9.

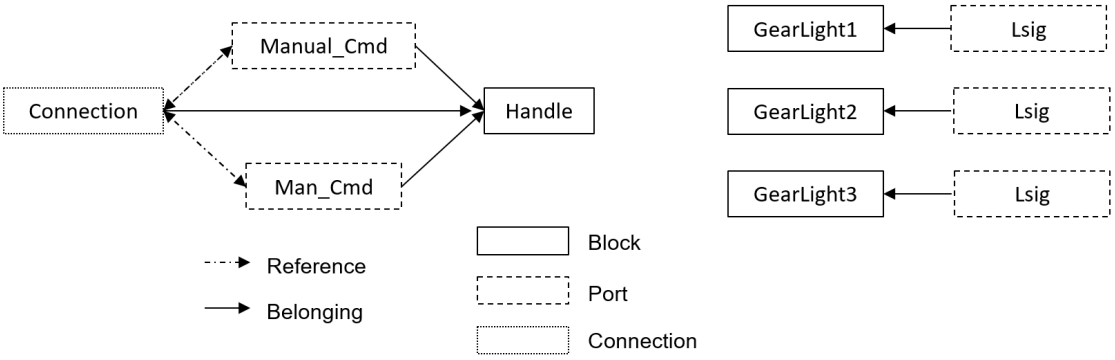


Figure 4.9: The Pilot interface S2ML block translated from MBSE, represented as a S2ML model category

Let us call this category PI_{Cat}; we have:

$$\begin{aligned} Ob(PI_{Cat}) = \{&Manual\_Cmd,\\ &Man\_Cmd,\\ &Connection,\\ &Handle,\\ &GearLight1,\\ &GearLight2,\\ &GearLight3,\\ &GearLight1.Lsig,\\ &GearLight2.Lsig,\\ &GearLight3.Lsig\} \end{aligned}$$

Where $GearLight1$, $GearLight2$ and $GearLight3$ are categories with $Ob(GearLightk) = \{\{Lsig\}\}$ and no morphisms other than the identities.

$Handle$ is a category with:

$$\begin{aligned} Ob(Handle) = \{&Manual\_Cmd,\\ &Man\_Cmd,\\ &Connection\} \end{aligned}$$

and morphisms

$$r_{Manual_Cmd,C}: \quad Manual\_Cmd \longrightarrow Connection\}$$

$$r_{C,Manual_Cmd}: \quad Connection \longrightarrow Manual\_Cmd$$

$$r_{Man\_Cmd,C}: \quad Man\_Cmd \longrightarrow Connection$$

$$r_{C,Man\_Cmd}: \quad Connection \longrightarrow Man\_Cmd$$

The $Handle$ category is the categorical representation of the `Handle` block that contains two ports, `Manual_Cmd` and `Man_Cmd`, represented by Catports containing symbols. It also contains a connection between both ports, repres-

ented by the Catconnection containing both Catports. The Catports and the Catconnection are linked by morphisms as described in the definition of Catblock.

PI_{Cat} contains the same morphisms as Handle, and morphisms representing the fact that blocks, ports, and connections belong to blocks. These morphisms are the identity over a Catport or a Catblock, and a constant morphism over the Catconnection, towards the Catconnection itself. Of course, any composition of these morphisms is also a morphism of PI_{Cat}. These composed morphisms are not represented in Fig. 4.9, as they would be redundant and make the graph difficult to read.

4.2.2 Models

Definition:
We have defined all elements of the models: Catblocks, Catports, and Catconnections; we can now use these concepts to define the model itself. The model contains the Catblocks, Catports, and Catconnections and is structured as its root (main) Catblock, with the difference that it also contains the root Catblock and morphisms between elements of the root Catblock and the root Catblock.

Definition 4.4. ***"Catmodel"***
A Catmodel is a category M such as:

- $Ob(M)$ *is a finite set of Catblocks, Catports, and Catconnections*
- *there exist a Catblock R called the root of M such as $Ob(R) = Ob(M) \setminus \{R\}$, and for each $x, y \in Ob(R)$, $Hom_M(x, y) = Hom_R(x, y)$*
- *The morphisms described between each object and a Catblock in the definition of a Catblock also hold between the objects of M and the Catblock R*
- *There are no other morphisms in the model than those described here and the morphisms derived from category theory axioms, i.e., identities and compositions.*

Catmodels are indeed categories, as the morphisms and objects that compose them are the same as Catblocks. Therefore the demonstration for composition and associativity is the same.

The root object in the model does contain all other objects; there will always be a belonging morphism from an object to the root object.

We can draw diagrams over S2ML models like diagrams are drawn over any other categories. In order to make these diagrams easy to read and faithful to the system, we only show the direct belonging morphisms and morphisms between Catports and Catconnections. Any morphism that is a composition or identity still exists in the category, but we do not show them in the diagrams.

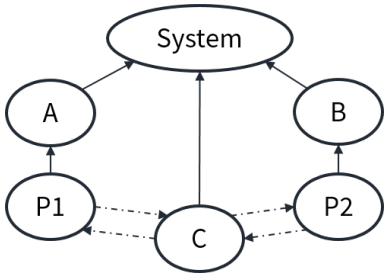


Figure 4.10: Diagram of the system model from Fig. 4.6

Example with the Pilot Interface:

In this example, we consider the Pilot Interface from the Landing Gear as a standalone model. The category associated to this model is represented in Fig. 4.11.

Compared to the Catblock category, the $Pilot\_Interface$ Catblock is added; it corresponds to the PI_{Cat} Catblock that we described in Fig. 4.9. Morphisms from the $Handle$, $GearLight1$, $GearLight2$, and $GearLight3$ Catblocks towards the $Pilot\_Interface$ Catblock and their compositions with the others are also added. Let us detail the morphism $\alpha_{Handle,Pilot\_Interface} : Handle \longrightarrow Pilot\_Interface$. Table 4.2 associate to each object of the Catblock it's image through $\alpha_{Handle,Pilot\_Interface}$ in the Catmodel.

Catblock	Catmodel
Handle	Handle
Manual_Cmd	Manual_Cmd
Man_Cmd	Man_Cmd
Connection	Connection

Table 4.2: Mapping for the belonging morphism between the Handle and the Pilot_Interface

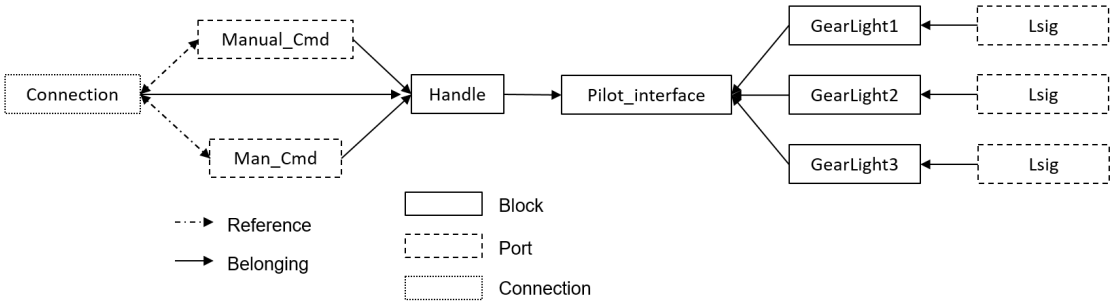


Figure 4.11: The Pilot interface S2ML model translated from MBSE, represented as an S2ML model category

The morphism mapping follows the same idea of mapping a morphism in the Catblock to itself in the model. This is the archetype of the belonging mappings in S2ML+Cat, a formal definition of such mapping and, therefore, how to recognize it is given in the next section.

4.2.3 Relations between the models: Belonging and reference morphisms

We have defined morphisms between the objects of the models. Although these morphisms can always be composed, they, in a way, are typed, as their properties can be used to explain the relationship they express between the objects. The belonging morphisms express that an object is contained in a Catblock.

Definition 4.5. ***"Belonging morphism", "Direct belonging morphism", "Ancestor", "Parent"***

Let M be a Catmodel, let $A, B \in Ob(M)$, with $A! = B$, let $f : A \longrightarrow B$ a morphism in M.

We call f a belonging morphism if and only if B is a Catblock, and:

- *A is a Catblock, and f is the identity over this Catblock*
- *A is a Catport, and f_{Ob} maps $p \in Ob(A)$ to $A \in Ob(B)$*
- *A is a Catconnection, and f_{Ob} maps any $x \in Ob(A)$ to $A \in Ob(B)$*

If f is a belonging morphism, we say that B is an ancestor of A.

We call f a direct belonging morphism if it is a belonging morphism and there is no Catblock $B' \in Ob(B)$ such as $A \in Ob(B')$.

If f is a direct belonging morphism, we say that B is the parent of A.

We call a morphism a belonging morphism if its target is a Catblock, and it either is similar to the identity when its source is a Catblock or a Catport, or it maps everything to the Catconnection itself if the source is a Catconnection.

The other significant kind of morphisms are reference morphisms. The reference morphism shows the relation between a Catport and the catonnection in which the Catport participates.

Definition 4.6. ***"Reference morphism"***

Let M be a Catmodel, let $P, C \in Ob(M)$ such as P is a Catport and C a Catconnection, let $f : P \longrightarrow C$ a morphism in M.

We call f a reference morphism if and only if f maps $p \in Ob(P)$ to $P \in Ob(C)$.

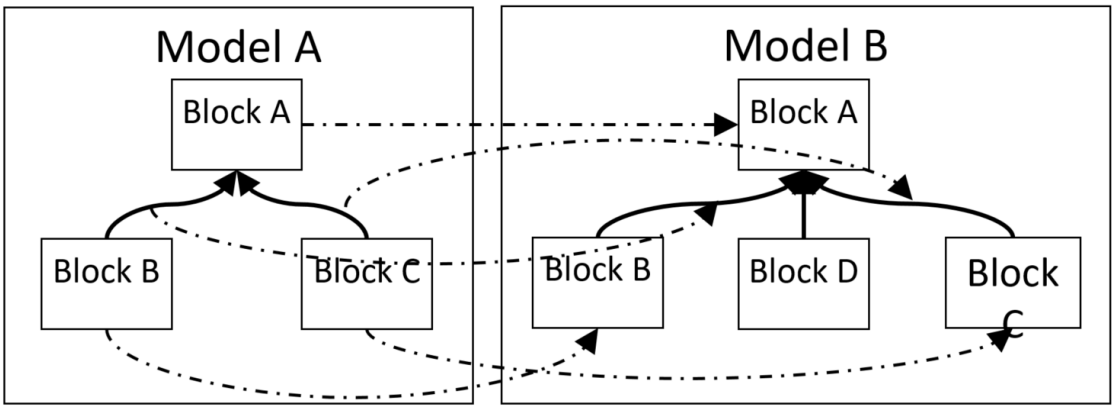


Figure 4.12: An injection between two models

4.2.4 A few notions necessary to introduce S2ML+Cat: Injections, orders and elementary blocks

We call injections a certain kind of functors between Catmodels. These functors represent the fact that a Catmodel is a part of another Catmodel.

Definition 4.7. ***"Injection"***
Let A, B be two Catmodels, let $F : A \longrightarrow B$ a functor.
We call F an injection if and only if

- *F is injective on objects and faithful*
- *F lets object's and morphism's properties unchanged, i.e.:*

  Catports, Catblocks, and Catconnections are respectively mapped to Catports, Catblocks, and Catconnections

  reference and belonging morphisms are respectively mapped to reference and belonging morphisms

A graphical illustration of injection between two Catmodels can be found in Figure 4.12

If we recall the Catblock `Pilot_Interface` represented in Fig. 4.9 and the Catmodel `Pilot_Interface` represented in Fig. 4.11, the Catblock is a part of the category, therefore it can be easily injected inside it. The identity functor is an injection from $Pilot\_Interface_{Block}$ to $Pilot\_Interface_{Model}$:

$$Id_{PI_{Block},PI_{Model}} : \quad Pilot\_Interface_{Block} \longrightarrow Pilot\_Interface_{Model}$$
$$X \longmapsto X$$

Interesting properties of the objects can also be defined, which we use in the demonstrations of theorems.

The order of an object is the number of depth levels in the system breakdown separating this object from the root object of the model. We also call the order of the model the number of depth levels of the system breakdown of the model. The notion of order is an indicator of the size of a Catmodel, as it determines the number of abstraction levels, it allows to make demonstrations by mathematical induction over Catmodels. It is also a way to mesure the distance between an object of the Catmodel, and the root object.

Definition 4.8. ***"Order" of a model object, "order" of a model***
Let M be a Catmodel, let $X \in Ob(M)$.
We call the order of X its distance to the root object, i.e., the number of direct belonging morphisms that must be composed to obtain a morphism $f : X \longrightarrow R$.
We call the order of M the upper bound of the orders of its objects.

Definition 4.9. ***"Elementary Catblock"***
We call an elementary Catblock any Catblock that contains no other Catblock.

4.2.5 S2ML+Cat

The Catmodels with the injections are a category that we call $S2ML+Cat$. $S2ML+Cat$ is the frame within which we will study the characteristics of the models.

Definition 4.10. ***"S2ML+Cat"***
We define $S2ML+Cat$ as being composed of:

- *$Ob(S2ML+Cat)$, the set of all Catmodels. We call them the objects of $S2ML+Cat$.*
- *for each $X, Y \in Ob(S2ML+Cat)$, $Hom(X,Y)$ is the set of the injections from X to Y.*

Proposition 4.2. *$S2ML+Cat$ is a category.*

Proof. To prove that $S2ML+Cat$ is a category, we need to prove the existence of the Identities and the composition and Associativity of morphisms.

- **Identity**:

  Let $X \in S2ML+Cat$

  The morphisms of S2ML+Cat are the injections between S2ML models; therefore if the identity Id_X is an injection, it exists in $Hom(X,X)$.

The identity is injective on objects and morphisms.

For any $x \in Ob(X)$, $Id_{X,Ob}(x) = x$, therefore if x is a Catport (resp. a Catblock or a Catconnection), its image is also a Catport (resp. a Catblock or a Catconnection).

For any $x, y \in Ob(X)$, for any $f \in Hom(x, y)$, $Id_{X,(x,y)}(f) = f$, therefore if f is a reference (resp. belonging), then its image is a reference (resp. belonging).

Thus Id_X lets object's and morphism's properties unchanged.

Id_X is an injection. Therefore $Id_X \in Hom_{S2ML+Cat}(X, X)$.

- **Composition**:

  Let $X, Y, Z \in S2ML + Cat$, let $F \in Hom_{S2ML+Cat}(X, Y)$ and $G \in Hom_{S2ML+Cat}(Y, Z)$.

  F and G are injections.

  $G_Ob \circ F_{Ob}$ is an injective application because it is a composition of injective applications and, thus, an injective application.

  For any $x, y \in Ob(X)$, $G_{F_{Ob}(x),F_{Ob}(y)} \circ F_{x,y}$ is an injective application because it is a composition of injective applications and, thus, an injective application.

  For any $x \in Ob(X)$, if x is a Catport (resp. a Catblock or a Catconnection), then $F_{Ob}(x)$ is a Catport (resp a Catblock or a Catconnection) because F is an injection; therefore because G is an injection $G_{Ob}(F_{Ob}(x))$ is also a Catport (resp. a Catblock or a Catconnection).

  $G \circ F$ lets object's properties unchanged.

  For any $x, y \in Ob(X)$, for any $f \in Hom(x, y)$, if f is a belonging (resp. a reference), then $F_{x,y}(f)$ is a belonging (resp. a reference) because F is an injection; therefore, because G is an injection $G_{F_{Ob}(x),F_{Ob}(y)}(F_{x,y}(f))$ is a belonging (resp. a reference).

  $G \circ F$ lets morphism's properties unchanged.

  Thus $G \circ F$ is an injection

  Therefore $H = G \circ F \in Hom_{S2ML+Cat}(X, Z)$.

- **Associativity**:

  Injections are functors between categories. The composition of functors is associative.

  Therefore the composition of injections is associative.

$S2ML+Cat$ presents identity morphisms, its morphisms can be composed, and their composition is associative. Therefore $S2ML+Cat$ is a category. □

We have defined a category that allows for the mathematical representation of S2ML models. Under the hypothesis that any structural model can be represented with blocks, ports, and connections, this mathematical representation readily applies to any such model. This is the basis for our mathematical framework.

4.3 Important properties in S2ML+Cat

4.3.1 The S2ML+Cat and S2ML equivalence

The definition we have given here allows a unique categorical representation of any S2ML model that was defined as a quintuplet. The quintuplet definition of S2ML models [18] is given in the state of the art of this thesis, in Section 1.3.6.

Theorem 4.1. *Any Catmodel of $Ob(S2ML+Cat)$ can be associated to a model of the set of the quintuples from [18] in a revertible way, up to the choice of symbols for blocks, and conversely.*

Proof. Let $S2ML$ be the set of all quintuplets from [18].

First, let us show that we can build a Catmodel from a S2ML model. To that purpose, we build F, an application that associate a Catmodel to any S2ML model.

Let F: $S2ML \rightarrow Ob(S2ML+Cat)$ with $X =< P,C,B,\alpha,r >$

$X \mapsto X_{cat}$

Where X_{cat} is such as:

- $Ob(X_{cat})$ contains:

  - for each $p \in P$, the Catport P' with $Ob(P') = \{p\}$

  - for each $c = \{p_1,...,p_n\} \in C$, the Catconnection c' such as $Ob(c') = \{P'_1,...,P'_n\}$ with, for any $k \in [|1,n|]$, P'_k the Catport with $Ob(P'_k) = \{p_k\}$.

  - for each block $b \in B$, the Catblock b' that contains all associated Catblocks, Catports, and Catconnections that are under b in the transitive closure of the α relation (i.e., any Catblock, Catport, Catconnection x such as there are b_1,..., b_n such as $(b,b_1),(b_1,b_2),...,(b_{n,x}) \in \alpha$ (possibly n=0 and $(b,x) \in \alpha$)).

- The morphisms in X are the morphisms described in definition 4.4.

The Catblock b' that we define is unique because it can be built by induction:

- first we build the elementary Catblocks, i.e. Catblocks that contain no Catblocks (the Catconnection and Catports being trivially unique, these Catblocks also are)
- then we build Catblocks of the next level of abstraction for the α relation
- we iterate on the second step until we have built all Catblocks that are contained in b'.

F allows us to convert any model from S2ML to a Catmodel in $Ob(S2ML + Cat)$.

Now let us show that we can build a S2ML model from a Catmodel. To that purpose, we build G, an application that associate a S2ML model to any Catmodel.

We assume that we have a mean to associate a symbol b' to a Catblock b.

Let G: $Ob(S2ML + Cat) \rightarrow S2ML$

$$X \mapsto X_{quint}$$

Where X_{quint} is the model $< P, C, B, \alpha, r >$ from S2ML, such as:

- $P = \{\ p \in Ob(P_{Cat}) \mid P_{Cat} \in Ob(X)$ is a Catport $\}$
- $B = \{\ b' \mid b'$ is a symbol associated to $b \in Ob(X)$ which is a Catblock $\}$
- $C = \{\ c' \mid c' = \{p \in Ob(P_{cat})$ for $P_{cat} \in c \mid c \in Ob(X)$ where c is a Catconnection$\}\}$
- $\alpha = \{\ (x', y') \in P \cup B \cup C \times B \mid Hom_X(x, y)$ contains a direct belonging morphism, where x, y are the pre images of x' and y' with the three first points $\}$
- r is the symbol associated to the root of X

Now we want to show that these two applications are the inverses of eachother, up to the choice of symbols for the Catblocks.

For that we first show that the image of a S2ML model through $G \circ F$.

Let $X_{quint} =< P, C, B, \alpha, r >$ be a S2ML model.

Let $X \in Ob(S2ML + Cat)$, with $X = F(X_{quint})$

Let $X_{1,quint} = G(X) =< P_1, C_1, B_1, \alpha_1, r_1 >$

$Ob(X) = P' \cup C' \cup B'$ where:

- $P' = \{p' \mid p'$ is the Catport such as $Ob(p') = p$, for $p \in P\}$
- $C' = \{\{p'_1, ..., p'_n\} \mid$ For any $k \in [|1, n|]$, p'_k the Catport such as $Ob(p'_k) = p_k$, for $p_k \in c$, with $c = \{p_1, ..., p_n\} \in C\}$

- $B' = \{b' \mid b'$ is the Catblock that contains all associated Catblocks, Catports, and Catconnections that are under b in the transitive closure of the α relation , with $b \in B\}$

Therefore by definition of G:

- $P_1 = \{\ p \in Ob(P_{Cat}) \mid P_{Cat} \in Ob(X)$ is a Catport$\} = P$, since the Catports of X are defined as the Catports that have the ports of X_{quint} as their unique object.
- $C_1 = \{\ c' \mid c' = \{p \in Ob(P_{cat})$ for $P_{cat} \in c \mid c \in Ob(X)$ where c is a Catconnection$\}\} = C$, since the Catconnections of X are defined as the Catconnections that have as objects the Catports associated to the ports of the connections of X_{quint}.
- $B_1 = \{\ b' \mid b'$ is a symbol associated to $b \in Ob(X)$ which is a Catblock$\}$, therefore, if we choose b' as the symbol of the block that b was associated to through F, $B_1 = B$.

  Pragmatically, this means that we choose a convenient name for the blocks, but that any model that is identical to X_{quint} up to the choice of the names of the blocks could also be associated to X, this is what is meant by "Up to the choice of symbols for blocks".
- $\alpha_1 = \{\ (x', y') \in P \cup B \cup C \times B \mid Hom_X(x, y)$ contains a direct belonging morphism, where x, y are the pre images of x' and y' with the three first points $\}$

  Any Catblock b of X contains exactly the Catblocks, Catports and Catconnections that are under it in the transitive closure of the α relation.

  We consider an object $x \in Ob(b)$, either there is a direct belonging between b and $x \in Ob(X)$, or there is a block $b1 \in Ob(b)$ such as $x \in Ob(b1)$, which implies that the pre image x' of x is under the pre image $b1'$ of $b1$ in the transitive closure (i.e. children elements and their children) of α and therefore $(x', b') \notin \alpha$ (with b' pre image of b).

  This means that there is a direct belonging between x and b if and only if the pre images x' and b' of x and b are such as $(x', c') \in \alpha$.

  Thus $(x', c') \in \alpha_1$ if and only if $(x', c') \in \alpha$: $\alpha_1 = \alpha$
- $r1$ is the symbol associated to the root of X.

  The root of X is a Catblock that contains every other Catblock, Catport and Catconnection of X.

  Therefore its pre image is a block of X_{quint} that is over any other block, port and connection in the transitive closure of α, the only corresponding block is r, the only block with no parent.

  Thus $r1 = r$.

This means that, up to the choice of symbols for the blocks, we have $G(F(X_{quint})) = G(X) = X_{quint}$, i.e., $G \circ F = Id$.

Now, let us show that $F \circ G = Id$.
Let $X \in Ob(S2ML + Cat)$.
Let $X_{quint} = G(X) =< P, C, B, \alpha, r >$.
Let $X' = F(X_{quint} = F \circ G(X)$.
$Ob(X') = P' \cup B' \cup C'$ where:

- $P' = \{p' \mid p'$ is the Catport such as $Ob(p') = p$, for $p \in P\}$

  However the $p \in P$ are the p such as there is a Catport $P" \in Ob(X)$ such as $Ob(P") = \{p\}$.

  Thus the Catports in $Ob(X')$ are exactly the Catports in $Ob(X)$.

- $C' = \{\{p'_1, ..., p'_n\} \mid$ For any $k \in [|1, n|]$, p'_k the Catport such as $Ob(p'_k) = p_k$, for $p_k \in c$, with $c = \{p_1, ..., p_n\} \in C\}$.

  However the $c \in C$ are the connections $c = \{p \in Ob(P_{cat})$ for $P_cat \in c'$ with $c' \in Ob(X)$. Thus the Catconnections in $Ob(X')$ are exactly the Catconnections in $Ob(X)$.

- $B' = \{b' \mid b'$ is the Catblock that contains all associated Catblocks, Catports, and Catconnections that are under b in the transitive closure of the α relation , with $b \in B\}$

  However, the α relation is $\{ (x', y') \in P \cup B \cup C \times B \mid Hom_X(x, y)$ contains a direct belonging morphism, where x, y are the pre images of x' and y' with the three first points $\}$, i.e, Catblocks of $Ob(X')$ contain eachothers and Catports/Catconnections exactly in the same way as Catblocks of $Ob(X)$.

  Therefore the Catblocks of $Ob(X')$ and the Catblocks of $Ob(X)$ are the same.

Thus, $Ob(X) = Ob(X')$.
The morphisms in X' are the morphisms described in definition 4.4, therefore, they are the same as the morpisms in X.
This means that $X' = F \circ G(X) = X$.
$F \circ G = Id$.
Therefore we have a way to map each element of $S2ML$ to an element of $Ob(S2ML+Cat)$. We also have a way to map each element of $Ob(S2ML+Cat)$ to a set of elements of $S2ML$. Finally, both these mappings are each-other inverse up to the choice of symbols for the blocks.

