Safety analysis integration in a systems engineering approach for mechatronic systems design

Faïda Mhenni

THÈSE

Présentée par Mme Faïda MHENNI

Pour l'obtention du

GRADE DE DOCTEUR

École Doctorale : École Centrale Paris (ED287)

Spécialité : **Sciences pour l'ingénieur**

Laboratoire d'accueil : **LISMMA (EA2336)**

VERS UNE APPROCHE INTÉGRÉE D'ANALYSE DE SÛRETÉ DE FONCTIONNEMENT DES SYSTÈMES MÉCATRONIQUES.

Safety Analysis Integration in a Systems Engineering Approach for Mechatronic Systems Design

---

Soutenue le 12/12/2014

Devant un jury composé de :

M. Alain RIVIERE	Professeurs des Universités	Supmeca – LISMMA, Saint-Ouen
M. Hubert KADIMA	Enseignant-chercheur, HDR	EISTI – L@RIS, Cergy-Pontoise
M. Hamid DEMMOU	Professeurs des Universités	Université de Toulouse – LAAS, Toulouse
M. Omar HAMMAMI	Professeur, HDR	ENSTA ParisTech U2IS, Palaiseau
M. Stanislao PATALANO	Maître de Conférences, HDR	University of Naples Federico II -Naples
M. Antoine RAUZY	Professeur, Directeur de la Chaire Blériot-Fabre	Centrale Supélec, Châtenay-Malabry
Mme Nga NGUYEN	Enseignant-chercheur	EISTI – L@RIS, Cergy-Pontoise
M. Jean-Yves CHOLEY	Maître de Conférences	Supmeca – LISMMA, Saint-Ouen
M. Wassim ABIDA	Technical Manager, Docteur, ingénieur	UTC AEROSPACE SYSTEMS, Buc

---

Année 2014 École Doctorale Sciences pour l'Ingénieur – ED287
Numéro d'Ordre : 2014ECAP0062 École Centrale Paris

“ A good decision made quickly is much better than a perfect decision made too late.”

Duane Kritzinger

Aircraft system safety, 2006

ECOLE CENTRALE DE PARIS

Abstract

Department or School Name

Doctor of Philosophy

Safety Analysis Integration in a Systems Engineering Approach for Mechatronic Systems Design

by Faïda MHENNI

Modern systems are getting more complex due to the integration of several interacting components with different technologies in order to offer more functionality to the final user. The increasing complexity in these multi-disciplinary systems, called mechatronic systems, requires new appropriate processes, tools and methodologies for their design, analysis and validation whilst remaining competitive with regards to cost and time-to-market constraints.

The main objective of this thesis is to contribute to the integration of safety analysis in a SysML-based systems engineering approach in order to make it more efficient and faster. To achieve this purpose, we tackled the following axes: formalizing a SysML-based design methodology that will be the support for safety analyses; providing an extension of SysML in order to enable the integration of specific needs for mechatronic systems modeling as well as safety concepts in the system model; allowing the automated exploration of the SysML models in order to extract necessary information to elaborate safety artifacts (such as FMEA and FTA) and the semi-automated generation of the latters. We have also integrated formal verification to verify if the system behaviors satisfy some safety requirements. The proposed methodology named SafeSysE was applied to case studies from the aeronautics domain: EMA (Electro Mechanical Actuator) and WBS (Wheel Brake System).

Acknowledgements

The work presented in this thesis is done in collaboration between the LISMMA Laboratory of SUPMECA and the L@RIS laboratory of EISTI.

I am indebted to many people who both directly and indirectly contributed to the achievement of this thesis.

First and foremost, I am deeply indebted to my supervisors. I would like to thank Professor Alain Rivière, head of SUPMECA and Professor Hubert Kadima, head of the L@RIS laboratory for supervising this work and for their valuable advice and encouragement during the fulfillment of this thesis. I am also very grateful for the help and friendship of Nga Nguyen and Jean-Yves Choley with whom I spent long work sessions and fruitful discussions that led to the fulfillment of this work.

I would sincerely like to thank all the members of my thesis committee, particularly Professor Omar Hammami and Professor Hamid Demmou who have accepted to spend their precious time to report my dissertation. I believe their questions and comments will help me in the final improvement of the thesis. I am particularly grateful for Professor Omar Hammami for his encouragement and the interesting discussions I have had with him. He gave me valuable guidelines at the beginning of this work that was of great help to me during all the duration of my work. My sincerest thanks also to Professor Antoine Rauzy and Professor Stanislao Patalano who have accepted to be in my thesis committee. I would particularly like to thank Professor Antoine Rauzy for organizing an open seminar on safety that constituted an opportunity to discuss interesting subjects and ask the right questions. I'm also honored by the presence of Doctor Engineer Wassim Abida, UTC-AS among the thesis committee.

I would also like to thank Flavien Raynaud and Thomas Papillon, students at EISTI, who implemented the prototype tool of SafeSysE.