□

4.3.2 The Cantor-Bernstein property of models

A remarkable property is that when two models can be injected into one another, they have the same structure; this can be expressed as follows.

Theorem 4.2. *Let $A, B \in Ob(S2ML+Cat)$, such as there exist two injections $F : A \longrightarrow B$ and $G : B \longrightarrow A$.*

Then there exists an injection $G' : B \longrightarrow A$ such as $G' \circ F = Id_A$ and $F \circ G' = Id_B$

Remark 4.1. *Theorem 4.2 is a generalization of the Cantor–Schröder–Bernstein theorem to Catmodels.*

Lemma 4.1. *Let $M_1, M_2 \in Ob(S2ML+Cat)$ such as there exist two injections $F : M_1 \longrightarrow M_2$, $G : M_2 \longrightarrow M_1$.*

Let $R1$ be the root object of M_1, $R2$ the root object of M_2.

Then $R2 = F(R1)$ and $R1 = G(R2)$

Proof. Let $M_1, M_2 \in Ob(S2ML+Cat)$ such as there exist two injections $F : M_1 \longrightarrow M_2$, $G : M_2 \longrightarrow M_1$.

F and G are injective on objects, therefore $card(Ob(M_1)) = card(Ob(M_2))$.

Let $R1$ be the root object of M_1, $R2$ the root object of M_2.

Any belonging morphism between an object and its parent in $M1$ will also be mapped to a belonging morphism in $M2$ between the object's image and its parent's image. Therefore, as F is injective, any object of M_1 with a parent is mapped to an object of M_2 which also has a parent.

Therefore all objects of M_2 that have a parent are the images of objects of M_1 that have a parent, since $card(Ob(M_1)) = card(Ob(M_2))$.

Thus, since F is injective on objects, $R2 = F(R1)$. □

Lemma 4.2. *Let $M_1, M_2 \in Ob(S2ML+Cat)$ such as there exist two injections $F : M_1 \longrightarrow M_2$, $G : M_2 \longrightarrow M_1$.*

Then Order(M1) = Order(M2)

Proof. Let $M_1, M_2 \in Ob(S2ML+Cat)$ such as there exist two injections $F : M_1 \longrightarrow M_2$, $G : M_2 \longrightarrow M_1$.

Let $X \in Ob(M1)$ of order k.

By definition, there exists $X_1, ..., X_{k-1} \in Ob(M1)$ and $r_1 : X \longrightarrow X_1$, ..., $r_k : X_{k-1} \longrightarrow R1$ such as the r_i are direct belonging morphisms.

Therefore there exists $X', ..., X'_{k-1} = F(X), ..., F(X_{k-1})$ and $r'_1 : X' \longrightarrow X'_1$, ..., $r'_{k-1} : X'_{k-1} \longrightarrow R2$, where the r'_i are belonging morphisms which are the images by F of the r_i.

This means that $Order(X') \geq Order(X)$.

If we take the objects of maximum order in M_1 and M_2, we obtain $Order(M_1) \geq Order(M_2)$ and inversely.

Therefore $Order(M_1) = Order(M_2)$. □

Lemma 4.3. *Let $M_1, M_2 \in Ob(S2ML+Cat)$ such as there exist two injections $F : M_1 \longrightarrow M_2$, $G : M_2 \longrightarrow M_1$.*

Let $k = Order(M_1) = Order(M_2)$.

For any $i \in [|0;k|]$ M_1 and M_2 have the same number of objects of order i. Any object of order i in M_1 (resp. M_2) is mapped through F (resp. G) to an object of order i.

Proof. Let $M_1, M_2 \in Ob(S2ML+Cat)$ such as there exist two injections $F : M_1 \longrightarrow M_2$, $G : M_2 \longrightarrow M_1$.

Let $k = Order(M_1) = Order(M_2)$.

Let n be the number of objects of order k in M_1, then M_2 has at least n objects of order k, since $Order(F(X)) \geq Order(X)$.

Since the same holds for G, we have that M_2 also has exactly n objects of order k.

Assume that M_1 and M_2 have the same number of objects of a specific order and objects are mapped to objects of the same order for any order above $i \in [|0;k|]$.

Let $X \in Ob(M_1)$ be an object of order i, then $Order(F(X)) \geq Order(X)$.

Any object of M_2 of order superior to i already is the image of an object of M_1 of order superior to i.

Therefore $Order(F(X)) = Order(X) = i$.

We know that the property holds for objects of order k.

For $i \in [|0;k|]$, if it holds for any order of $[|i+1,k|]$, then it holds for order i.

Thus by complete induction, it holds for any $i \in [|0;k|]$. □

Lemma 4.4. *Let $M_1, M_2 \in Ob(S2ML+Cat)$ such as there exist two injections $F : M_1 \longrightarrow M_2$, $G : M_2 \longrightarrow M_1$.*

The image by F_{Ob} (resp G_{Ob}) of a Catconnection is a Catconnection with the same number of Catports.

Proof. Let $M_1, M_2 \in Ob(S2ML+Cat)$ such as there exist two injections $F : M_1 \longrightarrow M_2$, $G : M_2 \longrightarrow M_1$.

Let C be a Catconnection of $M1$.

$card(Ob(F_{Ob}(C))) \geq card(Ob(C))$: indeed, if $Ob(C) = \{P_1,...,P_n\}$ and $Ob(F_{Ob}(C)) = \{P'_1,...,P'_m\}$, then we have, for each $i \in [|1,n|]$ a l such as $F_{Ob}(P_i) = P'_l$, because F preserves the reference morphisms, thus $m \geq n$

This means that M_2 has more Catconnections than M_1 for a specific number of Catports.

In the same way as objects' order, we obtain that the image of a Catconnection by F and G is a Catconnection with the same number of Catports. □

Proof. Theorem 4.2

Let $M_1, M_2 \in Ob(S2ML+Cat)$ such as there exist two injections $F : M_1 \longrightarrow M_2$, $G : M_2 \longrightarrow M_1$.

Let us show by complete induction that we can build an inverse of F.

Let n be the order of M_1 and M_2.

Let $R1$ be the root object of M_1, $R2$ the root object of M_2.

For $n = 0$, M_1 and M_2 only contain their root objects, which are elementary Catblocks that have no Catport or Catconnection.

Therefore F and G are entirely defined by $F(R1) = R2$ and $G(R2) = R1$; they are trivially inverse of each other.

Let $n > 0$, assume that for any $k < n$, for any models $M_{1,k}, M_{2,k} \in Ob(S2ML + Cat)$ of order k such as there exist two injections $F_k : M_{1,k} \longrightarrow M_{2,k}$, $G_k : M_{2,k} \longrightarrow M_{1,k}$, we can build an inverse of F.

Then:

- F maps $R1$ to $R2$ and vice versa.
- F maps each Catblock $B \in Ob(M_1)$ of order 1 to a Catblock $B' \in Ob(M_2)$ of order 1.

  The component of F on B and its descendants corresponds to an injection F_B between the models (of order $< n$) that have B and B' as their root objects.

  Thus we can associate an inverse G'_B to this injection.
- M_1 and M_2 have the same numbers of Catports of order 1 and connections of order 1.

  - Let $C \in Ob(M_1)$ be a Catconnection of order 1.

  We know that $F(C)$ has the same number of Catports as C and that the images of the Catports of C are the Catports of $F(C)$.

  Therefore we can define an inverse to the component of F on C and its ports, and this inverse is compatible with the ones defined for Catblocks of order 1.

  - We can associate two by two the remaining Catports.

  When we add the compositions to these components, we obtain a functor $G' : M_2 \longrightarrow M_1$, which is the inverse of F.

  Therefore we have shown theorem 4.2 by strong induction.

□

A direct corollary of theorem 4.2 is the following one.

Corollary 4.1. *Let $A, B \in Ob(S2ML + Cat)$, such as there exists injections $F : A \longrightarrow B$ and $G : B \longrightarrow A$.*

Then F is an equivalence of categories.

As a reminder, the definition of an equivalence of categories is given in the state of the art of this manuscript, in Section 1.4.3.

4.3.3 Equivalence of S2ML models

We can therefore give a simple definition of the equivalence of Catmodels.

Definition 4.11. ***"Equivalence of Catmodels"***

Let $A, B \in S2ML+Cat$.

We say that A and B are equivalent if there exist $F : A \longrightarrow B$ and $G : B \longrightarrow A$ two injections.

Theorem 4.3. *The equivalence of Catmodels is an equivalence relation over $Ob(S2ML+Cat)$.*

Proof. We need to show that this relation is reflexive, symmetric, and transitive.

- **Reflexivity**:

  Let A be a Catmodel.

  A is equivalent to A because the identity Id_A is an injection. Therefore $F = G = Id_A$ and α composed of, for each $x \in A$, $\alpha_x : F(x) \longrightarrow G(x)$ with $\alpha = id_x$ suits.

- **Symmetry**:

  A is equivalent to B, so the injections $F : A \longrightarrow B$ and $G : B \longrightarrow A$ exist. Therefore we have two injections between B and A, and as such, B is equivalent to A.

  The relation is symmetric.

- **Transitivity**:

  Let A, B, and C three Catmodels, with A equivalent to B and B equivalent to C.

  Let $F_{A,B}, G_{B,A}$, be the equivalence injections for A, B and $F_{B,C}, G_{C,B}$ be the equivalence injections for B, C.

  Then $F : A \longrightarrow C = F_{B,C} \circ F_{A,B}$ and $G : C \longrightarrow A = G_{B,A} \circ G_{C,B}$ are two injections between A and C, because the composition of injection is an injection, as shown in the proof of proposition 4.2. Therefore A and C are equivalent.

□

Another property of injection is that they, in a way, represent that one model is bigger than another since it is a part of it.

Definition 4.12. ***"Injection relation"***
For $A, B \in Ob(S2ML+Cat)$, we say that a Catmodel A is injected into a Catmodel B if there exists an injection $F : A \longrightarrow B$.

We call this relation the injection relation over the set of Catmodels.

Theorem 4.4. *The injection relation is a partial order relation over the set of Catmodels, given equivalences between Catmodels.*

Proof. We need to show that the injection relation is reflexive, antisymmetric, and transitive.

- **Reflexivity**:

  Let X be a Catmodel, let Id_x be the identity over X, i.e. the image of $x \in Ob(X)$ is x and for $g \in Hom_X(x, y)$ with $x, y \in Ob(X)$ the image of g is g.

  Then Id_X is trivially an injection.

  Therefore X is injected into X.

- **Antisymmetry**:

  Assuming X, Y, two Catmodels, such as X is injected in Y and Y is injected in X, with $F : X \longrightarrow Y$ and $G : Y \longrightarrow X$ such injections.

  Then we have two injections between X and Y; therefore X and Y are equivalent.

- **Transitivity**:

  Let X, Y, Z be Catmodels.

  Let $F : X \longrightarrow Y$, $G : Y \longrightarrow Z$ be injections.

  Let $H : X \longrightarrow Z$ be the composition of F and G on objects and morphisms, i.e., composed of:

  $$\begin{cases} H_{Ob} : & Ob(X) \longrightarrow Ob(Z) \\ & x \rightarrow G \circ F(x) \end{cases}$$

  and for each $x, y \in Ob(X)$,

  $$\begin{cases} H_{x,y} : & Hom_X(x, y) \longrightarrow Hom_Z(H_{Ob}(x), H_{Ob}(y)) \\ & f \rightarrow G_{F_{Ob}(x), F_{Ob}(y)} \circ F_{x,y}(x) \end{cases}$$

  Because each of the applications defined here are the composition of injective applications, they are injections. Therefore H is an injection; thus X is injected in Y.

Therefore the existence of these injections between Catmodels defines a partial order relation. □

Figure 4.13: The structure of a binary consistency relation

4.4 Consistency relation

Now that we have explained the category in which we are working, we can define a consistency relation between two S2ML models.

Definition 4.13. ***"Binary consistency relation"***

Let $\sim$ be a binary relation over $Ob(S2ML+Cat)$, for $A,B \in Ob(S2ML+Cat)$ we note $A \sim B$ if $(A,B) \in \sim$, i.e. if A is in relation with B.

$\sim$ is a Binary consistency relation if and only if for $A,B \in Ob(S2ML+Cat)$, there exist $A',B' \in Ob(S2ML+Cat)$ such as these injections exist:

- $f_A : A' \longrightarrow A$
- $f_B : B' \longrightarrow B$
- $F : A' \longrightarrow B'$
- $G : B' \longrightarrow A'$

and F and G respect the following properties:

- $F \circ G = Id_{B'}$
- $G \circ F = Id_{A'}$

This definition means that the two Catmodels are in a consistency relation if the structure shown in Figure 4.13 exists with f_a, f_b, F, and G being injections. All the morphisms being injections and the compositions of F and G being the identities of A' and B'. Note that theorem 4.2 implies that if this structure exists without the compositions of F and G being the identities, then we can find G' such as F and G' compose into the identities.

Such a relation is too general to describe a relevant specification of consistency between two Catmodels. To illustrate this idea, we can create a binary consistency relation such as any couple of Catmodels would be in the relation:

Proposition 4.3. *Let $\sim_{all}$ be the binary relation over $Ob(S2ML+Cat)$ with no other restriction than being a binary consistency relation.*

Let $A,B \in Ob(S2ML+Cat)$.

Then $A \sim_{all} B$.

Proof. Let $A, B \in S2ML + Cat$.

We have $A \sim_{all} B$ if there exists A', B' in $Ob(S2ML + Cat)$ such as there exist injections $f_A : A' \longrightarrow A$, $f_B : B' \longrightarrow B$, $F : A' \longrightarrow B'$, $G : B' \longrightarrow A'$ such as $F \circ G = Id_{B'}$ and $G \circ F = Id_{A'}$.

Because A and B are Catmodels, there exists unique Catblocks $r_A \in Ob(A)$ and $r_B \in B$ such as there are no morphisms in A (resp B) with domain r_A (resp r_B).

Let A' be the S2ML model with only one object r_A and no morphisms, and B' be the Catmodel with only one object r_B and no morphisms.

$\begin{cases} f_A : & A' \longrightarrow A \\ & r_A \rightarrow r_A \end{cases}$ is trivially an injection. (We only present the injections through their mappings over $Ob(X)$ here since the models have no morphisms)

We define f_B using the same construction.

Let $\begin{cases} F : & A' \longrightarrow B' \\ & r_A \rightarrow r_B \end{cases}$ and $\begin{cases} G : & B' \longrightarrow A' \\ & r_B \rightarrow r_A \end{cases}$.

F and G are injections, and $F \circ G = \begin{cases} B' \longrightarrow B' \\ r_B \rightarrow r_B \end{cases} = Id_{B'}$ and similarly $G \circ F = Id_{A'}$.

Therefore $A \sim_{all} B$. □

Our definition of a binary consistency relation describes the existence of a similarity between A and B. This similarity is the shared structure of A' and B'.

Because the definition is general, the fact that A and B are Catmodels induces a similarity: The existence of the `main` element of the Catmodel, namely r_A and r_B, unique elements of the Catmodels with no parents. This results in any couple of Catmodels being in relation.

This means that not all binary consistency relations are interesting for Catmodel comparison.

The main idea behind this definition is that we can complete it with restrictions to assert consistency over specific properties of the Catmodels. As an example, we define the dictionary consistency relation between two Catmodels:

Definition 4.14. ***"Dictionary consistency relation"***

We call a Dictionary consistency relation a binary consistency relation $\sim_{dic}$ *such as for* $A, B \in Ob(S2ML + Cat)$, $A \sim_{dic} B$ *implies:*

A' *and* B' *are two Catmodels injected in* A_{comp} *and* B_{comp}*; where* A_{comp} *and* B_{comp} *are two Catmodels such as, appart for their root Catblock,* $Ob(A_{comp})$ *and* $Ob(B_{comp})$ *contain only elementary Catblocks and Catports.*

Such a relation consists of matching objects from one Catmodel to corresponding objects on the other, except for some elements that are kept unmatched. Therefore this method can detect elements from one Catmodel that have no equivalent in the other Catmodel; they are the objects of A_{comp} that are not in A' (idem on the B side).

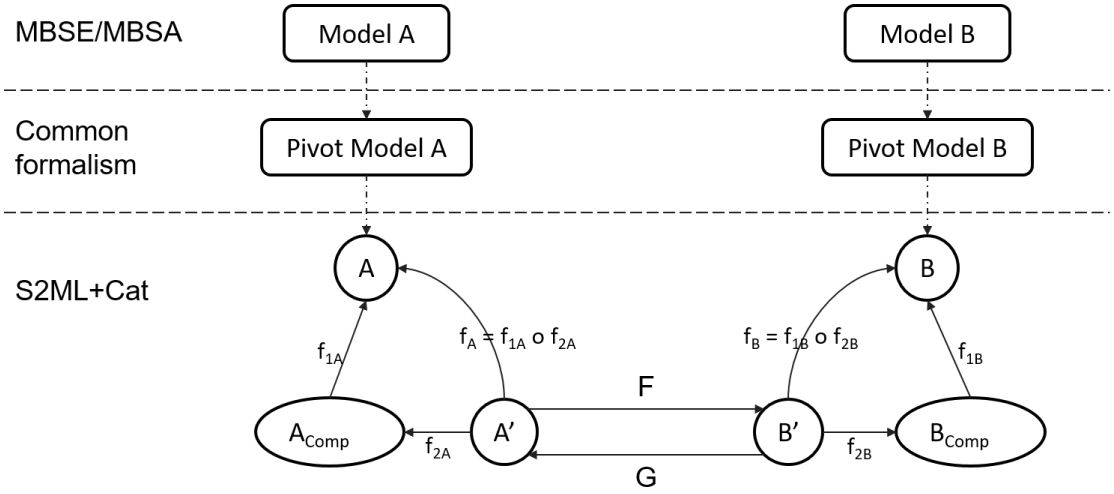


Figure 4.14: Reminder of the structure for binary consistency relations

As before, it could be argued that any couple of models is in relation using this definition, as we could also take the Catmodels with only the main elements. For example, this could be solved by having a unique "name" attribute in our objects and considering that objects linked by the dictionary should carry the same name. However, that would hide the real problem since there is no guarantee that elements are named the same way in the original Catmodels. The reality is that such a dictionary would have to be created by the engineers, as the Catmodel elements carry a meaning that the computer cannot understand. Generally, the way to express a "good" consistency relation is to have a maximality condition on the A' and B' models. One way to achieve this maximality is through the use of a pullback. We discuss these ideas in section 5.3.1.

While defining the dictionary relation, we used the A_{comp} and B_{comp} Catmodels as an intermediary step between A/B and A'/B'. We use these Catmodels to eliminate the elements we do not intend to compare in our binary consistency relation between A and B. In this way, the differences detected by the binary consistency relation are the objects from A_{comp} and B_{comp} that are not in A' and B'.

Therefore one possible structure to build a binary consistency relation over an existing synchronization methodology is the one found in Figure 4.4, a reminder of this figure is found here in Fig. 4.14. $S2ML+Cat$ could also be replaced by an equivalent construction over another common formalism.

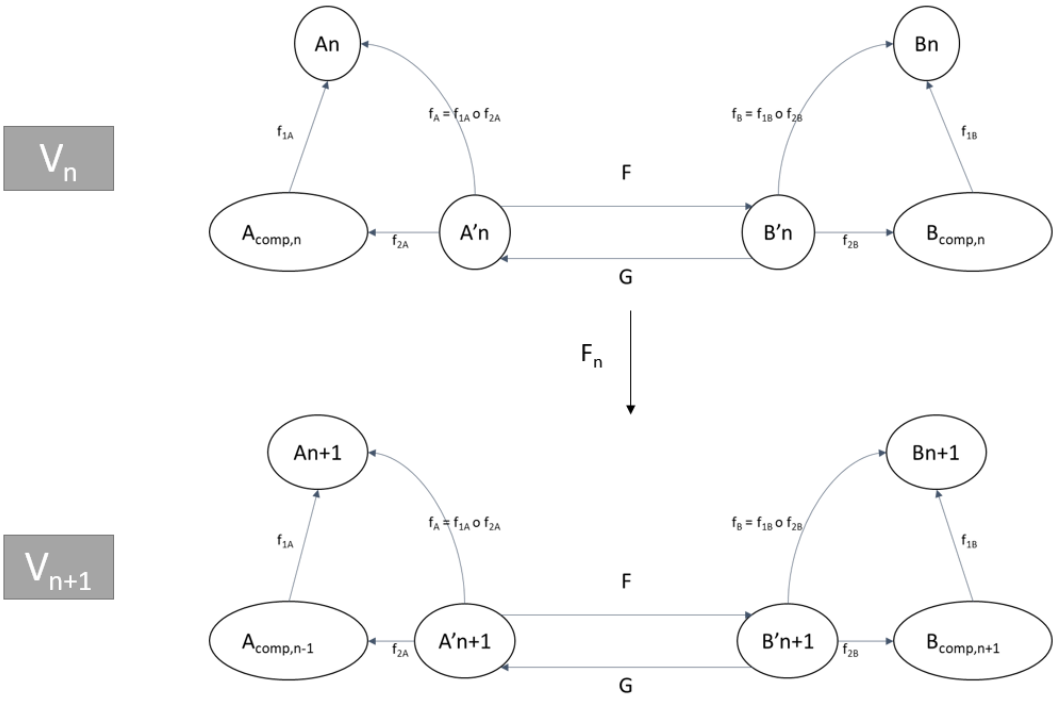


Figure 4.15: Functors between iterations of the model consistency structure

4.5 Simplifying comparison with versioning

The S2ML+Cat framework allows us to give mathematical definitions of consistency relations, but we believe that it is not the only way that it could be helpful. During the design and safety assessment of the system, iterations are made over the models. Because the models evolve, multiple consistency assessments are required each time a new version of the Catmodels is used for consistency assessment.

We believe that once an assessment has been made over the first version of the models, we can simplify the subsequent comparisons, using the knowledge that most of the models remain unchanged and therefore do not need to be compared again. Take the construction from Fig. 4.14 for the nth version of the Catmodels. We call A_n and B_n the S2ML models, $A_{comp,n}$ and $B_{comp,n}$ the comparison Catmodels, and A'_n and B'_n the compared Catmodels as a category, let us call it V_n. We should be able to define a functor $F_n : V_n \longrightarrow V_{n+1}$ to represent the correspondences between the nth and $n+1$th versions of the Catmodels, as depicted in Fig. 4.15.

This functor would have components on each of the categories of V_n, mapping A_n to A_{n+1} etc. These components would map each Catmodel element of

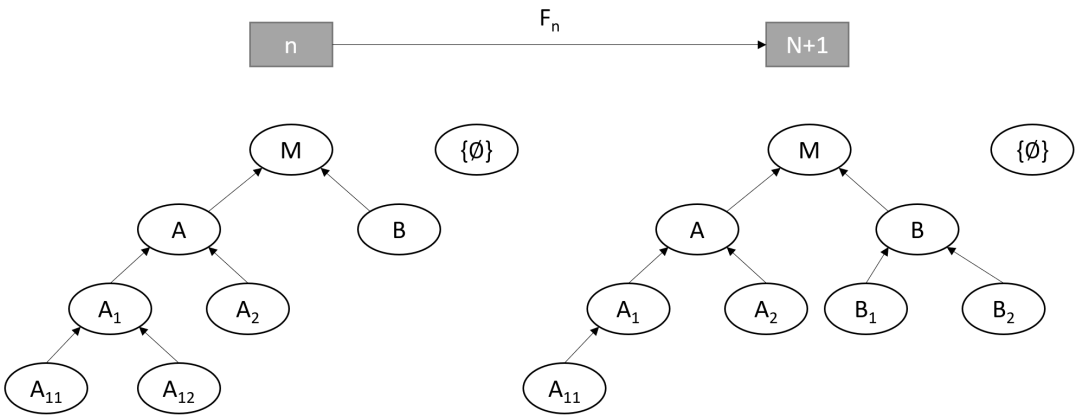


Figure 4.16: Mapping version n to version n+1

version n to its corresponding element in version $n+1$. Obviously, we have a problem if an element is deleted. Therefore this new idea requires a bit of work on the original concept of S2ML+Cat; we should be able to solve this issue by adding a new object, for example, $\{\emptyset\}$, and mapping the deleted elements to this new object.

CHAPTER 5

APPLICATION EXAMPLE: THE BLOOD DELIVERY DRONE

In this chapter, we present the blood delivery drone case study and apply the S2ML+Cat framework to the SmartSync synchronization of its models as proof of concept.

With the help of an intern at the Quartz - Supméca laboratory, we reverse-engineered MBSE, MBSA, multi-physics, and scenario models of the Zipline blood delivery drone. We considered the MBSE model as the specification of the system and synchronized the MBSA, multi-physics, and scenario models with MBSE. Thanks to a python script, we can compute and visualize the S2ML+Cat categories corresponding to each step of the synchronization process.

The first section presents the case study and the modeling of the MBSE, MBSA, Multi-Physics, and Scenario models. The second section applies the SmartSync methodology to the case study and shows how S2ML+Cat can be applied to the methodology. The third section discusses the results of this experiment through the angles of pragmatics, composition in the models, and connections.

5.1 Presentation of the case study

The study uses a fixed-wing drone as its applicative example. This drone is inspired by the Zipline company blood delivery drone [4]. The specific use of this drone is the delivery of blood packages to different hospitals and clinics across a large area from a blood storage center. This drone is a fixed-wing drone, meaning a drone similar to a plane. Its powertrain has two coaxial motors, each linked to a propeller. The drone also presents redundancy in its ailerons, used to control direction. A control processing unit is placed in the drone's remov-

able battery and given the flight plan before each flight. A QR code allows for identifying the blood package once installed in the drone cargo compartment until it is parachuted down to the delivery site. The drone takes off by being catapulted; then, it flies to the drop area. Cargo can be parachuted up to an 80 km radius, with a maximum time to the objective of 45 min. The drone then comes back to the storage site. It lands by being caught by a recovery system, a sling between two 10-m high towers that attaches to a hook on the drone. Figure 5.1 illustrates the drone's external architecture and flight scenario.

The modeling of this drone was made with the assistance of Imane Bouhali, an intern at the Supmeca Quartz Laboratory at the time.

5.1.1 Modeling the case study

We consider a SysML architecture model as the specification of the system. From this architecture, we derived three models for different intents:

- **Safety assessment:** An AltaRica 3.0 model represents the model for safety assessment.
- **Scenario:** A SCOLA (SCenario Oriented LAnguage) model represents the functional scenarios.
- **Multi-physics:** A Modelica model represents the multi-physics behavior of the drone's power electronics and aerodynamics.

We asserted consistency over this design using SmartSync to synchronize these three models with the architecture model.

MBSE:

The MBSE modeling of the drone was conducted through the same process as the Landing Gear case study (see Section 2.2.1), using the methodology from [86]. In this section, we are only interested in the drone's architecture.

The drone is decomposed into different parts: Unit, Avionic, Controller, Radar, Inertial Unit, Powertrain, and Battery, which are then decomposed into components. This decomposition, described by the BDD, can be found in Fig. 5.2.

We then built the IBD (found in Fig. 5.3) to represent how these different parts interact.

The Unit contains the wings and Ailerons of the system, as well as the rudder and the cargo bay. The powertrain contains the motors and propellers. Most parts of the system are connected to the controller.

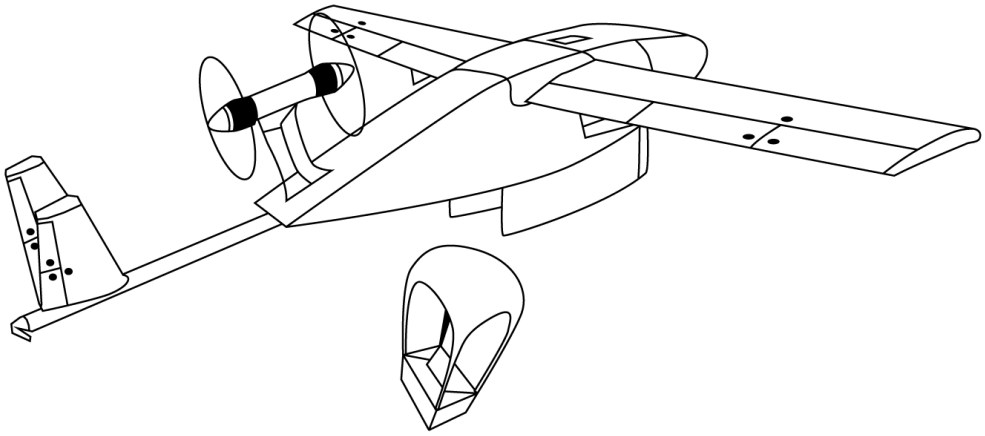


Figure 5.1: Illustration of the Zipline Flyer drone.

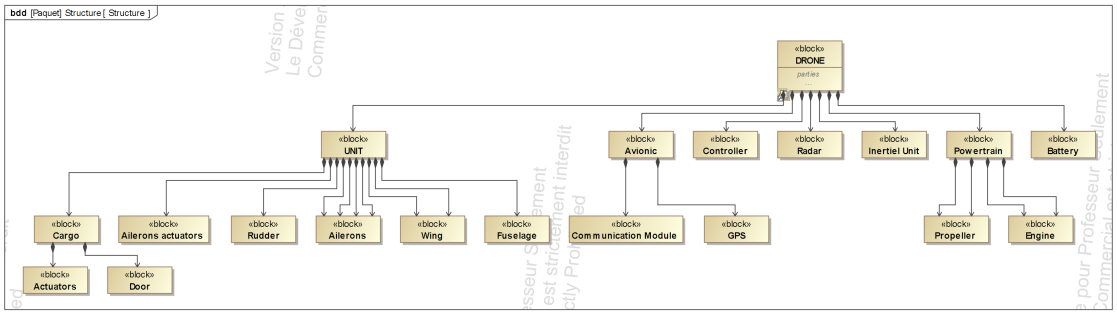


Figure 5.2: Product Breakdown Structure of the Blood Delivery Drone System (BDD)

MBSA:

The MBSA model is based on an NRComponent class similar to the NonRepairableComponent class we defined for the landing gear in Section 2.2.2.

This class that contains the basic OK/KO guarded transition system declined in each component of the drone is completed through a ComponentIO class that contains connected input and output variables. Those classes can be found in Fig. 5.4.

The complete AltaRica Model can be found in Appendice C.1.

Scenario:

We use the SCOLA to represent scenarios over the system. The SCOLA model contains a very simplified architecture of the system, found in Fig. 5.5, along

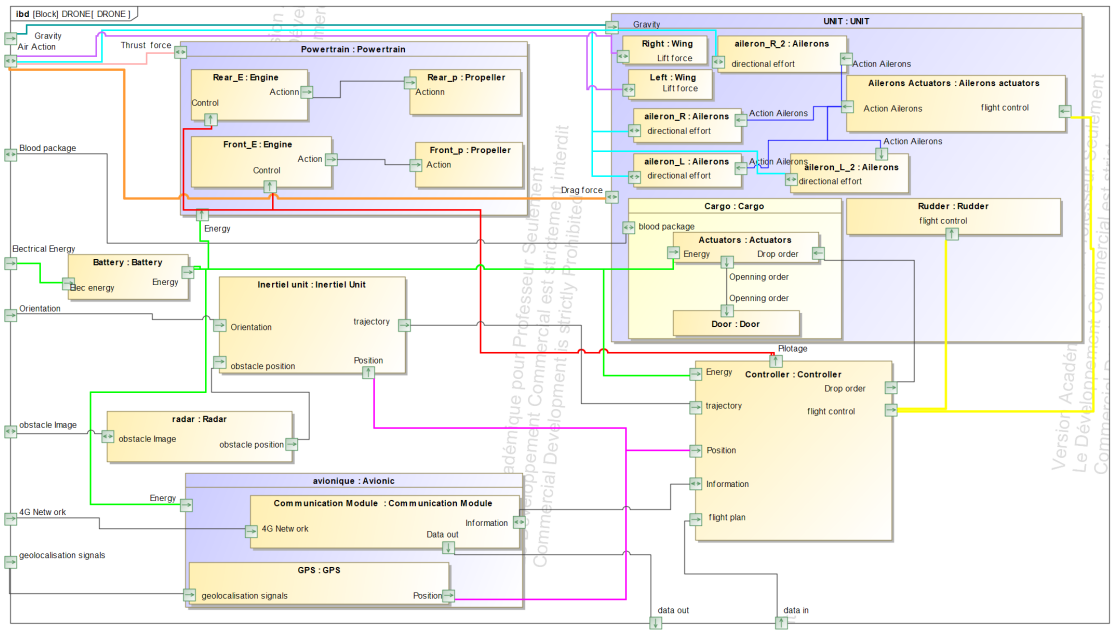


Figure 5.3: Physical Architecture of the Blood Delivery Drone System (IBD)

with a scenario that dynamically modifies the architecture. The scenario is detailed in appendice C.2.

As an example, the block "colis" (which is french for shipment) contains a boolean variable initialized at true; when the scenario enters the task "language" (drop), this variable is changed to false, as the payload is no longer in the drone.

Multi-Physics:

The multi-physics model is built with Modelica, a modeling language associated with a graphic interface. The graphical view of the model is illustrated in Fig. 5.6.

The Modelica model contains two motors, along with two propellers, that are associated with their electrical and aerodynamic equations. It also contains the wings and their aerodynamics. The motors are controlled by a PID that aims at keeping the drone at a fixed altitude.

The Modelica model intends to prove that the drone's performance allows it to fly at its designated altitude and speed. We can also deteriorate the drone (for example, turn off one of the motors) and verify if the drone's performances are still enough to accomplish the mission. The model contains computation artifacts unrelated to the drone's architecture; for example, the "World" is a fixed element that allows computing the drone's position.

```
class NRComponent
    Boolean is_working (init = true);
    parameter Real lambda = 10e-6;
    event failure (delay = exponential(lambda));
    transition
        failure : is_working == true -> is_working := false;
end

class ComponentIO
    extends NRComponent;
    Boolean in, out (reset = false);
    assertion
        out := if is_working then in else false;
end
```

Figure 5.4: Class NRComponent and ComponentIO in AltaRica 3.0

```
domain ETAT_GPS {ARRV_base, ARRV_dest, NONARRV_base, NONARRV_dest}
   end
block drone
    block GPS
    end
    block colis
        Boolean in true
    end
    block GroupeMoteur
        Boolean inPower1 false
        Boolean inPower2 false
    end
end
```

Figure 5.5: Representation of the architecture in the SCOLA model

5.2 Applying SmartSync to the Study Case

This section applies the mathematical framework to a case study through the SmartSync approach. We show how the application of SmartSync on this case study can be expressed in our framework, but the mathematical definition of SmartSync within our framework is yet to come. This will be discussed in Section 5.3. In the first subsection, we show how we assert consistency between the models, using SmartSync. Then, subsection 5.2.2 provides a categorical expression of the SmartSync comparison steps.

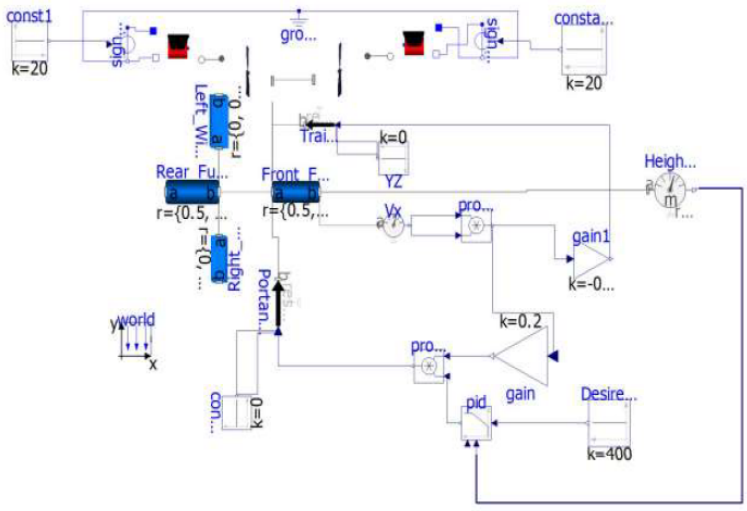


Figure 5.6: Modelica Graphical view of the Drone

5.2.1 Comparison with SmartSync

Before conducting each comparison, we first had to abstract the SysML model. The abstraction was made towards the S2ML language used in the SmartSync methodology.

We translated the information from the BDD and the IBD to conduce the abstraction. The S2ML model followed the BDD's hierarchy and the IBD's structure. We represented SysML "`partProperties`", i.e., components of the system in the S2ML model, with blocks. We also used S2ML ports to represent SysML ports and S2ML connections to represent SysML connections.

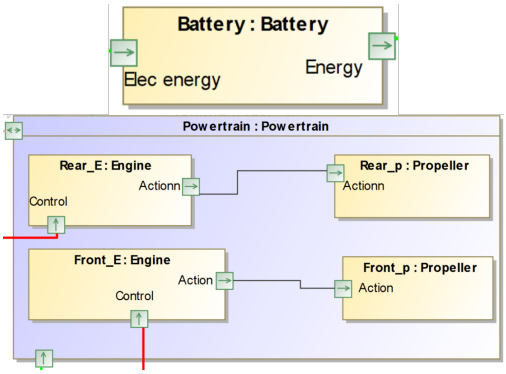
An example of the transformation over part of the SysML model (namely the battery and drivetrain) is shown in Figure 5.7. This translation is similar to the one operated on the Landing Gear models in Section 2.3.1, the complete S2ML model is found in appendice C.3.

Once we translated the SysML model, we operated the comparison with the AltaRica model and the Modelica model. This was done by translating those models, then operating the SmartSync comparison.

MBSE/MBSA:

To compare the SysML model and the AltaRica 3.0 model, we first translated the AltaRica 3.0 model to S2ML. This process was not complicated because the structural part of the AltaRica 3.0 languages is S2ML. Blocks were translated to blocks, variables to ports, and assertions to connections. An example of this translation can be found in Figure 5.8. In this example, it can be seen that some attributes giving more information on the AltaRica model were included in the S2ML model (e.g., "`type = "Boolean"`"); these attributes are not currently used

```
block DRONE
    block Battery
        port Elec_energy;
        port Energy;
    end
    block PowerTrain
        block Rear_E
            port Action;
            port control;
        end
        block Rear_p
            port Action;
        end
        block Front_E
            port Action;
            port control;
        end
        block Front_p
            port Action;
        end
        port propulsive_effort;
        port Energy;
        connection [Front_motor.Action,
             Front_propeller.Action];
        connection [Rear_motor.Action,
             Rear_propeller.Action];
    end
```



(**a**) (**b**)

Figure 5.7: A part of the SysML model (**b**) and its translation in the S2ML model (**a**).

in the SmartSync comparison. This translation is similar to the one operated on the Landing Gear models in Section 2.3.1, the complete S2ML model can be found in appendice C.4.

Once the translation was done, the models were given to the SmartSync tool as inputs. The tool then took an element of each model that it knew to correspond to one another. This was the main element of the models for the first step, i.e., the blocks representing the whole system. The tool then asked the user to align children of these two elements from both models. The user can align elements or give them the "`forget`" attribute — meaning the element is not supposed to have a counterpart in the other model — or do neither. The tool then iterated on each couple of aligned elements until all elements were either aligned or forgotten. When no element could be aligned anymore, the remaining elements with no counterpart were considered inconsistent.

In this case, the comparison between the architecture and AltaRica 3.0 models was made through 4 comparison steps. It shows that the MBSA model lacks some elements of the system. The missing elements are the redundant ailerons and the rudder airfoil. This is due to a communication issue when the safety model was designed based on an outdated version of the architecture model. Correction of these inconsistencies was conducted by adding the missing elements to the AltaRica 3.0 model.

Note that all safety artifacts were ignored in this comparison, such as state

```
block Avionics
    block Comms_module
        port is_working (type = ''Boolean'',
            kind = ''init'', value =
            ''true'');
        port lambda (type = ''Real'', value
            = 10e-6);
        port failure (type = ''event'',
            delay = ''exponential(lambda)'');
        connection [failure, is_working,
            is_working] (type =
            ''connection'', expr = ''true ->
            false'');
        port in4G (type = ''Boolean'', kind
            = ''reset'', value = true);
        port inCalculator (type =
            ''Boolean'', kind = ''reset'',
            value = ''false'');
        port outData (type = ''Boolean'',
            kind = ''reset'', value = true);
        port outCalculator (type =
            ''Boolean'', kind = ''reset'',
            value = true);
        connection [outData, is_working,
            in4G](type = ''assertion'');
        connection [outCalculateur,
            is_working, in4G](type =
            ''assertion'');
    end
```

(**a**)

```
block Avionics
    block Comms_module
        extends NRComponent;
        Boolean in4G (reset = true);
        Boolean inCalculator (reset = false);
        Boolean outData (reset = false);
        Boolean outCalculator (reset = true);
        assertion
            outData := if is_working then
                in4G and inCalculator else
                false;
            outCalculator := if is_working
                then in4G else false;
        end
```

(**b**)

Figure 5.8: A part of the AltaRica 3.0 model (**b**) and its translation in the S2ML model (**a**).

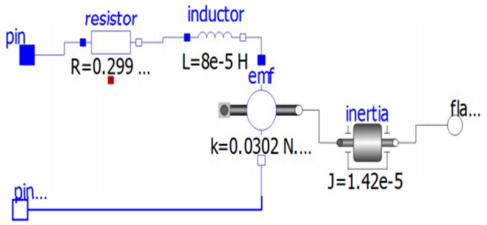
variables, events, etc. The comparison of the models also has shown that interactions with the environment, such as gravity, 4G network, etc., have not been considered in the MBSA model. We deemed this normal in this work, but it could be argued that it should be added; therefore, consistency assessment would have resulted in adding these interactions to the MBSA model. Such decisions are non-trivial and should be undertaken by the system engineer and safety analyst; this is why the consistency assessment cannot be fully automatized. Of course, after modification of the model, a new comparison should be operated to assert that the correction did not introduce new inconsistencies.

MBSE/Multi-Physics:

We needed to make a second comparison to assert the global consistency of our design with the Modelica/SysML comparison. The translation from Modelica to S2ML was done following the process described in [13], with the difference that we considered Modelica's variables. Part of this translation can be found in Figure 5.9 with the translation of the motor class. Modelica classes were abstracted to S2ML classes, with model instances being translated to blocks. Modelica variables were translated to S2ML ports, and connect clauses were translated to connections. The complete S2ML model can be found in appendice C.5.

```
class motor
    port pin, pin_n, flange_b;
    block resistor
        port _in, _out;
        port r (value=''0.299'');
    end
    block inductor
        port _in, _out;
        port l (value=''8e-5'');
    end
    block emf
        port _in, _out, flange;
        port const (value=''0.0302'');
    end
    block inertia
        port _in, _out;
        port J (value=''1.42e-5'');
    end
    connection [pin, resistor._in];
    connection [resistor._out, inductor._in];
    connection [inductor._out, emf._in];
    connection [emf._out, pin_n];
    connection [emf.flange, inertia._in];
    connection [inertia._out, flange_b];
end
```

(**a**)



(**b**)

Figure 5.9: A part of the Modelica model (**b**) and its translation in the S2ML model (**a**).

Once this translation was conducted, we compared the models with the SmartSync tool. A first iteration showed that the structure of the models is very different. For ease of comparison, we added `drivetrain` and `cell` blocks to the models, respectively, containing the motors and propellers and the fuselage and wings. We then translated again and compared the models.

After four comparison steps, we had three conclusions:

- Most components of the drone are missing in the Modelica model. This can be explained by the fact that we are here only interested in the propulsion and aerodynamics of the drone; therefore, components related to other functions are not represented.

- In some specific cases, the Modelica representation, is more detailed than the architecture model. This is the case of the motors since the SysML model only considers them through a black box view, whereas the Modelica view shows the inner parts of the motor for calculations.

- There is an inconsistency between the models. This inconsistency is the presence of the fuselage within the Modelica model, which is modeled for aerodynamic purposes, but unrepresented in the architecture model. This inconsistency is corrected by adding the fuselage to the architecture model.

MBSE/Scenario A third comparison is dedicated to MBSE/SCOLA synchronization. Because the SCOLA language is derived from S2ML, the translation is straightforward. We translate blocks to blocks and ports to ports in the architecture. We also translated the scenario part of the model, although it is not very interesting for comparison. The scenario was modeled as a block, and so were tasks and branches. Nexts were translated as connections between the states, which were translated as ports.

An example of the translation of the block GPS from the architecture can be found in Fig. 5.10.

```
block GPS
    ComponentState Recept_GPS STOPPED
    EtatGPS Position BASE
end
```

(**a**)

```
block GPS (type="{block, scenario, task}",
     value="block")
    port Recept_GPS (type="{INSTALLED,
         MISSING, STOPPED,
         EN_MARCHE}",value="STOPPED");
    port Position (type="{BASE, DEST,
         Route_Retour_BASE,
         Route_vers_DEST}",value="BASE");
end
```

(**b**)

Figure 5.10: A part of the SCOLA model (**b**) and its translation in the S2ML model (**a**).

The SmartSync comparison between the SCOLA and MBSE models is completed after three comparison steps. The comparison results are interesting because the SCOLA model only covers a small part of the architecture. Therefore, most model elements from the MBSE model are marked as `"forget"`. On the contrary, all elements from the SCOLA model are found in the MBSE model, except for two exceptions: GPS and Soute. These elements are lacking because they are found at a different level of abstraction in the MBSE model; this can be easily corrected in the comparison by adding an intermediate subsystem to the SCOLA model. However, it would be better for SmartSync to encompass this case, as will be discussed in section 5.3.2.

Of course, the scenario from the SCOLA model has no equivalent found in the SysML model. Although we could create a scenario through a sequence diagram and translate it to S2ML, we consider that in a realistic workflow, this scenario would only be created in one of the models. Finally, some variables from the SCOLA model do not have an equivalent in the MBSE model, for example, `"batterie.charged"`. These variables are indicators of the components related to the scenario.

5.2.2 Categorical Point of View

In this subsection, we analyze the comparisons made in subsection 5.2.1 in the context of the mathematical framework we defined in Chapter 4. We intend to

show how we can specify the SmartSync comparison of the drone models in the S2ML+Cat framework.

Visualisation of S2ML + Cat Catmodels with NetworkX:

In order to achieve this goal, we used oriented graphs that give a visual representation of the categorical S2ML models.

We created this representation at each step during the comparison to visualize which parts of the model the comparison traversed. Because SmartSync makes the comparison in a way that resembles a breadth-first search, this corresponds to taking the model with only its first level of abstraction (the main and its children) and then adding a level of abstraction for each iteration.

To obtain the graph representations, we wrote a python script that converts the CSV files created by the SmartSync tool to directed graphs over the underlying S2ML + Cat Catmodels. This script used the NetworkX package [55] as its basis for graph classes and visualization methods. This package is a library for the study of graphs and networks. It features classes for the representation of graphs and digraphs and methods for graph analysis and visualisation.

The graphs we show here over the S2ML + Cat Catmodels only show direct belonging morphisms, Catblocks, and Catports. The absence of composed morphisms is because we wanted to minimize the number of edges to allow for good readability. The absence of connections is related to the comparison made by SmartSync, which does not consider them yet.

For each iteration, the script also computed a correspondence matrix between objects of both models, which corresponds to giving the F and G functors from Figure 4.4. This matrix is represented as a table. Its element $A_{n+1,m+1}$ will contain -1 if the nth element of the first Catmodel or the mth element of the second has the attribute forget, 1 if they are aligned and 0 otherwise. The first column and line of the table, respectively, contain the first and second Catmodel elements' names, meaning $A_{n+1,0}$ contains the name of the nth element of the first Catmodel, and $A_{0,m+1}$ the name of the mth element of the second one, with the elements being arbitrarily numbered.

In this aspect, we can say that for each iteration, we have constructed the A_{Comp} and B_{Comp} S2ML + Cat Catmodels for the elements that were already aligned and the F and G functors. The A' and B' S2ML + Cat Catmodels are obtained by removing the objects with the attribute `forget` and the morphisms with these objects as the source or target.

The script can be found in appendice D.

Comparison with the S2ML + Cat Catmodels:

Figure 5.11 shows said graphs for comparing the fixed-wing drone's SysML and AltaRica 3.0 models. Blocks are represented with green vertices and ports with blue ones, and different concentric ellipses show diverse abstraction levels in the models of vertices.

The figure shows the graphs representing the Catmodels for the objects of the models that have been considered by the SmartSync tool. They represent the first, the second, and the last (fourth) comparison steps in the MBSE/MBSA comparison.

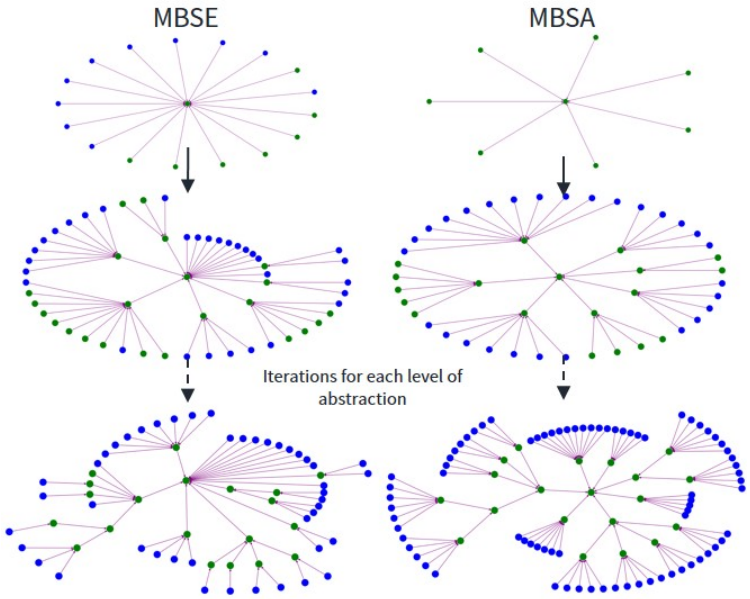


Figure 5.11: Diagrams for the S2ML models of the fixed-wing drone during iterations of SmartSync over the architecture and safety models.