I am especially grateful for Christel Compagnon, the LISMMA laboratory assistant for her help in the administrative tasks as well as for her encouragement mainly during the final steps of the achievement of this thesis.

Last but not least, I would like to thank all my family members for their encouragements and their endless support. They are the secret behind my success.

Contents

List of Figures

Abbreviations

AADL	**A**rchitecture **A**nalysis and **D**esign **L**anguage
AD	**A**ctivity **D**iagram
BSA	**B**ehavioral **S**afety **A**nalysis
BDD	**B**lock **D**efinition **D**iagram
CAD	**C**omputer **A**ided **D**esign
CCA	**C**ommon **C**ause **A**nalysis
CMA	**C**ommon **M**ode **A**nalysis
CSA	**C**ompositional **S**afety **A**nalysis
DoDAF	**D**epartment **o**f **D**efense **A**rchitecture **F**ramework
DRBD	**D**ynamic **R**eliability **B**lock **D**iagram
DSL	**D**omain **S**pecific **L**anguage
EFFBD	**E**nhanced **F**unctional **F**low **B**lock **D**iagram
EMA	**E**lectro **M**echanical **A**ctuator
FAST	**F**unction **A**nalysis **S**ystem **T**echnique
FFBD	**F**unctional **F**low **B**lock **D**iagram
FHA	**F**unctional **H**azard **A**ssessment
FMEA	**F**ailure **M**ode and **E**ffects **A**nalysis
FMECA	**F**ailure **M**odes, **E**ffects and **C**riticality **A**nalysis
FMES	**F**ailure **M**ode and **E**ffects **S**ummary
FSAP	**F**ormal **S**afety **A**nalysis **P**latform
FTA	**F**ault **T**ree **A**nalysis
HAZOP	**HAZ**ard and **OP**erability Analysis
HiP-HOPS	**Hi**erarchically **P**erformed **H**azardds **O**rigin and **P**ropagation **S**tudies
IBD	**I**nternal **B**lock **D**iagram

MARTE	**M**odeling and **A**nalysis of **R**eal-time **E**mbedded Systems
MBSA	**M**odel **B**ased **S**afety **A**nalysis
MBSE	**M**odel **B**ased **Sy**stems **E**ngineering
MODAF	**M**inistry **O**f **D**efense **A**rchitecture **F**ramework
OMG	**O**bject **M**anagement **G**roup
PHA	**P**reliminary **H**azard **Analysis**
PRA	**P**articular **R**isk **Analysis**
PSSA	**P**reliminary **S**ystem **S**afety **A**ssessment
RBD	**R**eliability **B**lock **D**iagram
RPN	**Risk Priority Number**
SADT	**S**tructured **A**nalysis and **D**esign **T**echnique
SafeSySE	**Safe**ty Analysis Integration in **S**ystems **E**ngineering Approach
SD	**S**equence **D**iagram
SIL	**S**afety **I**ntegrity **L**evel
SSA	**S**ystem **S**afety **A**ssessment
SysML	**Sys**tems **M**odeling **L**anguage
UPDM	**U**nified **P**rofile for **D**oDAF/**M**ODAF
UML	**U**nified **M**odeling **L**anguage
ZSA	**Z**onal **S**afety **A**nalysis

Dedicated to my lovely parents

Introduction

In nowadays world, we are surrounded by systems of different kinds and used for different purposes. We use systems for household, transportation, communication, work, machining, etc. They became a necessity for modern living. Ambitious and always looking for more and better as we are, we require more and more services to the systems around us. When using our cars, we need driving assistance to make less effort, listening to radio or music for distraction, GPS guidance to reach our destination, traffic information to avoid jam, and some projects are working on autonomous driving to allow drivers to do anything they want during their journey, except driving. Achieving a big number of functions by the same system requires the integration of several components of different technologies that communicate in a synergistic way in order to achieve the system missions. This results in an increasing complexity in manufactured systems.

In addition to the increasing complexity, the current industrial context is characterized with sharp competitiveness constraints. This leads to the integration of new functionality and enhancing the performances of the designed systems and consequently increasing the complexity even more. The competitive context also imposes shortening time-to-market and reducing costs while delivering reliable and efficient products.

With systems and technology also comes the exposure to mishaps as systems can fail or perform improperly resulting in damage, injury, and deaths. Using systems is not risk-less and implies to accept a certain exposure to risks. For safety critical domains, rigorous standards and regulations specify the acceptable risk level and impose precise constraints on the system design and verification methods to validate the designed system. The non-respect of such standards leads inevitably to unqualified systems that are not allowed to be used.

Safety critical systems then require a rigorous validation of their behavior and performances. The potential risks of such systems must be thoroughly identified and guarded against during the development cycle to bring them to an accepted level. The development of these systems is then challenging and requires adequate systems engineering approaches as well as rigorous safety analysis techniques in order to manage the complexity and satisfy performance and safety requirements.