Table 5.1 provides the correspondence matrix for the fixed-wing drone between the SysML and AltaRica 3.0 S2ML models. It can be observed that many ports are present in the SysML representation, whereas they are not in the AltaRica 3.0 model. This corresponds to the interfaces outside of the system discussed in Section 5.2.1.

From the final step of the comparison between the SysML and AltaRica 3.0 models, we can obtain the A' and B' Catmodels from Figure 4.4. These Catmodels are shown in Figure 5.12; this figure is obtained by giving as input the XML corresponding to the results found in appendice C.7 as an input to the visualization script. We observe that although these graphs have different names for their vertices, they are the same graphs, meaning we indeed found a common skeleton between both Catmodels. This property characterizes the way we mathematically define a binary consistency relation.

5.3 Discussion

Although this mathematical framework does not provide a new synchronization methodology, it allows us to observe the SmartSync methodology from

	ZippyFlyer	Avionics	Battery	Calculator	Cell	Inertial_measurement_unit	Power_unit	Radar
Alternative_1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AirAction	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.1
Battery	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
Data_in	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.1
Data_out	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.1
Ener_elec	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	-1.1
Gravity	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Obstacle_image	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Orientation	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Blood_bag	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
4G_network	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Geolocation_signak	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Avionics	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
Calculator	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Cell	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
Inertia_measurement_unit	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
Power_unit	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
radar	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Table 5.1: Correspondence table for the first iteration of the comparison between the SysML and AltaRica 3.0 models

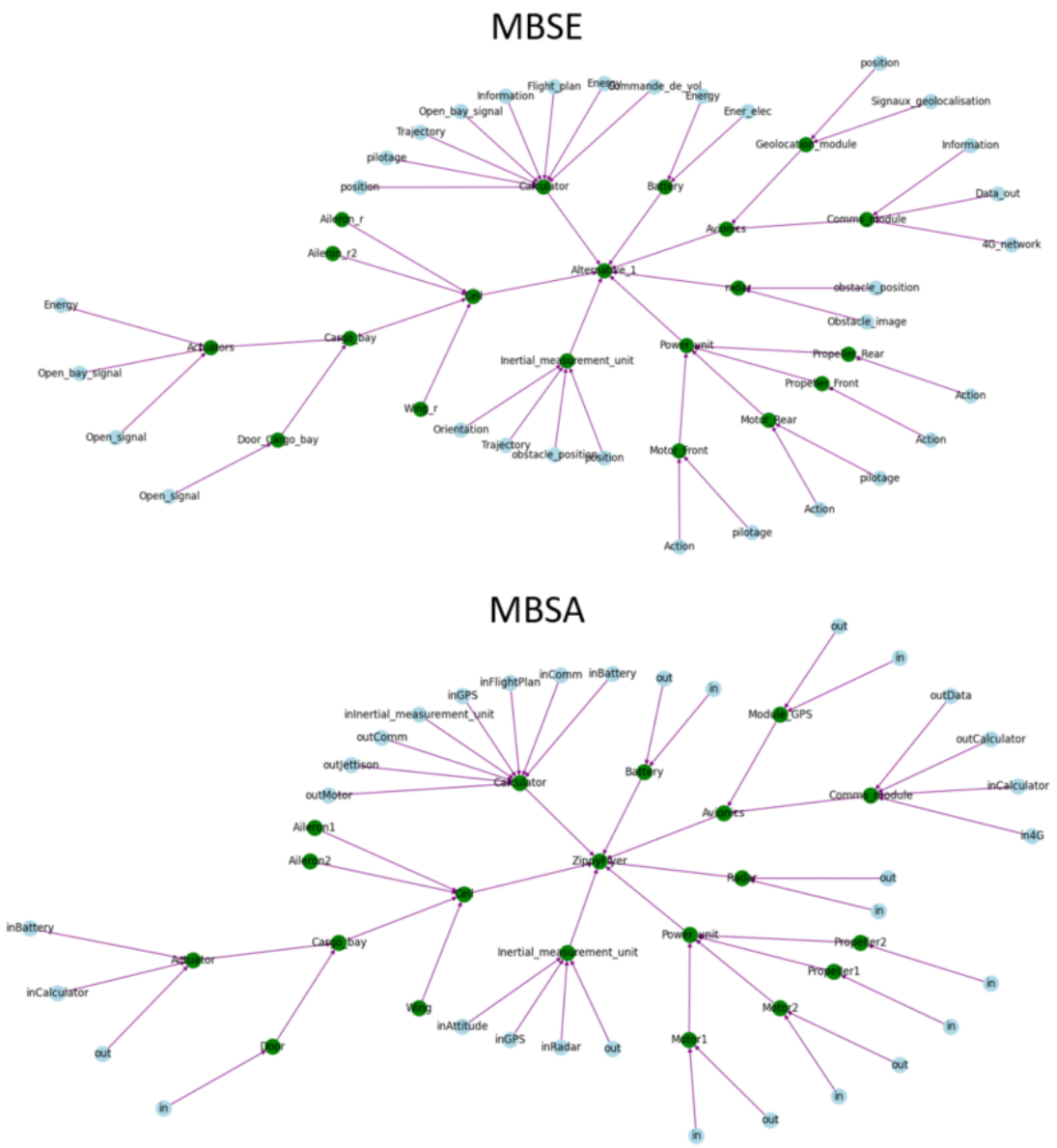


Figure 5.12: Diagrams for the A' and B' S2ML models of the fixed-wing drone at the final comparison step.

a new point of view. In light of this new point of view, we get a new comprehension of some of the characteristics of the SmartSync methodologies. This framework also allows us to formally show which exact characteristics of the models are compared through the synchronization methodology. We also show some current limitations such as not taking the connections into consideration, or the inability to consider objects that are at a different level of abstraction, and we can give ideas for improvement. In this section, we discuss how the S2ML+Cat framework helps us understand the topics of pragmatics, composition, and connections in the context of SmartSync.

5.3.1 Pragmatics

In SmartSync, and more generally in the methodologies we presented in section 1.3.4, the user is required to align the model elements manually. Although this seems like a bothersome and time-consuming step, we believe it is almost impossible to do otherwise in general. Even though models have syntax and semantics that allow them to be interpreted by a computer, they also carry a meaning that only makes sense to humans. This meaning is called pragmatics. Pragmatic models, such as the MBSE models, carry a meaning that is lost if model elements are replaced with abstract names, such as described in [100]. The interested reader may find more information on pragmatics in the book [99].

The SmartSync tool deals with the pragmatic aspect of models by creating a form of a dictionary – the CSV file. In the general definition of consistency that we have given in this paper, we do not force the consistency relation to consider pragmatics. Nevertheless, SmartSync still deals with pragmatics, so we should explain how that makes sense in the S2ML+Cat Framework.

One way to include pragmatics in the S2ML+Cat framework is the idea that we should build the biggest common skeleton for both models that respects the user's dictionary. This can be achieved by constructing a Catmodel $Dictionary$ that represents the dictionary. We shall then define functors towards this Catmodel from both A_{comp} and B_{comp}, and then, building the pullback of A_{comp} and B_{comp} will allow us to represent the alignment of objects by the user. Therefore, as depicted in Fig. 5.13, the A' and B' categories would be the pullback of A_{comp} and B_{comp}. F_{2A} and F_{2B} are the injections from $A' \equiv B'$ to A_{comp} and B_{comp} respectively, in the pullback.

We need to define $Dictionary$ precisely if we want to operate the computation of the pullback or even prove that it always exists. An idea for that would be to build it from A_{comp} and B_{comp} with these constraints:

- $Ob(Dictionary)$ is a set of Catblocks, Catports and Catconnections, such as we have $p_{ob} : Ob(A_{comp}) \longrightarrow Ob(Dictionary)$, $q_{ob} : Ob(B_{comp}) \longrightarrow Ob(Dictionary)$ such as for each $x \in Ob(A_{comp})$ and $y \in Ob(B_{comp})$, we have $p(x) = q(y)$ if and only if the elements have been aligned by the user.

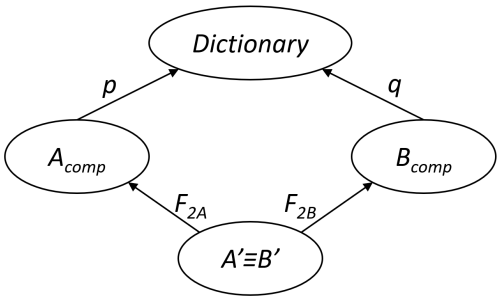


Figure 5.13: The pullback of A_{comp} and B_{comp} with regards to the dictionary

- p_{ob} and q_{ob} are injective applications
- We give $Dictionary$ the form of a model by building the morphisms needed to have images of the morphisms of A_{comp} and B_{comp} when we enrich p and q to make them functors.

We can make it so that this category is unique by choosing the symbols for the ports judiciously. However, it is unimportant, as any such category would suffice to represent that we aligned objects of A_{comp} and B_{comp}.

Although we do not care for the category structure in the dictionary, we can build it. The components over the objects of the p and q functors carry the essential information, i.e., the alignment of model elements.

5.3.2 Composition

The aspect of composition in category theory is of high interest in the concepts that we define.

The morphisms in the categories of S2ML+Cat define relations between the model elements to allow us both to apprehend the direct and composed relations. In [12], the definition of S2ML does provide a composition relation that only links an element and its parent. In S2ML+Cat, however, the belonging morphisms define the transitive closure of this relation. This allows us to identify all the ancestors of an object.

It is not yet considered in the methodology that we have tested with the case study. However, this means that the binary consistency relation that we define can encompass cases where a model element is at a certain level of abstraction in one model and another level of abstraction in the second model. A simple example of such a difference is if we represent a gear motor in the system. We can, in one model, have a gear motor block that contains a motor block and a gear block, and, in the other model, directly represent the motor and gear blocks without the gear motor level.

This would result in the categories shown in Fig. 5.14. These diagrams identify the direct belongings with the black arrows and their composition to

the dotted arrows. We can easily understand that the dotted arrows in the first case correspond to the black ones in the second; therefore, these models can be deemed consistent.

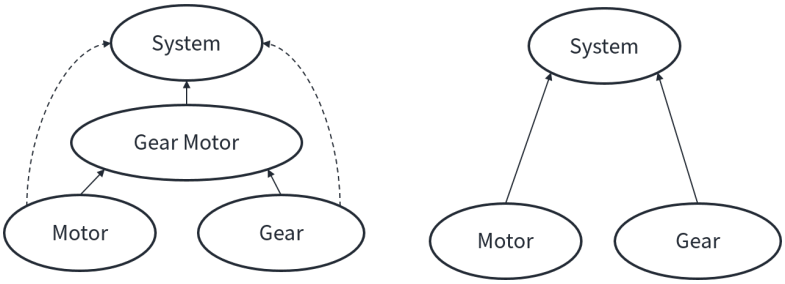


Figure 5.14: The categories for both possible level of abstraction of the gear motor

The second kind of relations that we have in the models are those between the ports and the connections they participate in. Because of how we defined the morphisms between ports and connections, and thanks to composition, we obtain morphisms between ports that participate in the same connection, as shown in Fig. 4.7.

We also obtain morphisms between indirectly connected ports, i.e., in the case where we have three ports, with two connections between them. Such a structure will lead to morphisms between the two ports that are not part of the same connections. This allows us to apprehend indirect connections between ports. Thus we could consider the case in which one connection on one side corresponds to multiple ones on the other side.

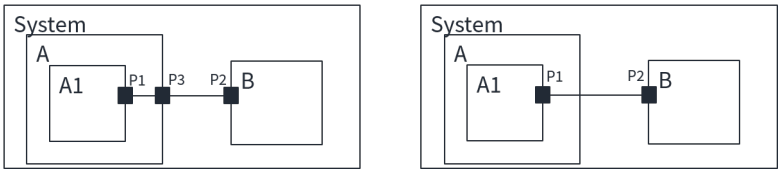


Figure 5.15: An example of block diagrams for a system with composed connections (left) that could be consistent with one connection (right)

An example of such a case is given in Fig. 5.15. Some system engineers consider the representation on the left best practice: a connection should not cross the frontier of a block; therefore, there should be intermediary ports when crossing to a different abstraction level. In a model dedicated to computation, such as the MBSA model, doing this is a problem because it increases the amount of computation to execute the system. Thus the representation on the left also makes sense to represent the system.

The categories for the two blocks diagrams from Fig. 5.15 can be found in Fig. 5.16. Thanks to the composition detailed above, we obtain the dotted ar-

rows that can be aligned in model comparison from the categorical point of view.

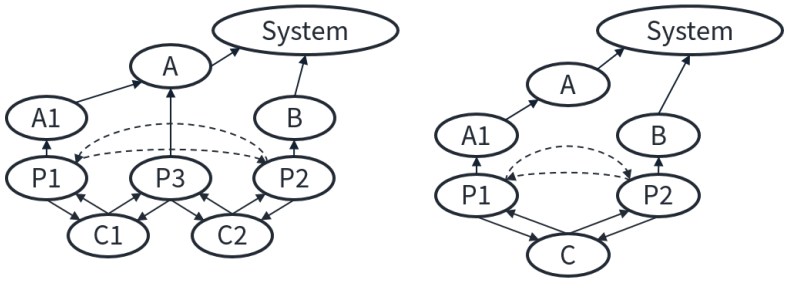


Figure 5.16: Diagrams of the categories for the block diagrams from Fig. 5.15

5.3.3 Connections

We discussed in section 5.3.2 the fact that composition allows for better consideration of the connection. SmartSync does not yet consider connections, so we have not demonstrated this in the case study.

It is interesting to note that it should theoretically not be difficult to add support of connections. The ports are already aligned in SmartSync. We could consider the Catports of a Catconnection in the Catmodel A and verify if their counterparts in the Catmodel B also are connected.

5.3.4 Connections and tuples

In section 4, we defined the connections as sets of ports. Although this definition is consistent with the quintuplet definition of S2ML models from [12], the reader could argue that the order of ports in a connection can be significant, for example, if there is a direction to the connection. This could be considered in the quintuplet definition using tuples rather than sets to define the connections.

In the case of the categorical definition of S2ML models, we could consider this property by defining connections as categories, with ports being the objects of the category and morphisms from a port A to a port B if the port A precedes the port B in the connection.

CONCLUSIONS AND PERSPECTIVES

In the context of system complexification, the way systems are modeled has to be rethought to consider this complexity. In the systems engineering domain, the classical document-based approaches are meant to be replaced by the Model-Based System Engineering (MBSE) approach. Similarly, the traditional approaches for safety assessment, such as Fault Tree Assessment (FTA), shall be replaced by Model-Based Safety Assessment (MBSA). MBSA provides models closer to the architecture and can account for complex phenomena such as reconfigurations.

Because the MBSA model aims to assert that the system is safe, this model must represent the right architecture. Therefore consistency assessment between the MBSE and MBSA models is a significant challenge.

In this work, we first wanted to understand the critical distinction between differences and inconsistencies. Therefore we decided to study the differences between MBSE and MBSA models through the consistency assessment of a case study: the landing gear. This led us to establish a typology where we identified three different kinds of differences related to their origin:

- Differences due to modeling tools and practices;
- Differences due to modeling intend;
- Differences due to modeling errors.

Out of these differences, only the latest has to be eliminated. The others are important to the models as they allow them to represent different behaviors toward different aims.

To tackle the challenge of consistency, researchers designed methodologies called model synchronization. They allow us to compare models and eliminate inconsistencies through pivot models – translations from the MBSE and MBSA models – written in a common formalism. This thesis focused on the SmartSync methodology, which uses the S2ML modeling language as its common

formalism. Through the comparison of the models of the landing gear using SmartSync, we show the importance of consistency assessment. Even though the methodology allows us to detect most inconsistencies between the models, it still has some limitations. We believe a formal frame around model synchronization could help with these limitations.

Thus we propose S2ML+Cat, a mathematical framework based on category theory. In this framework, we define a way to represent any S2ML model with category theory, and, by extension, any structural model. We call S2ML+Cat the category with these models as objects and injections between the models as morphisms. We specify injections, which are relations between the models that express that one model contains the other.

We have a need to assert consistency between models, SmartSync, along with S2ML, answers this need. Through category theory, we interpret the S2ML concepts that are relevant to model consistency. This is a new perspective over SmartSync, i.e. an interpretation within category theory, that is equivalent to the original definition proposed by SmartSync's authors. With this new point of view we apply a mathematical theory that we wanted to study on a real world application. We also intend to provide a definition of consistency that we hope can be generalised in the future, to methodologies other that SmartSync.

Within S2ML+Cat, we define what we call a binary consistency relation. This relation is defined by the existence of a certain structure between two models. This structure contains comparison models, the models where we eliminated any element that we do not intend to compare, and compared models, which are models where only the common skeleton of both models remains. This definition is very general, the idea is that we could define diverse consistency relations using it; in this work we only use it to work with the SmartSync methodology.

We provide a second case study, the blood delivery drone, which we use to apply S2ML+Cat to the SmartSync methodology. Based on this study, we can suggest that, except for the abstraction limitations we detected in SmartSync, in the general case, SmartSync builds the biggest common skeleton to two models. This common skeleton includes all model elements except for the connections. We also give leads to how the abstraction limitations can be overcome and how connections could be taken into account in further versions of SmartSync. We believe that further work to integrate S2ML+Cat in SmartSync would allow for improvements to the methodology.

This thesis is entitled "Category theory for consistency between multilevel system modeling (MBSE) and safety (MBSA)". However, we have shown that we can apply our work to Modelica and SCOLA models. Most models include some representation of the system's architecture, and any such model can be translated to a structural language such as S2ML. Therefore we argue that our work can be used to compare most models that would share a common architecture. In the case of MBSE and MBSA, that means that although the examples

chosen in this memoir are written using SysML and AltaRica 3.0, other MBSE and MBSA languages, such as Capella, Cecilia OCAS, SIMFIA Neo, HipHOPPS, Figaro, and others, could be easily compatible. This means that most scientists that need to compare the system architecture contained in heterogeneous models should be able to use a synchronization methodology and give it mathematical foundations using S2ML+Cat.

We also note that S2ML+Cat allows comparing the results of diverse synchronization methodologies, as we can compare which inconsistencies will be detected. It might also be possible to use S2ML+Cat to merge synchronization methodologies. Along with the S2ML+Cat category and definition of models, we provide the mathematical proofs required to show that S2ML models can be specified with no ambiguity as categories and that the framework is mathematically sound. In this work, we only considered S2ML+Cat through the scope of the SmartSync methodology. However, we believe that further work could show that S2ML+Cat is a sound framework to consider consistency, and therefore it could support other methodologies. In this case, this framework is not intended to be visible to the end user; it rather would be a means for the researcher to show that a methodology is robust and to prove that it will give the expected comparison results. Therefore, the framework would benefit from being applied to an industrial example and other methodologies, such as the Consistency Links, that were developed in the S2C (System to Safety Continuity) project.

Tools are sometimes uncorrelated from their underlying theoretical frameworks. Any informatic tool uses an algorithm, and algorithms rely on theoretical frameworks. Thus, tools must be associated with theories. With S2ML+Cat, we give a mathematical ground to prove the results of SmartSync. This mathematical framework allows us to show the correctness of the results formally. An important perspective of this thesis would be to show that S2ML+Cat can be applied to other methodologies and be a framework for model consistency in general.

We believe that a significant perspective of this mathematical framework is to work on the idea of simplifying the comparison in iterations of the models. Because we cannot entirely automatize the comparison, it can be a very time-consuming process, as engineers have to operate the alignment of model elements by hand. The MBSE and MBSA models (generally any models) are meant to evolve during the design process. A first version of the system is made, safety assessment over this version is operated, and based on the results, or other factors, a new architecture will be created, and its safety will be assessed. This means that successive versions of the models are to be compared. An important part of these new versions will probably be identical to the original models and, therefore, should not need a new consistency assessment. If we can identify this part through versioning, we could reduce the time required for the second – and further – iterations of consistency assessment. We believe

that this could be achieved through minor modifications to the way we represent S2ML models to allow the suppression of model elements.

BIBLIOGRAPHY

[1] ISO/IEC/IEEE International Standard – Systems Engineering – System Life Cycle Processes. *ISO/IEC 15288 First edition 2002-11-01*, pages 1–70, 2002.

[2] Nourhene Abdeljabbar, Faïda Mhenni, and Jean Yves Choley. A categorical framework for Safety critical mechatronic systems modeling. *2020 21st International Conference on Research and Education in Mechatronics, REM 2020*, 2020.

[3] Nourhene Abdeljabbar, Faïda Mhenni, and Jean Yves Choley. A Categorical Framework for Collaborative Design of Safety Critical Mechatronic Systems. *ISSE 2021 - 7th IEEE International Symposium on Systems Engineering, Proceedings*, 2021.

[4] Evan Ackerman and Michael Koziol. The blood is here: Zipline's medical delivery drones are changing the game in Rwanda. *IEEE Spectrum*, 56(5):24–31, 2019.

[5] R Adcock, B Wells, S Jackson, J Singer, and D Hybertson. Guide to the Systems Engineering Body of Knowledge (SEBoK), version 2.5. (October), 2021.

[6] AFNOR. EN 50129 Applications ferroviaires - Systèmes de signalisation, de télécommunications et de traitement - Systèmes électroniques de sécurité pour la signalisation, 2010.

[7] Aiste Aleksandraviciene and Aurelijus Morkevicius. *Magic Grid Book of Knowledge : A Practical Guide to System Modeling using MagicGrid from No Magic*. Vitae Litera, no magic edition, 2018.

[8] US Army. MIL-STD-1629A Procedures for performing a failure mode, effects, and criticality analysis, 1980.

[9] Modelica Association. Modelica ® A Unified Object-Oriented Language for Systems Modeling Language Specification, 2022.

[10] John Baez. Category Theory Course. 2019.

[11] Maria Bartnes. Safety vs. Security? *Proceedings of the Eighth International Conference on Probabilistic Safety Assessment & Management (PSAM)*, (May 2006):1202–1210, 2010.

[12] Michel Batteux, Jean-yves Choley, Faïda Mhenni, Luca Palladino, Tatiana Prosvirnova, Antoine Rauzy, and Maurice Theobald. Synchronization of system architecture, multi-physics and safety models. 2019.

[13] Michel Batteux, Jean-yves Choley, Faïda Mhenni, Tatiana Prosvirnova, and Antoine Rauzy. Synchronization of System Architecture and Safety Models : a Proof of Concept. 2019.

[14] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. System Structure Modeling Language (S2ML). 2015.

[15] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. AltaRica 3.0 Language Specification. page 126, 2017.

[16] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. AltaRica 3 . 0 in 10 Modeling Patterns. *International Journal of Critical Computer Based Systems (IJCCBS)*., 2018.

[17] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. From Models of Structures to Structures of Models. In *4th IEEE International Symposium on Systems Engineering, ISSE 2018 - Proceedings*, 2018.

[18] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. Model synchronization: A formal framework for the management of heterogeneous models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11842 LNCS:157–172, 2019.

[19] Ron Bell. Introduction to IEC 61508. *Conferences in Research and Practice in Information Technology Series*, 55:3–12, 2005.

[20] Aroua Berriche, Faïda Mhenni, Abdelfattah Mlika, and Jean-Yves Choley. Towards Model Synchronization in Model Driven Engineering of Mechatronic Systems. 2019.

[21] JP Blanquart and JM Astruc. Criticality categories across safety standards in different domains. *Erts-2012, ...*, (6), 2012.

[22] Simon Bliudze, Sébastien Furic, Joseph Sifakis, and Antoine Viel. Rigorous design of cyber-physical systems: Linking physicality and computation. *Software and Systems Modeling*, 18(3):1613–1636, 2019.

[23] B W Boehm. Verifying and Validating Software Requirements and Design Specifications. *IEEE Software*, 1(1):75–88, 1984.

[24] Frédéric Boniol and Virginie Wiels. The Landing Gear System Case Study. In Frédéric Boniol, Virginie Wiels, Yamine Ait Ameur, and Klaus-Dieter Schewe, editors, *ABZ 2014: The Landing Gear Case Study*, pages 1–18, Cham, 2014. Springer International Publishing.

[25] M. Bouissou, H. Bouhadana, M. Bannelier, and N. Villatte. Knowledge Modelling and Reliability Processing: Presentation of the Figaro Language and Associated Tools. *IFAC Proceedings Volumes*, 24(13):69–75, 1991.

[26] S. J. Brown. Overview of IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. *Nuclear Engineer*, 42(2):39–44, 2001.

[27] Olivia Caramello. The unification of Mathematics via Topos Theory. pages 1–42, 2010.

[28] Olivia Caramello. *Topos-theoretic background*. 2014.

[29] Kester Clegg, David Stamp, and John McDermid. Binding Fault Logic to System Design: A SysML Approach. (July):880–887, 2021.

[30] Collectif AFIS. Conception des architectures: Chapitre 8, le système est le résultat d'une conception. In *DÉCOUVRIR ET COMPRENDRE L'INGÉNIERIE SYSTÈME*. Collection AFIS, 2012.

[31] Edward Crawley, O Weck de, Steven Eppinger, Christopher Magee, Joel Moses, Warren Seering, Joel Schindall, David Wallace, Daniel Whitney, and Olivier De Weck. The Influence of Architecture in Engineering Systems. *Engineering Systems Monograph*, pages 1–30, 2004.

[32] Romaric Demachy and Sébastien Guilmeau. Short paper - Structural consistency of MBSE and MBSA models using Consistency Links. pages 1–4.

[33] Department of Defense (DoD). The DoDAF Architecture Framework Version 2 . 02. *DoDAF Journal*, page 289, 2011.

[34] T Doran. IEEE 1220: for practical systems engineering. *Computer*, 39(5):92–94, 2006.

[35] Joanne Bechta Dugan, Salvatore J. Bavuso, and Mark A. Boyd. Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems. *IEEE Transactions on Reliability*, 41(3):363–377, 1992.

[36] Julien Le Duigou, Vincent Chapurlat, and Jean-Luc Garnier. AFIS Academy-Industry Forum 2020 in Compiègne. *INSIGHT*, 24(4):9–11, 2021.

[37] Andrée Ehresmann. Genèse de l'approche catégorique des systèmes évolutifs à mémoire (MES). 2015.

[38] Andrée C. Ehresmann. MENS, an info-computational model for (Neuro-)Cognitive systems capable of creativity. *Entropy*, 14(9):1703–1716, 2012.

[39] Samuel Eilenberg and Saunders Mac Lane. General theory of natural equivalences. *Transactions of the American Mathematical Society*, 58(2):231–194, 1945.

[40] Schneider Electric. La Sûreté de Fonctionnement (SdF). 2004.

[41] Dominique Ernadote. *MB 2 SE : A Theoretical Foundation for Systems Engineering - Une fondation theorique pour l'ingenierie systeme*. PhD thesis, 2020.

[42] J. A. Estefan. Survey of Model-Based Systems Engineering (MBSE). *Jet Propulsion*, 2008.

[43] European Cooperation for Space Standardization ECSS. ECSS-Q-ST-30C Rev.1 – Dependability, 2017.

[44] European Cooperation for Space Standardization ECSS. ECSS-Q-ST-40C Rev.1 – Safety, 2017.

[45] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 717 LNCS(X):84–99, 1993.

[46] Donald G. Firesmith. Common Concepts Underlying Safety Security and Survivability Engineering. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2003.

[47] Robert W. Floyd. Assigning meanings to programs. pages 19–32, 1967.

[48] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML*. 2008.

[49] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML - The Systems Modeling Language*. Elsevier, 2015.

[50] Paul A Gagniuc. *Historical Notes*, chapter 1, pages 1–9. John Wiley and Sons, Ltd, 2017.

[51] Government Electronics and Information Technology Association (GEIA) and Electronic Industries Alliance (EIA). EIA-632 Processes For Engineering a System, 2003.

[52] Alexandre Grothendieck. Sur quelques points d'algébre homologique. *Tohoku Mathematical Journal*, 9(2):119–183, 1957.

[53] Matthias Güdemann and Frank Ortmeier. A framework for qualitative and quantitative formal model-based safety analysis. *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, pages 132–141, 2010.

[54] Christophe Guychard, Sylvain Guerin, Ali Koudri, Antoine Beugnard, and Fabien Dagnat. Conceptual interoperability through Models Federation. (OCTOBER):1–22, 2013.

[55] Aric Hagberg, Dan Schult, and Pieter Swart. NetworkX Reference (Python). *Python package*, page 464, 2011.

[56] David Harel. Statecharts: a visual complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[57] David Harel and Amnon Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[58] David Harel and Bernhard Rumpe. Meaningful modeling: What's the semantics of "semantics"? *Computer*, 37(10):64–72, 2004.

[59] Myron Hecht and David Baum. Use of SysML for the creation of FMEAs for Reliability, Safety, and Cybersecurity for Critical Infrastructure. *INCOSE International Symposium*, 29(1):145–158, 2019.

[60] Hans-Peter Hoffmann. Systems engineering best practices with the Rational Workbench for Systems and Software Engineering. Technical report, IBM Corporation, 2010.

[61] INCOSE. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 4th Edition*. Wiley, 2015.

[62] SAE International. Guidelines and methods for conducting the safety assesment process on civil airborne systems and equipment ARP4761. Technical report, SAE International, 1996.

[63] International Council on Systems Engineering INCOSE. *Systems Engineering Handbook: A "what To" Guide for All SE Practitioners*. 2004.

[64] International Council on Systems Engineering INCOSE. *Guide to the Systems Engineering Body of Knowledge v1.4*. 2012.

[65] International Electrotechnical Commision IEC. IEC 61226:2020 Nuclear power plants - Instrumentation, control and electrical power systems important to safety - Categorization of functions and classification of systems, 2009.

[66] International Electrotechnical Commision IEC. IEC/TR61838 Nuclear power plants - Instrumentation and control important to safety - Use of probabilistic safety assessment for the classification of functions. Technical report, 2009.

[67] International Electrotechnical Commision IEC. IEC 60300-3-4, 2022.

[68] International Organization for Standardization ISO. ISO 26262 Road vehicles – Functional safety, 2011.

[69] International Organization for Standardization ISO. ISO/IEC WD1 42020 – Architecture processes. (1), 2015.

[70] International Organization for Standardization ISO. ISO/IEC/IEEE 15288 - International Standard - Systems and software engineering - System life cycle processes. *ISO/IEC/IEEE 15288 First edition 2015-05-15*, 2015.

[71] International Organization for Standardization ISO and International Electrotechnical Commision IEC. ISO/IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related, 1990.

[72] David E. Keyes, Lois C. McInnes, Carol Woodward, William Gropp, Eric Myra, Michael Pernice, John Bell, Jed Brown, Alain Clo, Jeffrey Connors, Emil Constantinescu, Don Estep, Kate Evans, Charbel Farhat, Ammar Hakim, Glenn Hammond, Glen Hansen, Judith Hill, Tobin Isaac, Xiangmin Jiao, Kirk Jordan, Dinesh Kaushik, Efthimios Kaxiras, Alice Koniges, Kihwan Lee, Aaron Lott, Qiming Lu, John Magerlein, Reed Maxwell, Michael McCourt, Miriam Mehl, Roger Pawlowski, Amanda P. Randles, Daniel Reynolds, Beatrice Rivière, Ulrich Rüde, Tim Scheibe, John Shadid, Brendan Sheehan, Mark Shephard, Andrew Siegel, Barry

Smith, Xianzhu Tang, Cian Wilson, and Barbara Wohlmuth. Multiphysics simulations: Challenges and opportunities. *International Journal of High Performance Computing Applications*, 27(1):4–83, 2013.

[73] Daniel Krob. CESAM: CESAMES Systems Architecting Method A Pocket Guide. (January), 2017.

[74] Karl E. Kurbel. The Making of Information Systems. In *The Making of Information Systems*, 2008.

[75] Kimberly Lai, Thomas Robert, David Shindman, and Alison Olechowski. Integrating Safety Analysis into Model-Based Systems Engineering for Aircraft Systems: A Literature Review and Methodology Proposal. *INCOSE International Symposium*, 31(1):988–1003, 2021.

[76] Jean-CLaude Laprie and Jean Arlat. *Guide de la sûreté de fonctionnement*. Cépaduès-Editions, 1995.

[77] F. William Lawvere and Stephen F. Schanuel. *Conceptual Mathematics : a first introduction to categories*. 1997.

[78] Yung-Tsun Lee. Information Modeling: From Design to Implementation. 1999.

[79] Anthony Legendre. *Ingénierie système et Sûreté de fonctionnement : Méthodologie de synchronisation des modèles d'architecture et d ' analyse de risques*. PhD thesis, 2018.

[80] Oleg Lisagor, Tim Kelly, and Ru Niu. Model-based safety assessment: Review of the discipline and its challenges. *ICRMS'2011 - Safety First, Reliability Primary: Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*, pages 625–632, 2011.

[81] Saunders Mac Lane. *Categories for the Working Mathematician*. 1978.

[82] Saunders Mac Lane. Perspective The PNAS way back then. *Proceedings of the National Academy of Sciences of the United States of America*, 94(June):5983–5985, 1997.

[83] J N Martin. Overview of the EIA 632 standard: processes for engineering a system. In *17th DASC. AIAA/IEEE/SAE. Digital Avionics Systems Conference. Proceedings (Cat. No.98CH36267)*, volume 1, pages B32–1, 1998.

[84] Mathworks. Matlab & Simulink, Simulink Reference. Technical report, 2022.

[85] Meine Meulen. *Definitions for Hardware and Software Safety Engineers*. Springer London, 2000.

[86] Faïda Mhenni. *Safety analysis integration in a systems engineering approach for mechatronic systems design*. PhD thesis, Ecole Centrale Paris, 2014.

[87] Faïda Mhenni, Jean-Yves Choley, Nga Nguyen, and Christophe Frazza. Flight Control System Modeling with SysML to Support Validation, Qualification and Certification. *IFAC-PapersOnLine*, 49(3):453–458, 2016.

[88] Faïda Mhenni, Jean-Yves Choley, Olivia Penas, Régis Plateaux, and Moncef Hammadi. A SysML-based methodology for mechatronic systems architectural design. *Advanced Engineering Informatics*, 28(3):218–231, 2014.

[89] Bartosz Milewski. Category Theory for Programmers. page 467, 2019.

[90] OMG. OMG Unified Modeling Language, Version 2.5.1. Technical report, OMG, 2017.

[91] OMG. *OMG Systems Modeling Language (OMG SysML™)*. OMG, oct 2018.

[92] Claire Pagetti. Module de sûreté de fonctionnement, 2012.

[93] Yiannis Papadopoulos and John A. McDermid. Hierarchically performed hazard origin and propagation studies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1698(September 1999):139–152, 1999.

[94] C A Petri. *Kommunikation mit Automaten*. Schriften des Rheinisch-Westfälischen Institutes für Instrumentelle Mathematik an der Universität Bonn. Rheinisch-Westfälisches Institut f. instrumentelle Mathematik an d. Univ., 1962.

[95] Tatiana Prosvirnova, Michel Batteux, and Antoine Rauzy. SmartSync platform v0.0.1. Technical report, AltaRica Association, 2020.

[96] Alain Prouté. Introduction à la Logique Catégorique. pages 1–416, 2010.

[97] Marvin Rausand and Arnljot Høyland. *System reliability theory: models, statistical methods, and applications*. Wiley-Interscience, Hoboken, NJ, 2004.

[98] Antoine Rauzy. Guarded transition systems: A new states/events formalism for reliability studies. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 222(4):495–505, 2008.

[99] Antoine Rauzy. *Model-Based Reliability Engineering*. 2022.

[100] Antoine Rauzy and Cecilia Haskins. Foundations for model-based systems engineering and model-based safety assessment. *Systems Engineering*, 22(2):146–155, 2019.

[101] Emily Riehl. *Category Theory in Context*. 2016.

[102] Pascal Roques. MBSE with the ARCADIA Method and the Capella Tool. *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, page 11, 2016.

[103] Winston Royce. Managing the development of large software systems: concepts and techniques. In *ICSE '87*, 1987.

[104] RTCA. DO-178B Software Considerations in Airborne Systems and Equipment Certification, 1992.

[105] Andrew P Sage and Steven M Biemer. Processes for System Family Architecting, Design, and Integration. *IEEE Systems Journal*, 1(1):5–16, 2007.

[106] Pierre De Saqui-sannes, Ludovic Apvrille, Rob Vingerhoeds, Pierre De Saqui-sannes, Ludovic Apvrille, Rob Vingerhoeds, Checking Sysml, and Models Against. Checking SysML Models Against Safety and Security Properties To cite this version : HAL Id : hal-03423073 Checking SysML Models against Safety and Security Properties. 2021.

[107] Kenneth J Schlager. Systenas Engineering-Key to Modern Development. *Ire Transactions on Engineering Management*, pages 1953–1955, 1953.

[108] Patrick Schultz, David I. Spivak, and Christina Vasilakopoulou. Dynamical Systems and Sheaves. *Applied Categorical Structures*, 28(1):1–57, 2020.

[109] D.J. Sherwin and A. Bossche. *The reliability, availability and productiveness of systems*. Springer Dordrecht, 1993.

[110] Ankit Singh and Siemens Healthineers. Rtca do-178b (eurocae ed-12b). (March), 2017.

[111] David I Spivak. *Category theory for the sciences*. The MIT Press, 2014.

[112] Duo Su, Chang Feng, Qi Gong, and Yan Li. Application and research on model-based safety analysis. *Proceedings of 2015 the 1st International Conference on Reliability Systems Engineering, ICRSE 2015*, 2015.

[113] US Army. MIL-P-1629 Procedure for Performing a Failure Mode Effect and Criticality Analysis, 1949.

[114] Alain Villemeur. *Sûreté de fonctionnement des systèmes industriels : fiabilité, facteurs humains, informatisation*. Collection de la Direction des études et recherches d'Electricité de France 67. 1988.

[115] Paul Wach and Alejandro Salado. The Need for Semantic Extension of SysML to Model the Problem Space. *Recent Trends and Advances in Model Based Systems Engineering*, (October):279–289, 2020.

[116] H.A. Watson. Launch control safety study. Technical report, Murray Hill, NJ, USA, 1961.

[117] Tim Weilkiens, Jesko G. Lamm, Stephan Roth, and Markus Walker. *B: The V-Model*, pages 343–352. John Wiley and Sons, Ltd, 2015.

[118] Liudong Xing and Suprasad V Amari. *Fault Tree Analysis*, pages 595–620. Springer London, London, 2008.

APPENDIX A

PUBLICATIONS

Four conference publications and one journal article were written during this thesis.

LambdaMu22: Vidalie, J., Batteux, M., Choley, J.-Y., Mhenni, F., Kendel, M.-S. *Typology of the differences Between Model-Based System Engineering (MBSE) and Safety Assessment (MBSA) models: Analysis of a Reference System.* Congrès Lambda Mu 22, Institut pour la Maîtrise des Risques, Oct 2020, Le Havre (e-congrès), France. ⟨hal-03453551⟩

IEEE ISSE 2021: Vidalie, J., Kendel, M.-S., Mhenni, F., Batteux, M., Choley, J.-Y., *State Machines Consistency between Model Based System Engineering and Safety Assessment Models*, 2021 IEEE International Symposium on Systems Engineering (ISSE), 2021, pp. 1-8, doi: 10.1109/ISSE51541.2021.9582470.

The LambdaMu22 and IEEE ISSE 2021 publications can be identified in work presented in Chapter 2 and Chapter 3 of the thesis. They respectively present a typology of the differences between MBSE and MBSA models and the study of structural consistency between MBSE and MBSA state machines. Both of these studies use the Landing Gear case study that we present in Chapter 2.

ESREL 2021: Vidalie, J., Batteux, M., Mhenni, F., Choley, J.-Y. *Model-Based Safety Assessment of an Insulin Pump System with AltaRica 3.0.* Proceedings of the 31st European Safety and Reliability Conference (ESREL), Sep 2021, Angers, France. ⟨hal-03429161⟩

We do not present the work from the ESREL 2021 paper in this thesis. This paper presents the MBSA modeling and safety assessment of an insulin pump. This study is relevant to the model comparison between MBSE and MBSA since it concerns a safety-critical system. However, because we already present the Landing Gear and Blood Delivery Drone case studys, we believe it would be redundant to add a third example to this memoir.

MDPI Applied-Science: Vidalie, J., Batteux, M., Mhenni, F., Choley, J.-Y. *Category Theory Framework for System Engineering and Safety Assessment Model Synchronization Methodologies.* Applied Sciences. 2022; 12(12):5880. https://doi.org/10.3390/app12125880

The MDPI Applied-Science publication corresponds to the work presented in Chapter 4 and the case study in Chapter 5. In this journal article, we introduce the S2ML+Cat framework and apply it to synchronizing the models of a blood delivery drone with the SmartSync methodology.

IEEE ISSE 2022: Vidalie, J., Bouhali, I., Mhenni, F., Batteux, M., Choley, J.-Y. *Consistency of multiple system engineering models of a fixed wing drone* (ISSE IEEE 2022)

The IEEE ISSE 2022 publication focuses on the comparison of the blood delivery drone models from Chapter 5.

This thesis was also the object of many internal and external presentations. Amongst these presentations, two short popular science presentations received awards:

- A "Ma thèse en 180 secondes" (My thesis in 180 seconds) presentation obtained the second price at the AFIS (Association Française de l'Ingénièrie Système) doctoral day in december 2020 [36];
- A "Ma thèse en 6 minutes" (My thesis in 6 minutes) presentation obtained the third price of the IRT SystemX PhD day in july 2022.

These successes are the result of a special interest for popular science and sharing my work outside the system engineering and safety communities.

APPENDIX B

LANDING GEAR MODELS

B.1 AltaRica 3.0 code for the landing gear

This section contains the code for the AltaRica 3.0 model of the landing gear. This code is composed of the main block that instantiates the three main parts of the landing gear: the pilot interface, the mecha-hydraulic part, and the digital part. The rest of the model comprises classes for each subsystem, subsubsystem, ..., and component. Each model element is instantiated as a block by its parent, from a class.

All system components are associated with a guarded transition system defined in the nrpState class.

```
block LandingSys
    PilotInterface pilotInterface;
    MechaHydraulicalPart mechahydraulicPart;
    DigitalPart digitalPart;
    assertion
        digitalPart.CPIOM1.input := pilotInterface.udHandle.output;
        digitalPart.CPIOM2.input := pilotInterface.udHandle.output;
        digitalPart.CPIOM1.analogicalSwitchFlag := pilotInterface.analogicalSwitch.output;
        digitalPart.CPIOM2.analogicalSwitchFlag := pilotInterface.analogicalSwitch.output;
        digitalPart.CPIOM1.gearLocked := mechahydraulicPart.gearLocked;
        digitalPart.CPIOM2.gearLocked := mechahydraulicPart.gearLocked;
        digitalPart.CPIOM1.doorLocked := mechahydraulicPart.doorLocked;
        digitalPart.CPIOM2.doorLocked := mechahydraulicPart.doorLocked;
        mechahydraulicPart.hydraulicSys.generalElectroValve.order :=
             digitalPart.CPIOM1.outputGeneralValve;
        mechahydraulicPart.hydraulicSys.doorHydraulicSys.order := digitalPart.CPIOM1.outputDoor;
        mechahydraulicPart.hydraulicSys.gearHydraulicSys.order := digitalPart.CPIOM1.outputGear;
end
-------------------------------------------------------------------------------------------------

// Pilot Interface : Cockpit interface for the pilot, that includes landing gear status indicator
     lights and the actuation handle.
-------------------------------------------------------------------------------------------------

domain nrpState {OK, KO}
class NonRepairableComponent
    nrpState s (init = OK);
    event failure (delay = Dirac(0));
    transition
```

```
      failure: s == OK -> s := KO;
end

class PilotInterface
   GearLight frontGearLight;
   GearLight leftGearLight;
   GearLight rightGearLight;
   UpDownHandle udHandle;
   AnalogicalSwitch analogicalSwitch;
end

class GearLight
   extends NonRepairableComponent;
   Boolean input, output (reset = true);
   assertion
      output := if s == OK then input else false;
end

class UpDownHandle
   extends NonRepairableComponent;
   Boolean output (reset = true);
   Boolean position (reset = true);
   assertion
      output := if s == OK then position else false;
end

class AnalogicalSwitch
   extends NonRepairableComponent;
   Boolean recentChange (reset = true);
   Boolean output (reset = true);
   assertion
      output := if s == OK then recentChange else false;
end
-------------------------------------------------------------------------------------------

//Mechanical & Hydraulic Parts : Mechanical systems allowing actuation of the landing gear, and
     hydraulic systems powering the mechanical part.
-------------------------------------------------------------------------------------------

class MechaHydraulicalPart
   LandingSet frontLandingSet;
   //LandingSet rightLandingSet;
   //LandingSet leftLandingSet;
   HydraulicSys hydraulicSys;
   Boolean gearLocked (reset = true);
   Boolean doorLocked (reset = true);
   assertion
      frontLandingSet.doorCylinder.input := if
           hydraulicSys.doorHydraulicSys.extensionDoorElectroValve.output then true else if
           hydraulicSys.doorHydraulicSys.retractionDoorElectroValve.output then true else false;
      //rightLandingSet.doorCylinder.input := if
           hydraulicSys.doorHydraulicSys.extensionDoorElectroValve.output then true else if
           hydraulicSys.doorHydraulicSys.retractionDoorElectroValve.output then true else false;
      //leftLandingSet.doorCylinder.input := if
           hydraulicSys.doorHydraulicSys.extensionDoorElectroValve.output then true else if
           hydraulicSys.doorHydraulicSys.retractionDoorElectroValve.output then true else false;
      frontLandingSet.gearCylinder.input := if
           hydraulicSys.gearHydraulicSys.extensionGearElectroValve.output then true else if
           hydraulicSys.gearHydraulicSys.retractionGearElectroValve.output then true else false;
      //rightLandingSet.gearCylinder.input := if
           hydraulicSys.gearHydraulicSys.extensionGearElectroValve.output then true else if
           hydraulicSys.gearHydraulicSys.retractionGearElectroValve.output then true else false;
      //leftLandingSet.gearCylinder.input := if
           hydraulicSys.gearHydraulicSys.extensionGearElectroValve.output then true else if
           hydraulicSys.gearHydraulicSys.retractionGearElectroValve.output then true else false;
      gearLocked := /*if frontLandingSet.gear.rightPos == rightLandingSet.gear.rightPos ==
           leftLandingSet.gear.rightPos then frontLandingSet.gear.rightPos else false*/
           frontLandingSet.gear.rightPos;
      doorLocked := /*if frontLandingSet.door.rightPos == rightLandingSet.door.rightPos ==
           leftLandingSet.door.rightPos then frontLandingSet.door.rightPos else false*/
           frontLandingSet.door.rightPos;
end
```

```
//LandingSet : une unite de train

class LandingSet
    UplockBox uplockBox;
    Door door;
    Cylinder doorCylinder;
    Cylinder gearCylinder;
    Gear gear;
    assertion
        door.input := doorCylinder.output;
        gear.input := gearCylinder.output;

end


class UplockBox
    extends NonRepairableComponent;
    Boolean rightpos, input (reset = true);
    assertion
        rightpos := if s == OK then input else false;
end

class Cylinder
    extends NonRepairableComponent;
    Boolean input (reset = true);
    Boolean output (reset = true);
    assertion
        output := if s == OK then input else false;
end

class Door
    extends NonRepairableComponent;
    Boolean input, rightPos (reset = true);
    assertion
        rightPos := if s == OK then input else false;
end

class Gear
    extends NonRepairableComponent;
    Boolean input, rightPos (reset = true);
    assertion
        rightPos := if s == OK then input else false;
end

//Hydraulic System : The hydraulic system powering the actuators of the landing gear system

class HydraulicSys
    PressureSensor pressureSensor;
    ElectroValve generalElectroValve;
    DoorHydraulicSys doorHydraulicSys;
    GearHydraulicSys gearHydraulicSys;
    assertion
        doorHydraulicSys.input := generalElectroValve.output;
        gearHydraulicSys.input := generalElectroValve.output;
end

domain pressureSensorState {OK, notOK}

class PressureSensor
    extends NonRepairableComponent;
    Boolean input, output (reset = true);
    assertion
        output := if s == OK then input else false;
end

//Door Hydraulic System : The part of the hydraulic systems powering the landing gear doors.

class DoorHydraulicSys
    Boolean input (reset = true);
    Boolean order (reset = true);
    ElectroValve retractionDoorElectroValve;
    ElectroValve extensionDoorElectroValve;
    assertion
```

```
        retractionDoorElectroValve.input := input;
        extensionDoorElectroValve.input := input;
        retractionDoorElectroValve.order := order;
        extensionDoorElectroValve.order := order;
end

//Gear Hydraulic System : The part of the hydraulic system powering the landing gear gears.

class GearHydraulicSys
    Boolean input (reset = true);
    Boolean order (reset = true);
    ElectroValve retractionGearElectroValve;
    ElectroValve extensionGearElectroValve;
    assertion
        retractionGearElectroValve.input := input;
        extensionGearElectroValve.input := input;
        retractionGearElectroValve.order := order;
        extensionGearElectroValve.order := order;
end


class ElectroValve
    extends NonRepairableComponent;
    Boolean input (reset = true);
    Boolean output (reset = true);
    Boolean order (reset = true);
    assertion
        output := if s == OK then input and order else false;
end
-------------------------------------------------------------------------------------------------

//Digital Part : Computing units of the landing gear system.
     -------------------------------------------------------------------------------------------------

class DigitalPart
    CPIOM CPIOM1;
    CPIOM CPIOM2;
end

class CPIOM
    extends NonRepairableComponent;
    Boolean analogicalSwitchFlag (reset = true);
    Boolean input (reset = true);
    Boolean outputGeneralValve (reset = true);
    Boolean outputGear (reset = true);
    Boolean outputDoor (reset = true);
    Boolean gearLocked (reset = true);
    Boolean doorLocked (reset = true);
    assertion
        outputGear := if analogicalSwitchFlag == true and s == OK then input else false;
        outputDoor := if analogicalSwitchFlag == true and s == OK then input else false;
        outputGeneralValve := if analogicalSwitchFlag == true and s == OK then input else false;
end
```

B.2 S2ML code for the MBSE model

This section contains the code for the S2ML translation of the MBSE model. We translated the model manually, although it would be possible to automatize this process, for example, through a java plugin for Cameo System Modeler, the modeling tool we used. This translation corresponds to the IBD and BDD diagrams. We translated Partproperties to blocks, ports to ports, and connections to connections.

```
block Landing_gear_systeme
-------------------------------------------------------------------------------------------------
```

```
   //DigitalPart : Digital part of the landing gear that controls the system
-------------------------------------------------------------------------------------------------------

   block Digital_part1
      block Monitoring
         port Lsig;
         port Dstat;
         port Gstat;
      end
      block Electrical_command
         port Electrical_Cmd;
         port E_Cmd_GEV;
         port E_Cmd_EGEV;
         port E_Cmd_RGEV;
         port E_Vmd_CdEV;
         port E_Cmd_OdEV;
      end
      port ElectricPower;
   end
   block Digital_part2
      block Monitoring
         port Lsig;
         port Dstat;
         port Gstat;
      end
      block Electrical_command
         port Electrical_Cmd;
         port E_Cmd_GEV;
         port E_Cmd_EGEV;
         port E_Cmd_RGEV;
         port E_Vmd_CdEV;
         port E_Cmd_OdEV;
      end
      port ElectricPower;
   end
-------------------------------------------------------------------------------------------------------

   //MechaHydraulicPart : Mechanical and Hydraulic parts of the landing gear system
-------------------------------------------------------------------------------------------------------

   block mecahydraulic_part
      block Right_landing_box
         block Right_Gear
            port Mech;
            port Status;
            port Mech_Ground;
         end
         block Right_door
            port Mech;
            port Status;
            port Mech_Aircraft;
         end
      end
      block Front_landing_box
         block Front_Door
            port Mech;
            port Status;
            port Mech_Aircraft;
         end
         block Front_Gear
            port Mech;
            port Status;
            port Mech_Ground;
         end
      end
      block Left_landing_box
         block Left_Door
            port Mech;
            port Status;
            port Mech_Aircraft;
         end
         block Left_Gear
```

```
            port Mech;
            port Status;
            port Mech_Ground;
        end
    end
    block Hydraulic_part
        block Right_Door_cylinder
            port HP_Close;
            port HP_Open;
            port Mech;
        end
        block Front_Door_cylinder
            port HP_Close;
            port HP_Open;
            port Mech;
        end
        block Left_door_cylinder
            port HP_Close;
            port HP_Open;
            port Mech;
        end
        block Open_door_Electro-Valve
            port HPin;
            port E_Cmd_OdEV;
            port HPout;
        end
        block Close_door_Electro-valve
            port HPin;
            port E_Vmd_CdEV;
            port HPout;
        end
        block General_electro_valves
            port Elec_Cmd;
            port Man_Cmd;
            port Hydraulic_power;
            port Hydraulic_Pwr_Out;
        end
        block Retract_Gear_Electro_valve
            port HPin;
            port E_Cmd_RGEV;
            port HPout;
        end
        block Extend_Gear_Electro-Valve
            port HPin;
            port E_Cmd_EGEV;
            port HPout;
        end
        block Right_Gear_Cylinder
            port HP_extend;
            port HP_retract;
            port Mech;
        end
        block Front_Gear_cylinder
            port HP_extend;
            port HP_retract;
            port Mech;
        end
        block Left_Gear_cylinder
            port HP_extend;
            port HP_retract;
            port Mech;
        end
    end
end
-------------------------------------------------------------------------------------------------------

    //PilotInterface : Interface between the pilot and the landing gear
-------------------------------------------------------------------------------------------------------

    block Pilot_interface
        block Handle
            port Manual_Cmd;
            port Man_Cmd;
```

```
        end
        block Gearlight1
            port Lsig;
        end
        block GearLight2
            port Lsig;
        end
        block Gearlight3
            port Lsig;
        end
    end
    port Manual_Cmd;
    port Hydraulic_power;
    port Electrical_Cmd;
    port ElectricPower;
    port Mech_Ground;
    port Mech_Aircraft;
end
```

B.3 S2ML code for the MBSA model

This section contains the code for the S2ML translation of the AltaRica 3.0 model. We translated the model manually, although it would be possible to automatize this process. We translated blocks to blocks and variables to ports. In this example, we ignored all model elements related to the guarded transition systems and not the architecture, as we were not interested in comparing these artifacts. We also ignored the assertions because the SmartSync methodology does not encompass them; this allows better readability of the S2ML model and has no impact on the comparison results.

```
//LandingSys : block representant le systeme train d'atterissage et integrant l'IHM, la partie
     controle et les parties meca/hydrau
block landingSys
-------------------------------------------------------------------------------------------------

    //PilotInterface : Interface between the pilot and the landing gear
-------------------------------------------------------------------------------------------------

    block pilotInterface
        block frontGearLight
            port input;
            port output;
        end
        block leftGearLight
            port input;
            port output;
        end
        block rightGearLight
            port input;
            port output;
        end
        block udHandle
            port output;
            port position;
        end
        block analogicalSwitch
            port recentChange;
            port output;
        end
    end
-------------------------------------------------------------------------------------------------

    //MechaHydraulicPart : Mechanical and Hydraulic parts of the landing gear system
-------------------------------------------------------------------------------------------------
```

```
block mechahydraulicPart
    block frontLandingSet
        block upLockBox
            port rightpos;
            port input;
        end
        block door
            port input;
            port rightPos;
        end
        block doorCylinder
            port input;
            port output;
        end
        block gearCylinder
            port input;
            port output;
        end
        block gear
            port input;
            port rightPos;
        end
    end
    block leftLandingSet
        block upLockBox
            port rightpos;
            port input;
        end
        block door
            port input;
            port rightPos;
        end
        block doorCylinder
            port input;
            port output;
        end
        block gearCylinder
            port input;
            port output;
        end
        block gear
            port input;
            port rightPos;
        end
    end
    block rightLandingSet
        block upLockBox
            port rightpos;
            port input;
        end
        block door
            port input;
            port rightPos;
        end
        block doorCylinder
            port input;
            port output;
        end
        block gearCylinder
            port input;
            port output;
        end
        block gear
            port input;
            port rightPos;
        end
    end
    block hydraulicSys
        block pressureSensor
            port input;
            port output;
        end
```

```
            block generalElectroValve
                port input;
                port output;
                port order;
            end
            block doorHydraulicSys
                block retractionDoorElectroValve
                    port input;
                    port output;
                    port order;
                end
                block extensionDoorElectroValve
                    port input;
                    port output;
                    port order;
                end
                port input;
                port order;
            end
            block gearHydraulicSys
                block retractionGearElectroValve
                    port input;
                    port output;
                    port order;
                end
                block extensionGearElectroValve
                    port input;
                    port output;
                    port order;
                end
                port input;
                port order;
            end
        end
        port gearLocked;
        port doorLocked;
    end
-------------------------------------------------------------------------------------------------

    //DigitalPart : Digital part of the landing gear that controls the system
-------------------------------------------------------------------------------------------------

    block digitalPart
        block CPIOM1
            port analogicalSwitchFlag;
            port input;
            port outputGeneralValve;
            port outputGear;
            port outputDoor;
            port gearLocked;
            port doorLocked;
        end
        block CPIOM2
            port analogicalSwitchFlag;
            port input;
            port outputGeneralValve;
            port outputGear;
            port outputDoor;
            port gearLocked;
            port doorLocked;
        end
    end
end
```

B.4 Comparison results

This section contains the comparison results after the last step of the SmartSync comparison of the MBSE and MBSA models for the landing gear. Each row contains one element of the system and its representations in each model if it exists.

The first column indicates the element type, which can be either a port or a block. The inscription "MCB" indicates that the SmarSync tool output "`Missing Corresponding Block`" for this alignment; this is because the tool currently does not allow to align two model elements if their parents are not aligned.

The second row contains the name of the model element in the MBSE model.

The third row contains the name of the model element in the MBSA model.

Type	Model1	Model2
	main. Landing_gear_systeme	main. landingSys
port	ElectricPower	forget
port	Electrical_Cmd	forget
port	Hydraulic_power	forget
port	Manual_Cmd	forget
port	Mech_Aircraft	forget
port	Mech_Ground	forget
block	Pilot_interface	pilotInterface
block	mecahydraulic_part	mechahydraulicPart
port	Digital_part1. ElectricPower	forget
block	Pilot_interface. GearLight2	pilotInterface. leftGearLight
block	Pilot_interface. Gearlight1	pilotInterface. frontGearLight
block	Pilot_interface. Gearlight3	pilotInterface. rightGearLight
block	Pilot_interface. Handle	pilotInterface. udHandle
port	forget	mechahydraulicPart. doorLocked
port	forget	mechahydraulicPart. gearLocked
block	mecahydraulic_part. Front_landing_box	mechahydraulicPart. frontLandingSet
block	mecahydraulic_part. Hydraulic_part	mechahydraulicPart. hydraulicSys
block	mecahydraulic_part. Left_landing_box	mechahydraulicPart. leftLandingSet
block	mecahydraulic_part. Right_landing_box	mechahydraulicPart. rightLandingSet

port	Pilot_interface. GearLight2. Lsig	pilotInterface. leftGearLight. input
port	Pilot_interface. Gearlight1. Lsig	pilotInterface. frontGearLight. input
port	Pilot_interface. Gearlight3. Lsig	pilotInterface. rightGearLight. input
port	Pilot_interface. Handle. Man_Cmd	pilotInterface. udHandle. output
port	Pilot_interface. Handle. Manual_Cmd	pilotInterface. udHandle. position
block	mecahydraulic_part. Front_landing_box. Front_Door	mechahydraulicPart. frontLandingSet. door
block	mecahydraulic_part. Front_landing_box. Front_Gear	mechahydraulicPart. frontLandingSet. gear
block	mecahydraulic_part. Hydraulic_part. Front_Door_cylinder	mechahydraulicPart. frontLandingSet. doorCylinder
block	mecahydraulic_part. Hydraulic_part. Front_Gear_cylinder	mechahydraulicPart. frontLandingSet. gearCylinder
block	mecahydraulic_part. Hydraulic_part. Left_Gear_cylinder	mechahydraulicPart. leftLandingSet. gearCylinder
block	mecahydraulic_part. Hydraulic_part. Left_door_cylinder	mechahydraulicPart. leftLandingSet. doorCylinder
block	mecahydraulic_part. Hydraulic_part. Right_Door_cylinder	mechahydraulicPart. rightLandingSet. doorCylinder
block	mecahydraulic_part. Hydraulic_part. Right_Gear_Cylinder	mechahydraulicPart. rightLandingSet. gearCylinder
block	mecahydraulic_part. Left_landing_box. Left_Door	mechahydraulicPart. leftLandingSet. door
block	mecahydraulic_part. Right_landing_box. Right_Gear	mechahydraulicPart. rightLandingSet. gear
block	mecahydraulic_part. Right_landing_box. Right_door	mechahydraulicPart. rightLandingSet. door
port	mecahydraulic_part. Front_landing_box. Front_Door. Mech	mechahydraulicPart. frontLandingSet. door. input

port	mecahydraulic_part. Front_-landing_box. Front_Door. Mech_Aircraft	forget
port	forget	mechahydraulicPart. front-LandingSet. door. rightPos
port	mecahydraulic_part. Front_-landing_box. Front_Gear. Mech	mechahydraulicPart. front-LandingSet. gear. input
port	mecahydraulic_part. Front_-landing_box. Front_Gear. Mech_Ground	forget
port	forget	mechahydraulicPart. front-LandingSet. gear. rightPos
MCB	mecahydraulic_part. Hy-draulic_part. Front_Door_-cylinder	forget
MCB	mecahydraulic_part. Hy-draulic_part. Front_Gear_-cylinder	forget
MCB	mechahydraulicPart. front-LandingSet. doorCylinder	forget
MCB	mechahydraulicPart. front-LandingSet. gearCylinder	forget
MCB	mechahydraulicPart. leftLand-ingSet. gearCylinder	forget
MCB	mechahydraulicPart. leftLand-ingSet. doorCylinder	forget
MCB	mechahydraulicPart. rightLand-ingSet. doorCylinder	forget
MCB	mechahydraulicPart. rightLand-ingSet. gearCylinder	forget
port	mecahydraulic_part. Left_land-ing_box. Left_Door. Mech	mechahydraulicPart. leftLand-ingSet. door. input
block	mecahydraulic_part. Left_land-ing_box. Left_Gear	mechahydraulicPart. leftLand-ingSet. gear
MCB	mecahydraulic_part. Hy-draulic_part. Left_door_cyl-inder	forget
MCB	mecahydraulic_part. Hy-draulic_part. Left_Gear_cyl-inder	forget
port	mecahydraulic_part. Right_-landing_box. Right_Gear. Mech	mechahydraulicPart. rightLand-ingSet. gear. input

port	mecahydraulic_part. Right_-landing_box. Right_door. Mech	mechahydraulicPart. rightLand-ingSet. door. input
MCB	mecahydraulic_part. Hy-draulic_part. Right_Door_-cylinder	forget
MCB	mecahydraulic_part. Hy-draulic_part. Right_Gear_-Cylinder	forget
port	mecahydraulic_part. Left_land-ing_box. Left_Gear. Mech	mechahydraulicPart. leftLand-ingSet. gear. input
block		digitalPart
block	Digital_part1	
block	Digital_part2	
block		pilotInterface. analogicalSwitch
port		pilotInterface. leftGearLight. output
port		pilotInterface. frontGearLight. output
port		pilotInterface. rightGearLight. output
port		pilotInterface. analogicalSwitch
block		mechahydraulicPart. front-LandingSet. upLockBox
block	mecahydraulic_part. Hy-draulic_part. Close_door_-Electro_valve	
block	mecahydraulic_part. Hy-draulic_part. Extend_Gear_-Electro_Valve	
block	mecahydraulic_part. Hy-draulic_part. General_elec-tro_valves	
block	mecahydraulic_part. Hy-draulic_part. Open_door_-Electro_Valve	
block	mecahydraulic_part. Hy-draulic_part. Retract_Gear_-Electro_valve	
block		mechahydraulicPart. hydraulic-Sys. doorHydraulicSys
block		mechahydraulicPart. hydraulic-Sys. gearHydraulicSys

block		mechahydraulicPart. hydraulic-Sys. generalElectroValve
block		mechahydraulicPart. hydraulic-Sys. pressureSensor
block		mechahydraulicPart. leftLand-ingSet. upLockBox
block		mechahydraulicPart. rightLand-ingSet. upLockBox
port	mecahydraulic_part. Front_-landing_box. Front_Door. Status	
port	mecahydraulic_part. Front_-landing_box. Front_Gear. Status	
block		mechahydraulicPart. front-LandingSet. upLockBox
block	mecahydraulic_part. Hy-draulic_part. Close_door_-Electro_valve	
block	mecahydraulic_part. Hy-draulic_part. Extend_Gear_-Electro_Valve	
block	mecahydraulic_part. Hy-draulic_part. General_elec-tro_valves	
block	mecahydraulic_part. Hy-draulic_part. Open_door_-Electro_Valve	
block	mecahydraulic_part. Hy-draulic_part. Retract_Gear_-Electro_valve	
block		mechahydraulicPart. hydraulic-Sys. doorHydraulicSys
block		mechahydraulicPart. hydraulic-Sys. gearHydraulicSys
block		mechahydraulicPart. hydraulic-Sys. generalElectroValve
block		mechahydraulicPart. hydraulic-Sys. pressureSensor
port	mecahydraulic_part. Left_land-ing_box. Left_Door. Mech_Air-craft	

port	mecahydraulic_part. Left_landing_box. Left_Door. Status	
port		mechahydraulicPart. leftLandingSet. door. rightPos
block		mechahydraulicPart. leftLandingSet. upLockBox
port	mecahydraulic_part. Right_landing_box. Right_Gear. Mech_Ground	
port	mecahydraulic_part. Right_landing_box. Right_Gear. Status	
port		mechahydraulicPart. rightLandingSet. gear. rightPos
port	mecahydraulic_part. Right_landing_box. Right_door. Mech_Aircraft	
port	mecahydraulic_part. Right_landing_box. Right_door. Status	
port		mechahydraulicPart. rightLandingSet. door. rightPos
block		mechahydraulicPart. rightLandingSet. upLockBox
port	mecahydraulic_part. Left_landing_box. Left_Gear. Mech_Ground	
port	mecahydraulic_part. Left_landing_box. Left_Gear. Status	
port		mechahydraulicPart. leftLandingSet. gear. rightPos

APPENDIX C

BLOOD DELIVERY DRONE MODELS

C.1 AltaRica 3.0 code for the Blood delivery drone

This section contains the code for the AltaRica 3.0 model of the blood delivery drone. This code is composed of the main block that instantiates the main subsystems of the drone. Contrary to the Landing Gear model from appendix B.1, components and subsystems are modeled as blocks rather than using specific classes.

Those blocks extend either the NRComponent class, which contains the standard OK/KO guarded transition system used in each model component or the ComponentIO class, which instantiates the NRComponent class and adds input and output variables with an assertion linking them.

```
block ZippyFlyer
   block Cell
      block Fuselage
         extends NRComponent;
      end

      block Wing
         extends NRComponent;
      end

      block Bunker
         block Actuator
            extends NRComponent;
            Boolean inBattery, inCalculator, out (reset = false);
            assertion
               out := if is_working then inCalculator and inBattery else false;
         end

         block Door
            extends NRComponent;
            Boolean in, pos (reset = false);
            assertion
               pos := if is_working then in else false;
         end
         assertion
```

```
            Door.in := Actuator.out;
    end

    Aileron Aileron1;
    Aileron Aileron2;

end

block Avionics
    block Comms_module
        extends NRComponent;
        Boolean in4G (reset = true);
        Boolean inCalculator (reset = false);
        Boolean outData (reset = false);
        Boolean outCalculator (reset = true);
        assertion
            outData := if is_working then in4G and inCalculator else false;
            outCalculator := if is_working then in4G else false;
    end

    block GPS_module
        extends ComponentIO;
        assertion
            in := true;
    end
end

block Battery
    extends ComponentIO;
    assertion
        in := true;
end

block Radar
    extends ComponentIO;
    assertion
        in := true;
end

block Drivetrain
    block Motor1
        extends ComponentIO;
    end
    block Motor2
        extends ComponentIO;
    end
    block Propeller1
        extends ComponentIO;
    end
    block Propeller2
        extends ComponentIO;
    end
    assertion
        Propeller1.in := Motor1.out;
        Propeller2.in := Motor2.out;
end

block Inertial_measurement_unit
    extends NRComponent;
    Boolean inAttitude, inRadar, inGPS, out (reset = false);
    assertion
        inAttitude := true;
        out := if is_working then inAttitude and inRadar and inGPS else false;
end

block Calculator
    extends NRComponent;
    Boolean inPlanVol (init = true);
    Boolean inCentraleInertielle, inBattery, inGPS, outLargage, outComm, outMotor (reset =
        false);
    Boolean inComm (reset = true);
end
```

```
    assertion
        Inertial_measurement_unit.inRadar := Radar.out;
        Inertial_measurement_unit.inGPS := Avionics.GPS_module.out;
        Calculator.inGPS := Avionics.GPS_module.out;
        Calculator.inBattery := Battery.out;
        Calculator.inCentraleInertielle := Inertial_measurement_unit.out;
        Calculator.inComm := Avionics.Comms_module.outCalculator;
        Avionics.Comms_module.inCalculator := Calculator.outComm;
        Drivetrain.Motor1.in := Calculator.outMotor;
        Drivetrain.Motor2.in := Calculator.outMotor;
        Cell.Bunker.Actuator.inCalculator := Calculator.outLargage;
    observer Boolean oPuissanceNominale = Drivetrain.Motor1.out and Drivetrain.Motor2.out;
    observer Boolean oPuissanceDegradee = not (Drivetrain.Motor1.out and Drivetrain.Motor2.out) and
         (Drivetrain.Motor1.out or Drivetrain.Motor2.out);
end

class NRComponent
    Boolean is_working (init = true);
    parameter Real lambda = 10e-6;
    event failure (delay = exponential(lambda));
    transition
        failure : is_working == true -> is_working := false;
end

class ComponentIO
    extends NRComponent;
    Boolean in, out (reset = false);
    assertion
        out := if is_working then in else false;
end

class Aileron
    extends NRComponent;
end
```

C.2 SCOLA code for the Blood delivery drone

This section contains the SCOLA code for the blood delivery drone. This model is divided into two parts. First, the block drone contains the architecture of the drone, with blocks representing the subsystems and components. Second, a scenario is defined, with states that contain tasks. The tasks are computed during the execution of the model; they can impact the architecture of the drone, for example, by modifying the values of the variables or by creating new blocks/variables or deleting them.

```
domain EtatGPS {BASE, DEST, Route_Retour_BASE, Route_vers_DEST} end
domain ComponentState {INSTALLED, MISSING, STOPPED, EN_MARCHE} end
domain STATE_Porte_soute {OPENED, CLOSED} end

block drone
  block GPS
      ComponentState Recept_GPS STOPPED
      EtatGPS Position BASE                        /*initialement, le drone se trouve dans la base*/
  end
  block Soute
      ComponentState colis MISSING                 /*initialement, le colis n'est pas encore
           chargee dans le drone*/
      block Actionneurs
      ComponentState Actionneurs_state STOPPED
      end
      block Porte
```

```
    STATE_Porte_soute Porte_soute OPENED        /*initialement, la porte de la soute est ouverte
         pour pouvoir charger le colis de sang*/
    end
  end
  block GroupeMotopropulseur
    block Moteur_Avant
    Boolean F1 false          /*les moteurs sont initialement en arret*/
    end
    block Moteur_Arriere
    Boolean F2 false
    end
    block Helice_Avant
    ComponentState HeliceAvant_state STOPPED    /*les helices sont initialement en arret*/
    end
    block Helice_Arriere
    ComponentState HeliceArriere_state STOPPED
    end
  end
  block batterie
    ComponentState batterie_state MISSING       /*initialement, la batterie n'est pas encore
         chargee dans le drone*/
    Boolean Charged true
  end
  block Calculateur
    ComponentState calculateur STOPPED
   end
end



   scenario DRONE as drone
     state Initial                                                  /*a l'etat initial on entre
          les donnees de la mission*/
      scenario preparation_mission
         task chargement                                           /*la tache de chargement de
              batterie et colis*/
           set Soute.colis INSTALLED                               /*le colis est chargee apres
                avoir terminee la tache du chargement*/
           set batterie.batterie_state INSTALLED  /*la batterie est chargee apres avoir terminee
                la tache du chargement*/
           set GPS.Recept_GPS EN_MARCHE                            /*le recepteur GPS se met en
                marche juste apres le chargement*/
           set Calculateur.calculateur EN_MARCHE
           set Soute.Porte.Porte_soute CLOSED                      /*apres avoir chargee le
                colis, l'operateur ferme la porte de la soute*/
          end
          state lecture_code_QR
          choice verification_code_QR
           branch code_bon
           branch QR_different

          end
          state erreur
          state confirmation

       next chargement lecture_code_QR
       next lecture_code_QR verification_code_QR
       next verification_code_QR.code_bon confirmation
       next verification_code_QR.QR_different erreur

    end

    task Decollage
       set GroupeMotopropulseur.Moteur_Avant.F1 true /*les moteurs se mettent en marche pour
            propulsion*/
       set GroupeMotopropulseur.Moteur_Arriere.F2 true
       set GPS.Position Route_vers_DEST                              /*le drone est maintenant
            en route vers la destination*/
       set GroupeMotopropulseur.Helice_Avant.HeliceAvant_state EN_MARCHE /*les helices commencent
            a tourner grace aux moteurs*/
       set GroupeMotopropulseur.Helice_Arriere.HeliceArriere_state EN_MARCHE
    end
```

```
 state Vol

scenario MOTEUR_AVANT
 state initial
 choice Etat_moteur_avant
    branch Fct
    branch Dysfct
 end
 task dysfonctionnement_moteur_avant
    set GroupeMotopropulseur.Moteur_Avant.F1 false
 end
 state fonctionnement_moteur_avant
 state fin_test

 next initial Etat_moteur_avant
 next Etat_moteur_avant.Fct fonctionnement_moteur_avant
 next Etat_moteur_avant.Dysfct dysfonctionnement_moteur_avant
 next fonctionnement_moteur_avant fin_test
 next dysfonctionnement_moteur_avant fin_test
end

scenario MOTEUR_ARRIERE
 state initial
 choice Etat_moteur_arriere
    branch Fct
    branch Dysfct
 end
 task dysfonctionnement_moteur_arriere
    set GroupeMotopropulseur.Moteur_Arriere.F2 false
 end

state fonctionnement_moteur_arriere
state fin_test

next initial Etat_moteur_arriere
next Etat_moteur_arriere.Fct fonctionnement_moteur_arriere
next Etat_moteur_arriere.Dysfct dysfonctionnement_moteur_arriere
next fonctionnement_moteur_arriere fin_test
next dysfonctionnement_moteur_arriere fin_test
end

test Test_Moteurs
    case Dysfonct_Moteur_avant (and (eq GroupeMotopropulseur.Moteur_Avant.F1 false)(eq
         GroupeMotopropulseur.Moteur_Arriere.F2 true))
    case Dysfonct_Moteur_arriere (and (eq GroupeMotopropulseur.Moteur_Avant.F1 true)(eq
         GroupeMotopropulseur.Moteur_Arriere.F2 false))
    case Dysfonct_2_moteurs (and (eq GroupeMotopropulseur.Moteur_Avant.F1 false)(eq
         GroupeMotopropulseur.Moteur_Arriere.F2 false))
    case FCT_2_moteurs (and (eq GroupeMotopropulseur.Moteur_Avant.F1 true)(eq
         GroupeMotopropulseur.Moteur_Arriere.F2 true))
end


  scenario Guidage as GPS                                          /*pendant le vol, le GPS
      calcule toujours la position (un sous-scenario du GPS)*/
    state calcul
    choice Info_GPS
       branch DEST                 /*le drone est arrivee a la destination*/
       branch Route_vers_DEST      /*le drone est en encore en route vers la destination*/
       branch Route_Retour_BASE    /*le drone est en route pour retourner a la base*/
       branch BASE                /*le drone est arrivee a la base*/
    end
    task arrive_dest
       set Position DEST
       set owner.Soute.Porte.Porte_soute OPENED          /*si le drone est arrivee a la
            destination, les actionneurs vont se mettre en marche pour ouvrir la porte*/
       set owner.Soute.Actionneurs.Actionneurs_state EN_MARCHE
    end
    task arrive_base
       set Position BASE
    end

     next calcul Info_GPS
```

```
        next Info_GPS.DEST arrive_dest
        next Info_GPS.BASE arrive_base
      end

    task largage
       set Soute.colis MISSING              /*apres le largage, le colis est livree*/
       set GPS.Position Route_Retour_BASE   /*le drone retourne a la base*/
       set Soute.Porte.Porte_soute CLOSED   /*la porte est fermee*/
    end
    state VolRetour

    task Atterrissage
       set GroupeMotopropulseur.Moteur_Avant.MoteurAvant_state STOPPED
       set GroupeMotopropulseur.Moteur_Arriere.MoteurArriere_state STOPPED
       set GroupeMotopropulseur.Helice_Avant.HeliceAvant_state STOPPED
       set GroupeMotopropulseur.Helice_Arriere.HeliceArriere_state STOPPED
       set Calculateur.calculateur STOPPED
       set GPS.Position BASE
    end
    choice Inspection
       branch maintenance        /*si le drone a besoin de maintenance*/
       branch accept             /*si le drone n'a pas besoin de maintenance*/
       branch reject             /*fin de vie*/
    end
    task maintenance end
    state Perte_puissance
    state Chute
    state Terminal

    next Initial preparation_mission.chargement
    next preparation_mission.confirmation Decollage
    next Decollage Test_Moteurs
    next Test_Moteurs.FCT_2_moteurs Vol
    next Test_Moteurs.Dysfonct_Moteur_avant Perte_puissance
    next Test_Moteurs.Dysfonct_Moteur_arriere Perte_puissance
    next Test_Moteurs.Dysfonct_2_moteurs Chute
    next Perte_puissance Terminal
    next Chute Terminal

    next Vol Guidage.calcul
    next Guidage.Info_GPS.Route_vers_DEST Vol
    next Guidage.arrive_dest largage
    next Guidage.arrive_base Atterrissage
    next largage VolRetour
    next VolRetour Guidage.calcul
    next Guidage.Info_GPS.Route_Retour_BASE VolRetour
    next Atterrissage Inspection
    next Inspection.maintenance maintenance
    next Inspection.accept preparation_mission.chargement
    next maintenance preparation_mission.chargement
    next Inspection.reject Terminal
   end
```

C.3 S2ML code for the MBSE model

This section contains the S2ML code translation of the MBSE model for the blood delivery drone. The translation was operated manually, SysML part properties were translated to blocks, ports to ports and connections to connections.

```
block Alternative_1
   block Batterie
      port Ener_elec;
      port Energie;
   end
   block groupe_Motopropulseur
```

```
        block Moteur_Arr
            port Action;
            port pilotage;
        end
        block Helice_Arr
            port Action;
        end
        block Moteur_Av
            port pilotage;
            port Action;
        end
        block Helice_Av
            port Action;
        end
        port effort_propulsion;
        port Energie;
        connection [Moteur_Arr.Action, Helice_Arr.Action];
        connection [Moteur_Av.Action, Helice_Av.Action];
    end
    block centrale_inertielle
        port Orientation;
        port position_obstacle;
        port position;
        port trajectoire;
    end
    block radar
        port Image_obstacle;
        port position_obstacle;
    end
    block avionique
        block module_de_communication
            port Reseau_4G;
            port Information;
            port Data_out;
        end
        block module_de_geolocalisation
            port Signaux_geolocalisation;
            port position;
        end
        port Energie;
    end
    block cellule
        block aile_d
            port effort_portance;
        end
        block aile_g
            port effort_portance;
        end
        block aileron_d
            port effort_directionnel;
            port Action_aileron;
        end
        block aileron_g
            port effort_directionnel;
            port Action_aileron;
        end
        block aileron_d2
            port effort_directionnel;
            port Action_aileron;
        end
        block aileron_g2
            port effort_directionnel;
            port Action_aileron;
        end
        block actionneurs_Ailerons
            port Action_aileron;
            port Commande_de_vol;
        end
        block gouverne_de_direction
            port Commande_de_vol;
        end
        block soute
            block actionneurs
```

```
            port Energie;
            port ordre_ouverture;
            port Ordre_largage;
        end
        block Porte_soute
            port ordre_ouverture;
        end
        port Poche_de_sang;
        connection [actionneurs.ordre_ouverture, Porte_soute.ordre_ouverture];
    end
    port Gravitee;
    port effort_trainee;
    connection [actionneurs_Ailerons.Action_aileron, aileron_d.Action_aileron];
    connection [actionneurs_Ailerons.Action_aileron, aileron_d2.Action_aileron];
    connection [actionneurs_Ailerons.Action_aileron, aileron_g.Action_aileron];
    connection [actionneurs_Ailerons.Action_aileron, aileron_g2.Action_aileron];
end
block calculateur
    port Energie;
    port pilotage;
    port Ordre_largage;
    port Commande_de_vol;
    port Information;
    port plan_de_vol;
    port position;
    port trajectoire;
end
port Gravitee;
port ActionAir;
port Poche_de_sang;
port Ener_elec;
port Orientation;
port Image_obstacle;
port Data_in;
port Reseau_4G;
port Signaux_geolocalisation;
port Data_out;
connection [Gravitee, cellule.Gravitee];
connection [ActionAir, groupe_Motopropulseur.effort_propulsion];
connection [Ener_elec, Batterie.Ener_elec];
connection [Data_in, calculateur.plan_de_vol];
connection [Orientation, centrale_inertielle.Orientation];
connection [Image_obstacle, radar.Image_obstacle];
connection [ActionAir, cellule.effort_trainee];
connection [calculateur.pilotage, groupe_Motopropulseur.Moteur_Arr.pilotage];
connection [calculateur.pilotage, groupe_Motopropulseur.Moteur_Av.pilotage];
connection [Reseau_4G, avionique.module_de_communication.Reseau_4G];
connection [Signaux_geolocalisation,
     avionique.module_de_geolocalisation.Signaux_geolocalisation];
connection [avionique.module_de_communication.Data_out, Data_out];
connection [avionique.module_de_communication.Information, calculateur.Information];
connection [avionique.module_de_geolocalisation.position, centrale_inertielle.position];
connection [avionique.module_de_geolocalisation.position, calculateur.position];
connection [radar.position_obstacle, centrale_inertielle.position_obstacle];
connection [centrale_inertielle.trajectoire, calculateur.trajectoire];
connection [ActionAir, cellule.aile_d.effort_portance];
connection [ActionAir, cellule.aile_g.effort_portance];
connection [ActionAir, cellule.aileron_d.effort_directionnel];
connection [ActionAir, cellule.aileron_d2.effort_directionnel];
connection [ActionAir, cellule.aileron_g.effort_directionnel];
connection [ActionAir, cellule.aileron_g2.effort_directionnel];
connection [Poche_de_sang, cellule.soute.Poche_de_sang];
connection [calculateur.Ordre_largage, cellule.soute.actionneurs.Ordre_largage];
connection [calculateur.Commande_de_vol, cellule.actionneurs_Ailerons.Commande_de_vol];
connection [calculateur.Commande_de_vol, cellule.gouverne_de_direction.Commande_de_vol];
connection [Batterie.Energie, cellule.soute.actionneurs.Energie];
connection [Batterie.Energie, calculateur.Energie];
connection [Batterie.Energie, avionique.Energie];
connection [Batterie.Energie, groupe_Motopropulseur.Energie];
end
```

C.4 S2ML code for the MBSA model

This section contains the code for the S2ML translation of the AltaRica 3.0 model. We translated the model manually, although it would be possible to automatize this process. We translated blocks to blocks and variables to ports. Contrary to the translation for the landing gear in appendix B.3, we translated the whole model, including safety artifacts only related to the Guarded Transition Systems and assertions.

Assertions and transitions are translated using connections. Annotations are used in this to give precise information over the ports and connections, for example a connection will be annotated `(type = "connection")` if it is a transition, and `(type = "assertion")` if it is an assertion. The information contained in the annotations is not used by the SmartSync tool but can be read by the engineers.

```
block ZippyFlyer
   block Cellule
      block Fuselage
         port is_working (type = "Boolean", kind = "init", value = "true");
         port lambda (type = "Real", value = 10e-6);
         port failure (type = "event", delay = "exponential(lambda)");
         connection [failure, is_working, is_working] (type = "connection", expr = "true ->
              false");
      end
      block Aile
         port is_working (type = "Boolean", kind = "init", value = "true");
         port lambda (type = "Real", value = 10e-6);
         port failure (type = "event", delay = "exponential(lambda)");
         connection [failure, is_working, is_working] (type = "connection");
      end
      block Soute
         block Actionneur
            port is_working (type = "Boolean", kind = "init", value = "true");
            port lambda (type = "Real", value = 10e-6);
            port failure (type = "event", delay = "exponential(lambda)");
            connection [failure, is_working, is_working] (type = "connection", expr = "true ->
                 false");
            port inBatterie (type = "Boolean", kind = "reset", value = "false");
            port inCalculateur (type = "Boolean", kind = "reset", value = "false");
            port out (type = "Boolean", kind = "reset", value = "false");
            connection [out, is_working, inCalculateur, inBatterie](kind = "assertion");
         end
         block Porte
            port is_working (type = "Boolean", kind = "init", value = "true");
            port lambda (type = "Real", value = 10e-6);
            port failure (type = "event", delay = "exponential(lambda)");
            connection [failure, is_working, is_working] (type = "connection", expr = "true ->
                 false");
            port in (type = "Boolean", kind = "reset", value = false);
            port pos (type = "Boolean", kind = "reset", value = false);
            connection [pos, is_working, in](type = "assertion");
         end
         connection [Porte.in, Actionneur.out](type = "assertion");
      end
      block Aileron1
         port is_working (type = "Boolean", kind = "init", value = "true");
         port lambda (type = "Real", value = 10e-6);
         port failure (type = "event", delay = "exponential(lambda)");
         connection [failure, is_working, is_working] (type = "connection", expr = "true ->
              false");
      end
      block Aileron2
         port is_working (type = "Boolean", kind = "init", value = "true");
         port lambda (type = "Real", value = 10e-6);
```

```
        port failure (type = "event", delay = "exponential(lambda)");
        connection [failure, is_working, is_working] (type = "connection", expr = "true ->
            false");
    end
end
block Avionique
    block Module_communication
        port is_working (type = "Boolean", kind = "init", value = "true");
        port lambda (type = "Real", value = 10e-6);
        port failure (type = "event", delay = "exponential(lambda)");
        connection [failure, is_working, is_working] (type = "connection", expr = "true ->
            false");
        port in4G (type = "Boolean", kind = "reset", value = true);
        port inCalculateur (type = "Boolean", kind = "reset", value = "false");
        port outData (type = "Boolean", kind = "reset", value = true);
        port outCalculateur (type = "Boolean", kind = "reset", value = true);
        connection [outData, is_working, in4G](type = "assertion");
        connection [outCalculateur, is_working, in4G](type = "assertion");
    end
    block Module_GPS
        port is_working (type = "Boolean", kind = "init", value = "true");
        port lambda (type = "Real", value = 10e-6);
        port failure (type = "event", delay = "exponential(lambda)");
        connection [failure, is_working, is_working] (type = "connection", expr = "true ->
            false");
        port in (type = "Boolean", kind = "reset", value = false);
        port out (type = "Boolean", kind = "reset", value = false);
        connection [out, is_working, in](type = "assertion");
        connection [in](type = "assertion");
    end
end
block Batterie
    port is_working (type = "Boolean", kind = "init", value = "true");
    port lambda (type = "Real", value = 10e-6);
    port failure (type = "event", delay = "exponential(lambda)");
    connection [failure, is_working, is_working] (type = "connection", expr = "true -> false");
    port in (type = "Boolean", kind = "reset", value = false);
    port out (type = "Boolean", kind = "reset", value = false);
    connection [out, is_working, in](type = "assertion");
    connection [in](type = "assertion");
end
block Radar
    port is_working (type = "Boolean", kind = "init", value = "true");
    port lambda (type = "Real", value = 10e-6);
    port failure (type = "event", delay = "exponential(lambda)");
    connection [failure, is_working, is_working] (type = "connection", expr = "true -> false");
    port in (type = "Boolean", kind = "reset", value = false);
    port out (type = "Boolean", kind = "reset", value = false);
    connection [out, is_working, in](type = "assertion");
    connection [in](type = "assertion");
end
block Groupe_motopropulseur
    block Motor1
        port is_working (type = "Boolean", kind = "init", value = "true");
        port lambda (type = "Real", value = 10e-6);
        port failure (type = "event", delay = "exponential(lambda)");
        connection [failure, is_working, is_working] (type = "connection", expr = "true ->
            false");
        port in (type = "Boolean", kind = "reset", value = false);
        port out (type = "Boolean", kind = "reset", value = false);
        connection [out, is_working, in](type = "assertion");
    end
    block Motor2
        port is_working (type = "Boolean", kind = "init", value = "true");
        port lambda (type = "Real", value = 10e-6);
        port failure (type = "event", delay = "exponential(lambda)");
        connection [failure, is_working, is_working] (type = "connection", expr = "true ->
            false");
        port in (type = "Boolean", kind = "reset", value = false);
        port out (type = "Boolean", kind = "reset", value = false);
        connection [out, is_working, in](type = "assertion");
    end
    block Propeller1
```

```
            port is_working (type = "Boolean", kind = "init", value = "true");
            port lambda (type = "Real", value = 10e-6);
            port failure (type = "event", delay = "exponential(lambda)");
            connection [failure, is_working, is_working] (type = "connection", expr = "true ->
                 false");
            port in (type = "Boolean", kind = "reset", value = false);
            port out (type = "Boolean", kind = "reset", value = false);
            connection [out, is_working, in](type = "assertion");
        end
        block Propeller2
            port is_working (type = "Boolean", kind = "init", value = "true");
            port lambda (type = "Real", value = 10e-6);
            port failure (type = "event", delay = "exponential(lambda)");
            connection [failure, is_working, is_working] (type = "connection", expr = "true ->
                 false");
            port in (type = "Boolean", kind = "reset", value = false);
            port out (type = "Boolean", kind = "reset", value = false);
            connection [out, is_working, in](type = "assertion");
        end
        connection [Propeller1.in, Motor1.out](type = "assertion");
        connection [Propeller2.in, Motor2.out](type = "assertion");
    end
    block Centrale_inertielle
        port is_working (type = "Boolean", kind = "init", value = "true");
        port lambda (type = "Real", value = 10e-6);
        port failure (type = "event", delay = "exponential(lambda)");
        connection [failure, is_working, is_working] (type = "connection", expr = "true -> false");
        port inAttitude (type = "Boolean", kind = "reset", value = false);
        port inRadar (type = "Boolean", kind = "reset", value = false);
        port inGPS (type = "Boolean", kind = "reset", value = false);
        port out (type = "Boolean", kind = "reset", value = false);
        connection [inAttitude](type = "assertion");
        connection [out, is_working, inAttitude, inRadar, inGPS](type = "assertion");
    end
    block Calculateur
        port is_working (type = "Boolean", kind = "init", value = "true");
        port lambda (type = "Real", value = 10e-6);
        port failure (type = "event", delay = "exponential(lambda)");
        connection [failure, is_working, is_working] (type = "connection", expr = "true -> false");
        port inPlanVol (type = "Boolean", kind = "reset", value = true);
        port inCentraleInertielle (type = "Boolean", kind = "reset", value = false);
        port inBatterie (type = "Boolean", kind = "reset", value = false);
        port inGPS (type = "Boolean", kind = "reset", value = false);
        port inComm (type = "Boolean", kind = "reset", value = false);
        port outLargage (type = "Boolean", kind = "reset", value = false);
        port outComm (type = "Boolean", kind = "reset", value = false);
        port outMotor (type = "Boolean", kind = "reset", value = false);
    end
    connection [Centrale_inertielle.inRadar, Radar.out](type = "assertion");
    connection [Centrale_inertielle.inGPS, Avionique.Module_GPS.out](type = "assertion");
    connection [Calculateur.inGPS, Avionique.Module_GPS.out](type = "assertion");
    connection [Calculateur.inBatterie, Batterie.out](type = "assertion");
    connection [Calculateur.inCentraleInertielle, Centrale_inertielle.out](type = "assertion");
    connection [Calculateur.inComm, Avionique.Module_communication.outCalculateur](type =
         "assertion");
    connection [Avionique.Module_communication.inCalculateur, Calculateur.outComm](type =
         "assertion");
    connection [Groupe_motopropulseur.Motor1.in, Calculateur.outMotor](type = "assertion");
    connection [Groupe_motopropulseur.Motor2.in, Calculateur.outMotor](type = "assertion");
    connection [Cellule.Soute.Actionneur.inCalculateur, Calculateur.outLargage](type = "assertion");
end
```

C.5 S2ML code for the SCOLA model

This section contains the code for the S2ML translation of the SCOLA model. The translation of the architecture part is straightforward, with blocks translated to blocks and variables to ports. We also translated the scenario part of

the model, although it is not very interesting for comparison. The scenario was modeled as a block, and so were tasks and branches. The "nexts" were translated as connections between the states, which were translated as ports.

Annotations are used the same way as in the MBSA to S2ML translation. They carry information that can be used by the engineers but is not considered by the SmartSync tool.

```
block main
   block drone (type="{block, scenario, task}", value="block")
      block GPS (type="{block, scenario, task}", value="block")
        port Recept_GPS (type="{INSTALLED, MISSING, STOPPED, EN_MARCHE}",value="STOPPED");
        port Position (type="{BASE, DEST, Route_Retour_BASE, Route_vers_DEST}",value="BASE");
      end
      block Soute (type="{block, scenario, task}", value="block")
        port colis (type="{INSTALLED, MISSING, STOPPED, EN_MARCHE}",value="MISSING");
        block Actionneurs (type="{block, scenario}", value="block")
          port Actionneurs_state (type="{INSTALLED, MISSING, STOPPED,
               EN_MARCHE}",value="STOPPED");
        end
        block Porte (type="{block, scenario, task}", value="block")
          port Porte_soute (type="{OPENED, CLOSED}",value="OPENED");
        end
      end
      block GroupeMotopropulseur (type="{block, scenario, task}", value="block")
        block Moteur_Avant (type="{block, scenario}", value="block")
          port Working(type="Boolean",value="false");
        end
        block Moteur_Arriere (type="{block, scenario, task}", value="block")
          port Working (type="Boolean",value="false");
         end
         block Helice_Avant (type="{block, scenario, task}", value="block")
         port HeliceAvant_state (type="{INSTALLED, MISSING, STOPPED,
               EN_MARCHE}",value="STOPPED");
         end
         block Helice_Arriere (type="{block, scenario, task}", value="block")
         port HeliceArriere_state (type="{INSTALLED, MISSING, STOPPED,
               EN_MARCHE}",value="STOPPED");
         end
      end
      block batterie (type="{block, scenario, task}", value="block")
        port batterie_state (type="{INSTALLED, MISSING, STOPPED, EN_MARCHE}",value="MISSING");
        port Charged (type="Boolean",value="true");
      end
      block Calculateur (type="{block, scenario, task}", value="block")
        port calculateur (type="{INSTALLED, MISSING, STOPPED, EN_MARCHE}",value="STOPPED");
      end
   end
   block scenario_drone (type="{block, scenario, task}", value="scenario")
        port Initial, Terminal, lecture_code_QR,erreur, confirmation, Vol, VolRetour,
             Perte_puissance, Chute;
        block verification_code_QR (type="{block, scenario, task}", value="task")
          port default, code_bon, QR_different;
         end
        block preparation_mission (type="{block, scenario, task}", value="task")

          block chargement (type="{block, scenario, task}", value="task")
           port _in,_out;
           connection [Soute.colis] (type="set", value="INSTALLED");
           connection [batterie.batterie_state] (type="set", value="INSTALLED");
           connection [GPS.Recept_GPS] (type="set", value="EN_MARCHE");
           connection [Calculateur.calculateur] (type="set", value="EN_MARCHE");
           connection [Soute.Porte.Porte_soute] (type="set", value= "CLOSED");
          end
          connection [chargement._out, lecture_code_QR];
          connection [lecture_code_QR, verification_code_QR.default];
          connection [verification_code_QR.code_bon, confirmation];
          connection [verification_code_QR.QR_different, erreur];
        end
        block Decollage (type="{block, scenario, task}", value="task")
          port _in, _out;
```

```
   connection [GroupeMotopropulseur.Moteur_Avant.Working] (type="set", value="true");
   connection [GroupeMotopropulseur.Moteur_Arriere.Working] (type="set", value="true");
   connection [GPS.Position] (type="set", value="Route_vers_DEST");
   connection [GroupeMotopropulseur.Helice_Avant.HeliceAvant_state] (type="set",
        value="EN_MARCHE");
   connection [GroupeMotopropulseur.Helice_Arriere.HeliceArriere_state] (type="set",
        value="EN_MARCHE");
end
block Guidage (type="{block, scenario, task}", value="scenario")
   port calcul;
   block Info_GPS
      port default,DEST, Route_vers_DEST, Route_Retour_BASE, BASE;
   end
   block arrive_dest (type="{block, scenario, task}", value="task")
      port _in,_out;
      connection [GPS.Position] (type="set", value="DEST");
      connection [Soute.Porte.Porte_soute] (type="set", value="OPENED");
      connection [Soute.Actionneurs.Actionneurs_state] (type="set", value="EN_MARCHE");
   end
   block arrive_base (type="{block, scenario, task}", value="task")
      port _in, _out;
      connection [GPS.Position] (type="set", value="BASE");
   end

   connection [calcul, Info_GPS.default];
   connection [Info_GPS.DEST, arrive_dest];
   connection [Info_GPS.BASE, arrive_base];
end

block largage (type="{block, scenario, task}", value="task")
   port _in, _out;
   connection [Soute.colis] (type="set", value="MISSING");
   connection [GPS.Position] (type="set", value="Route_Retour_BASE");
   connection [Soute.Porte.Porte_soute] (type="set", value="CLOSED");
end

block Atterrissage (type="{block, scenario, task}", value="task")
   port _in, _out;
   connection [GroupeMotopropulseur.Moteur_Avant.Working] (type="set", value="false");
   connection [GroupeMotopropulseur.Moteur_Arriere.Working] (type="set", value="false");
   connection [GroupeMotopropulseur.Helice_Avant.HeliceAvant_state] (type="set",
        value="STOPPED");
   connection [GroupeMotopropulseur.Helice_Arriere.HeliceArriere_state] (type="set",
        value="STOPPED");
   connection [Calculateur.calculateur] (type="set", value="STOPPED");
   connection [GPS.Position] (type="set", value="BASE");
end

block Inspection (type="{block, scenario, task}", value="task")
   port default, maintenance, accept, reject;
end
block TEST_MOTEURS (type="{block, scenario, task}", value="scenario")
   port Default, Fct, Dysfct1, Dysfct2, Dysfct;
   connection [GroupeMotopropulseur.Moteur_Avant.Working.false,
        GroupeMotopropulseur.Moteur_Arriere.Working.true, Dysfct1];
   connection [GroupeMotopropulseur.Moteur_Avant.Working.true,
        GroupeMotopropulseur.Moteur_Arriere.Working.false, Dysfct2];
   connection [GroupeMotopropulseur.Moteur_Avant.Working.false,
        GroupeMotopropulseur.Moteur_Arriere.Working.false, Dysfct];
   connection [GroupeMotopropulseur.Moteur_Avant.Working.true,
        GroupeMotopropulseur.Moteur_Arriere.Working.true, Fct];
end


connection [Initial, preparation_mission.chargement._in];
connection [preparation_mission.confirmation, Decollage._in];
connection [Decollage._out, TEST_MOTEURS.Default];
connection [TEST_MOTEURS.Fct, Vol];
connection [TEST_MOTEURS.Dysfct1, Perte_puissance];
connection [TEST_MOTEURS.Dysfct2, Perte_puissance];
connection [TEST_MOTEURS.Dysfct, Chute];
connection [Perte_puissance, Terminal];
connection [Chute, Terminal];
```

```
        connection [Vol, Guidage.calcul];
        connection [Guidage.Info_GPS.Route_vers_DEST, Vol];
        connection [Guidage.arrive_dest, largage._in];
        connection [Guidage.arrive_base, Atterrissage._in];
        connection [largage._out, VolRetour];
        connection [VolRetour, Guidage.calcul];
        connection [Guidage.Info_GPS.Route_Retour_BASE, VolRetour];
        connection [Atterrissage._out, Inspection.default];
        connection [Inspection.maintenance, maintenance];
        connection [Inspection.accept, preparation_mission.chargement];
        connection [maintenance, preparation_mission.chargement];
        connection [Inspection.reject, Terminal];
      end
   end
end
```

C.6 S2ML code for the Modelica model

This section contains the code for the S2ML translation of the modelica model. Modelica classes were abstracted to S2ML classes, with model instances being translated to blocks. Modelica variables were translated to S2ML ports, and connect clauses were translated to connections. This translation was operated following the protocol described in [13].

```
class moteur
 port pin, pin_n, flange_b;
 block resistor
       port _in, _out;
       port r (value="0.299");
 end
 block inductor
       port _in, _out;
       port l (value="8e-5");
 end
 block emf
       port _in, _out, flange;
       port const (value="0.0302");
 end
 block inertia
       port _in, _out;
       port J (value="1.42e-5");
 end
 connection [pin, resistor._in];
 connection [resistor._out, inductor._in];
 connection [inductor._out, emf._in];
 connection [emf._out, pin_n];
 connection [emf.flange, inertia._in];
 connection [inertia._out, flange_b];
end

class helice
   port frame_a, flange_a;
   block revolute (type="liaison")
       port b, a, flange;
   end
   block Blade_1
       port a, b;
   end
   block Blade_2
       port a, b;
   end
   block torque (type="couple")
       port _in, flange_b;
   end
   block Poussee (type="force")
       port _in, frame_b;
```

```
    end
    block gain
        port u, y;
        port v (value="7e-6");
    end
    block gain1
        port u, y;
        port v (value="2e-7");
    end
    block product
        port u1, u2, y;
    end
    block speedSensor (type="sensor")
        port flange, w;
    end
    block fixedTranslation (type="liaison")
        port a, b;
    end

    connection [frame_a, revolute.b];
    connection [revolute.a, fixedTranslation.a];
    connection [fixedTranslation.b, Blade_1.a, Blade_2.a];
    connection [revolute.flange, flange_b, speedSensor.flange];
    connection [speedSensor.w, product.u1, product.u2];
    connection [product.y, gain.u, gain1.u];
    connection [gain.y, Poussee._in];
    connection [gain1.y, torque._in];
    connection [Poussee.frame_b, frame_a];
    connection [torque.flange_b, flange_a];
end


block drone_modelica

    moteur moteur1, moteur2;
    helice helice1, helice2;

    block SignalVoltage1 end
    block SignalVoltage2 end
    block constant1
        port Volt (value="20");
    end
    block const1
        port Volt (value="20");
    end


    block Front_Fuselage
        port a, b;
    end
    block Rear_Fuselage
        port a, b;
    end
    block Left_Wing
        port a, b;
    end
    block Right_Wing
        port a, b;
    end
    block Trainee (type="force")
        port _in, frame_b;
    end
    block Portance (type="force")
        port _in, frame_b;
    end
    block Vx (type="sensor")
        port frame, V;
    end
    block Height_Y (type="sensor")
        port frame, y;
    end
    block product
        port u1, u2, y;
```

```
    end
    block product1
        port u1, u2, y;
    end
    block gain
        port u, y;
        port v (value="0.2");
    end
    block gain1
        port u, y;
        port v (value="-0.01");
    end
    block Desired_Height
        port _out;
        port y (value="400");
    end
    block pid
        port u_s, u_m, y;
    end


    connection [helice1.frame_a, helice2.frame_a, Front_Fuselage.a, Rear_Fuselage.b, Left_Wing.a,
         Right_Wing.a, Trainee.frame, Portance.frame ];
    connection [Front_Fuselage.b, Vx.frame];
    connection [Vx.V, product.u1, product.u2];
    connection [product.y, gain.u, gain1.u];
    connection [gain1.y, Trainee._in];
    connection [gain.y, product1.u1];
    connection [Desired_Height._out, pid.u_s];
    connection [Height_Y.y, pid.u_m];
    connection [pid.y, product1.u2];
    connection [product1.y, Portance._in];
end
```

C.7 Comparison results

This section contains the comparison results after the last step of the SmartSync comparison of the MBSE and MBSA models for the blood delivery drone. Each row contains one element of the system and its representations in each model if it exists.

The first column indicates the element type, which can be either a port or a block. The inscription "MCB" indicates that the SmarSync tool output "`Missing Corresponding Block`" for this alignment; this is because the tool currently does not allow to align two model elements if their parents are not aligned.

The second row contains the name of the model element in the MBSE model.

The third row contains the name of the model element in the MBSA model.

These results were used to generate the graphical views of the categories shown in Section 5.2.2.

Type	Model1	Model2
	main. Alternative_1	main. ZippyFlyer
port	main. Alternative_1. Cell. Cargo_bay. Door_Cargo_bay. Open_signal	main. ZippyFlyer. Cell. Cargo_bay. Door. in

port	main. Alternative_1. Cell. Cargo_bay. Actuators. Energy	main. ZippyFlyer. Cell. Cargo_-bay. Actuator. inBattery
port	main. Alternative_1. Cell. Cargo_bay. Actuators. Open_-bay_signal	main. ZippyFlyer. Cell. Cargo_-bay. Actuator. inCalculator
port	main. Alternative_1. Cell. Cargo_bay. Actuators. Open_-signal	main. ZippyFlyer. Cell. Cargo_-bay. Actuator. out
port	main. Alternative_1. Avionics. Comms_module. Data_out	main. ZippyFlyer. Avionics. Comms_module. outData
port	main. Alternative_1. Avionics. Comms_module. Information	main. ZippyFlyer. Avionics. Comms_module. inCalculator
port	main. Alternative_1. Avionics. Comms_module. 4G_network	main. ZippyFlyer. Avionics. Comms_module. in4G
port	main. Alternative_1. Avionics. Comms_module. Information	main. ZippyFlyer. Avionics. Comms_module. outCalculator
port	main. Alternative_1. Avionics. Geolocation_module. Signaux_-geolocalisation	main. ZippyFlyer. Avionics. Module_GPS. in
port	main. Alternative_1. Avionics. Geolocation_module. position	main. ZippyFlyer. Avionics. Module_GPS. out
block	main. Alternative_1. Cell. Cargo_bay. Door_Cargo_bay	main. ZippyFlyer. Cell. Cargo_-bay. Door
block	main. Alternative_1. Cell. Cargo_bay. Actuators	main. ZippyFlyer. Cell. Cargo_-bay. Actuator
port	main. Alternative_1. Power_-unit. Propeller_Rear. Action	main. ZippyFlyer. Power_unit. Propeller2. in
port	main. Alternative_1. Power_-unit. Propeller_Front. Action	main. ZippyFlyer. Power_unit. Propeller1. in
port	main. Alternative_1. Power_-unit. Motor_Rear. Action	main. ZippyFlyer. Power_unit. Motor2. out
port	main. Alternative_1. Power_-unit. Motor_Rear. pilotage	main. ZippyFlyer. Power_unit. Motor2. in
port	main. Alternative_1. Power_-unit. Motor_Front. Action	main. ZippyFlyer. Power_unit. Motor1. out
port	main. Alternative_1. Power_-unit. Motor_Front. pilotage	main. ZippyFlyer. Power_unit. Motor1. in
port	main. Alternative_1. Battery. Ener_elec	main. ZippyFlyer. Battery. in
port	main. Alternative_1. Battery. Energy	main. ZippyFlyer. Battery. out

block	main. Alternative_1. Avionics. Comms_module	main. ZippyFlyer. Avionics. Comms_module
block	main. Alternative_1. Avionics. Geolocation_module	main. ZippyFlyer. Avionics. Module_GPS
port	main. Alternative_1. Calculator. Commande_de_vol	main. ZippyFlyer. Calculator. outComm
port	main. Alternative_1. Calculator. Energy	main. ZippyFlyer. Calculator. inBattery
port	main. Alternative_1. Calculator. Information	main. ZippyFlyer. Calculator. inComm
port	main. Alternative_1. Calculator. Open_bay_signal	main. ZippyFlyer. Calculator. outJettison
port	main. Alternative_1. Calculator. pilotage	main. ZippyFlyer. Calculator. outMotor
port	main. Alternative_1. Calculator. Flight_plan	main. ZippyFlyer. Calculator. inFlightPlan
port	main. Alternative_1. Calculator. position	main. ZippyFlyer. Calculator. inGPS
port	main. Alternative_1. Calculator. Trajectory	main. ZippyFlyer. Calculator. inInertial_measurement_unit
block	main. Alternative_1. Cell. Wing_r	main. ZippyFlyer. Cell. Wing
block	main. Alternative_1. Cell. Aileron_r	main. ZippyFlyer. Cell. Aileron1
block	main. Alternative_1. Cell. Aileron_r2	main. ZippyFlyer. Cell. Aileron2
block	main. Alternative_1. Cell. Cargo_bay	main. ZippyFlyer. Cell. Cargo_bay
port	main. Alternative_1. Inertial_measurement_unit. Orientation	main. ZippyFlyer. Inertial_measurement_unit. inAttitude
port	main. Alternative_1. Inertial_measurement_unit. position	main. ZippyFlyer. Inertial_measurement_unit. inGPS
port	main. Alternative_1. Inertial_measurement_unit. obstacle_position	main. ZippyFlyer. Inertial_measurement_unit. inRadar
port	main. Alternative_1. Inertial_measurement_unit. Trajectory	main. ZippyFlyer. Inertial_measurement_unit. out
block	main. Alternative_1. Power_unit. Propeller_Rear	main. ZippyFlyer. Power_unit. Propeller2
block	main. Alternative_1. Power_unit. Propeller_Front	main. ZippyFlyer. Power_unit. Propeller1

block	main. Alternative_1. Power_-unit. Motor_Rear	main. ZippyFlyer. Power_unit. Motor2
block	main. Alternative_1. Power_-unit. Motor_Front	main. ZippyFlyer. Power_unit. Motor1
port	main. Alternative_1. radar. Obstacle_image	main. ZippyFlyer. Radar. in
port	main. Alternative_1. radar. obstacle_position	main. ZippyFlyer. Radar. out
block	main. Alternative_1. Battery	main. ZippyFlyer. Battery
block	main. Alternative_1. Avionics	main. ZippyFlyer. Avionics
block	main. Alternative_1. Calculator	main. ZippyFlyer. Calculator
block	main. Alternative_1. Cell	main. ZippyFlyer. Cell
block	main. Alternative_1. Inertial_-measurement_unit	main. ZippyFlyer. Inertial_-measurement_unit
block	main. Alternative_1. Power_unit	main. ZippyFlyer. Power_unit
block	main. Alternative_1. radar	main. ZippyFlyer. Radar

APPENDIX D

CATEGORY VISUALISATION SCRIPT

This appendix contains the script used to generate categories and display them based on the CSV files output by the SmartSync tool.

The `csv2cat` method outputs a 3-tuple containing the category for model 1, the category for model 2 and the correspondance matrix between the models elements. The input for this method is the csv file output by SmartSync.

The `ListToCat` method is used by csv2cat to transform a list of model elements into a S2ML model category.

Finally the `afficherCat` allows to display a S2ML model category.

This script uses the NetworkX python package, which allows to represent and display graphs. By overloading the graphs we are able to make them contain enough information to represent the S2ML model categories.

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

def csv2cat(aFile):
    theFile = open(aFile, 'r').readlines()
    theCat1Elements = []
    theCat2Elements = []
    for theElement in theFile:
        theObject = theElement.split(';')
        if theObject[0].strip() != 'Type':
            if theObject[1].strip() != "forget" and theObject[1].strip() != '':
                if theObject[0].strip() != '':
                    theCat1Elements.append((theObject[1].strip().replace('main.', ''),
                        theObject[0].strip()))
                else:
                    theCat1Elements.append((theObject[1].strip().replace('main.', ''), 'block'))
            if theObject[2].strip() != "forget" and theObject[2].strip() != "":
                if theObject[0].strip() != '':
                    theCat2Elements.append((theObject[2].strip().replace('main.', ''),
                        theObject[0].strip()))
                else:
                    theCat2Elements.append((theObject[2].strip().replace('main.', ''), 'block'))
    theCat1Elements.sort()
    theCat2Elements.sort()
    theCat1 = ListToCat(theCat1Elements)
    theCat2 = ListToCat(theCat2Elements)
    theFunctor = np.zeros((len(theCat1[0].nodes()), len(theCat2[0].nodes())))
```

```
    for theLine in theFile:
        theElementCouple = theLine.split(';')
        if theElementCouple[0] != 'Type' and theElementCouple[1].strip() != '' and
             theElementCouple[1].strip() != 'forget' and theElementCouple[2].strip() != 'forget'
             and theElementCouple[2].strip() != '':
            theFunctor[list(theCat1[0].nodes()).index(theElementCouple[1].strip().replace('main.',
                 '')), list(theCat2[0].nodes()).index(theElementCouple[2].strip().replace('main.',
                 '').replace('\n', ''))] = 1
    for theLine in theFile:
        theElementCouple = theLine.split(';')
        if theElementCouple[1].strip() == 'forget':
            for i in range(0, len(list(theCat1[0].nodes())) - 1):
                theFunctor[i,
                     list(theCat2[0].nodes()).index(theElementCouple[2].strip().replace('main.',
                     '').replace('\n', ''))] = -1
        if theElementCouple[2].strip() == 'forget':
            for i in range(0, len(list(theCat2[0].nodes())) - 1):
                theFunctor[list(theCat1[0].nodes()).index(theElementCouple[1].strip('
                     ').replace('main.', '')), i] = -1
    return (theCat1, theCat2, theFunctor)

def ListToCat(aListofElements):
    theCat = nx.DiGraph()
    theLabels = {}
    theLabelsofEdges = {}
    theBlocks = []
    thePorts = []
    theConnections = []
    thePortsofConnections = []
    theColorMap = []
    theColorMapofEdges = []
    ncon = 0
    n = 0
    theColors = []
    theNodes = []
    theTypes = {}
    for a in aListofElements:

        if a[1] == "port":
            theCat.add_node(a[0], color = "lightblue")
            thePorts.append(a[0])
            theNodes.append(a[0])
            theTypes[a[0]] = "port"
        elif a[1] == "block":
            theCat.add_node(a[0], color = "green")
            theBlocks.append(a[0])
            theNodes.append(a[0])
            theTypes[a[0]] = "block"
        elif a[1] == "connection":
            ncon += 1
            connection = a[0].split(' ')
            connection[0] = connection[0]+".connection"+str(ncon)
            theCat.add_node(connection[0], color = "red")
            theConnections.append(connection[0])
            thePortsofConnections.append(connection)
            theNodes.append(connection[0])
            theTypes[connection[0]] = "connection"
    nx.set_node_attributes(theCat, theTypes, name = "type")

    for node in theCat:
        word = node.split(".")
        if node not in theConnections:
            theLabels[node] = word[len(word) - 1]
        else:
            if len(word[len(word) - 1]) == 12 :
                theLabels[node] = 'c' + word[len(word) - 1][10] + word[len(word) - 1][11]
            else :
                theLabels[node] = 'c' + word[len(word) - 1][10]
        target = ''
        for i in range(0, len(word)-1):
            if target == '':
                target = word[i]
            else:
```

```
                target = target + "." + word[i]
        if target != '':
            theCat.add_edge(node, target, color = 'purple')
        theLabelsofEdges[(node, target)] = "parent"
        #theColorMapofEdges.append('purple')

    for con in thePortsofConnections:
        for i in range(1,len(con)):
            theCat.add_edge(con[i], con[0], color = 'r')
            theLabelsofEdges[(con[i], con[0])] = "appartient"
            #theColorMapofEdges.append('red')
    theColorMapofEdges = [theCat[u][v]['color'] for u,v in theCat.edges()]

    for node in theCat:
        if node in thePorts:
            theColorMap.append('lightblue')
            theColors.append(node)
        elif node in theBlocks:
            theColorMap.append('green')
            theColors.append(node)
        elif node in theConnections:
            theColorMap.append('red')
            theColors.append(node)
    return (theCat, theColorMap, theColorMapofEdges, theLabels, theTypes)

def afficherCat(aCat, aColor_map, aEdge_color_map, aListofLabels):
    thePositions = nx.networkx.drawing.nx_pydot.graphviz_layout(aCat, prog='twopi')
    nx.draw(aCat, thePositions, node_color = aColor_map, edge_color = aEdge_color_map)
    #nx.draw_networkx_labels(aCat, thePositions, labels = aListofLabels)
    plt.show()
```

RÉSUMÉ ÉTENDU EN FRANÇAIS

Dans un contexte où les systèmes sont de plus en plus complexes, il a fallu revoir la façon dont ils sont modélisés pour prendre en compte la globalité de leurs caractéristiques. Ainsi, dans le domaine de l'ingénièrie système, les approches documentaires classiques sont vouées à être remplacées par l'approche dite MBSE, pour Model-Based System Engineering, l'ingénièrie système basée sur les modèles. Ces modèles sont constitués de divers diagrammes qui représentent les exigences vis à vis du système, son environnement, son cycle de vie, ses fonctions ou encore son architecture. Cette thèse se place dans un contexte aéronautique, dans lequel il est crucial de démontrer la sûreté des systèmes avant de pouvoir les mettre sur le marché. Pour cela les approches MBSA – Model-Based Safety Assessment – d'analyse de sûreté basée sur les modèles propose une vision moins abstraite que les approches traditionnelles telles que les arbres de défaillance, elle permettent d'appréhender des phénomènes complexes tels que les reconfigurations.

Évidemment, ces modèles n'ont d'intérêt pour la conceptions que si ils sont justes. Si cette affirmation nous paraît trivial, il n'est pour autant pas toujours facile de vérifier que les modèles ne contiennent pas d'erreur. Dans cette thèse nous nous intéressons au cas de la cohérence de ceux-ci entre eux. Il s'agit de vérifier qu'il n'y ait pas, dans un modèle d'information qui en contredise un autre, c'est à dire que les modèles représentent bien le même système.

Avant toute chose, il est important de bien comprendre que toute difference entre les modèles n'est pas une incohérence, dans ce manuscrit nous identifions expérimentalement trois sources de différences entre les modèles:

- Les différences dues aux outils et pratiques de modélisation;
- Les différences dues aux intentions de modélisation;
- Les différences dues aux erreurs de modélisation.

Seules ces dernières doivent être éliminées, les autres étant cruciales a une

modélisation utile et efficace, en effet chaque modèle a pour but de modéliser un comportement différent du système, dans un but précis.

Des méthodes dites de synchronisation des modèles ont étés développées dans le but de détecter les incohérences entre modèles MBSE et MBSA, et de les éliminer. Dans ce travail nous nous appuyons sur la méthodologie SmartSync, qui utilise le langage de modélisation S2ML comme formalisme pivot dans lequel les modèles sont comparés. Au travers d'une étude de cas, nous montrons la nécessité de la synchronisation, ainsi que certaines limites actuelles de la méthodologie. Nous nous proposons alors de définir un cadre mathématique qui permettre d'apporter une interprétation formelle aux méthodologies de synchronisation, de manière a pouvoir démontrer les résultats qu'elles renvoient, et éventuellement proposer des pistes d'amélioration.

Pour cela on définit une manière de représenter tout modèle S2ML sous forme de catégorie – un artefact mathématique dans lequel on distingue des objets et des morphismes qui sont des relations entre les objets. A partir de là, on propose d'appeler S2ML+Cat la catégorie dans laquelle les objets sont les modèles S2ML, et les morphismes sont les injections entre modèles S2ML, une relation qui indique qu'un modèle est contenu dans un autre. En construisant une structure dans laquelle on élimine les éléments de modèle que l'on ne souhaite pas comparer, puis ceux qui sont différents entre deux modèles, on donne une définition dans S2ML+Cat de ce qu'on appelle une relation binaire de cohérence. Finalement nous appliquons ce cadre mathématique sur SmartSync au travers d'un second cas d'étude afin de montrer que S2ML+Cat nous permet bien de modéliser une méthodologie de synchronisation. On est alors capable de suggérer quel est le résultat de smartsync dans le cas général, en précisant des améliorations possibles en réactions aux limitations que nous avions observées.

Nous utilisons également d'autres langages de modélisation, tels que Modelica et SCOLA, cela nous permet de démontrer que notre cadre mathématique, ainsi que la méthodologie SmartSync, ne permettent pas seulement de comparer des modèles SysML et AltaRica 3.0 comme nous l'avions originalement fait, où des modèles MBSE et MBSA, mais plus globalement tous modèles qui fasse apparaître l'architecture du système.

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Titre: Théorie des catégories pour la cohérence des modèles multi-niveaux systèmes (MBSE) et sûreté de fonctionnement (MBSA)
Mots clés: Ingénierie système, Sûreté de fonctionnement, MBSE, MBSA, Cohérence, Théorie des catégories

Résumé: La thèse s'intéresse au sujet de la cohérence des modèles d'architecture système (MBSE) et de sûreté de fonctionnement (MBSA). En effet, si ces modèles s'attachent à représenter un système à deux intentions différentes, d'un côté la représentation de son architecture, et de l'autre la validation de sa sûreté, il est nécessaire de prouver qu'ils représentent effectivement tous deux le même système, c'est-à-dire qu'ils sont cohérents l'un avec l'autre. Il existe différentes méthodologies qui cherchent à vérifier la cohérence de ces modèles ainsi qu'à les synchroniser. Dans ces travaux de thèse, nous proposons un cadre mathématique dans lequel on peut représenter ces modèles, ainsi qu'une définition de ce qu'est de manière générale une relation de cohérence. L'objectif est de proposer à travers ce cadre, une façon mathématique de représenter les méthodologies de synchronisation de modèles MBSE et MBSA, de manière à contribuer à la démonstration mathématique de l'efficacité de ces méthodes. Cette thèse est effectuée dans le cadre du projet S2C (https://www.irt-systemx.fr/projets/s2c/), projet collaboratif entre l'IRT SystemX (organisme d'hébergement de la thèse) et l'IRT St Exupery. Ce projet est réalisé en partenariat avec des entités académiques, industrielles et institutionnelles, en particulier intéressées par le domaine de l'aéronautique.

Title: Category theory for consistency between multilevel system modeling (MBSE) and safety (MBSA)
Keywords: System engineering, Safety assessment, MBSE, MBSA, Consistency, Category theory

Abstract: The thesis focuses on the topic of system architecture to safety analysis models consistency. Those models aim at representing a same system towards two different modeling intents, representation of the architecture and safety validation of the system. It is therefore necessary to show that both models indeed represent the same system, namely that they are consistent with each-other. There exist different methodologies that allow for consistency assertion and synchronisation of models. In this work we propose a mathematical frame within which such models can be represented, and a general definition of what is a consistency relation between such mathematical objects. This intention of this thesis is to provide means to formally represent MBSE/MBSA models synchronisation methodologies and to do mathematical proofs that such a methodology is efficient. This thesis is part of the S2C project (https://www.irt-systemx.fr/projets/s2c), which is a collaborative project between IRT SystemX and IRT St Exupery. This project is done in partnership with academic, industrial and institutional entities.