Usually, safety analyses are very long to carry out and take an important part of the whole development process. They also require a deep knowledge of the technical and safety domains. Traditionally performed with separate tools, it's hard to maintain them consistent with the system model that continues to evolve during these analyses. For a successful use, safety analysis techniques should be integrated efficiently in the design process. This can only be done with the use of adequate tools and methods to facilitate ensuring consistency between the two domains.

Model-Based Systems Engineering (MBSE) approach seems a good candidate for the design of safety critical complex systems. So, in this work, we first focused on establishing an MBSE approach that will be the basis for the next steps. Our aim is then to efficiently integrate safety analyses within the MBSE design approach in order to bridge the gap between the two approaches by enhancing communication and consistency.

A review of MBSE approaches reveals that, SysML, the OMG systems modeling language is being widely used to support these approaches. Indeed, SysML allows to express the main concepts inherent to the different aspects of system development. The language is used to build a system model used as a common reference for all the domains. This model captures all the relevant information and harmonizes the whole process.

In this work, we have analyzed the capability of SysML to express specific concepts related to safety, and the way SysML models can be explored to provide relevant information for safety analysis. We noticed that some concepts very useful for complex systems modeling and for safety analysis are not explicitly supported in SysML. We have also noticed the lack of well established MBSE approach based on SysML. However, we believe that only a well established MBSE methodology can successfully support other analyses like safety assessment.

To tackle these issues, this thesis is organized as follows.

Chapter 1 introduces this work by presenting the state of the art about the main concepts that are useful for this thesis. First, it defines the complexity and identifies its causes to demonstrate the challenging aspect of designing complex systems and thus highlight the need for systems engineering approach. Then, the chapter presents the key concepts of systems engineering. The second part of the chapter is dedicated to safety analysis. In this part, the main safety analysis techniques are presented. Finally, the chapter gives a state of the art about the integration of safety analyses within the systems engineering approaches.

Chapter 2 first presents the OMG languages UML and SysML respectively and discusses their ability to model the concepts needed for this work. The extensibility mechanisms of these languages allow their extension to integrate new domain specific concepts. In this work, we added some extensions to SysML to better support safety analysis and also to add some concepts relevant to complex mechatronic (multi-disciplinary) systems. A SysML safety profile and a mechatronic extended modeling profile are then developed for this purpose. To better define the way of integrating safety analysis in the whole MBSE process, we felt the need to first establish a well defined design methodology. This feeling is even enforced by the lack of SysML-based methodologies in the literature except some very software oriented ones. A SysML-based methodology that constitutes the support for the whole process is then presented in this chapter. Finally, the integrated SafeSysE methodology is detailed. In this methodology, developments are done to facilitate the extraction of relevant information and the automated generation of some safety analysis artifacts. The next chapters give more detail about our contribution for each artifact.

Chapter 3 deals with the Failure Mode and Effects Analysis (FMEA). It first introduces the main concepts about FMEA. Then, it deals with the automated generation of FMEA from SysML models. Two different types of FMEAs are generated, the functional FMEA and the component FMEA. In the proposed process, a preliminary FMEA is automatically generated by exploring the SysML model and extracting the relevant information form it. The developed safety profile allows the system engineer to annotate the system model with safety relevant information like failure modes and their effects, probability of occurrence etc. The generated FMEA is filled with all relevant information automatically extracted from SysML model. Then, it is completed by the safety expert. The generation

of the preliminary FMEA and the extraction of data from the system model automated in our approach is a long and error prone task if done manually. Once the safety expert has completed the FMEA, the same process allows to feedback the system model with the FMEA results and store them in the system model. This allows to re-generate new FMEA when needed (when design changes occurred) without losing the safety expert's work. Consistency is also maintained between these two artifacts since the generated component FMEA includes a reminder of the functional FMEA results. The chapter then illustrates the approach with a case study.

Chapter 4 deals with the Fault Tree Analysis (FTA) technique. First, it introduces the main concepts about FTA. Then, our approach to automatically generate fault trees from SysML models is presented. In this work, we present a pattern-based approach, in which the tool explores the system internal structure given in a SysML IDB and identifies relevant patterns. The patterns are identified with respect to their interfaces (input and output ports) or via the attribution of specific stereotypes from the developed safety profile to components. A partial fault tree is generated for each pattern, and then all the generated sub-fault trees are assembled to form the resultant fault tree. In this approach, two levels of fault trees are generated. The first one is a generic fault tree that is based on the system topology. This fault tree describes fault propagation throughout the system. The second one is a specific fault tree dealing with a particular undesired event. The specific fault tree explores the component FMEA to only include the specific component failure modes leading to the undesired to event of the fault tree. The consistency is again maintained with previous analysis results. Fault trees cans be generated in two different formats. The fist one is graphic and is used to help understanding fault propagation in qualitative analysis. The second one is the Open-PSA format that allows performing further quantitative analyses. Again, with the automated generation based on the system models, we contribute to reducing error proneness and development time of safety analysis and thus of the whole process. The chapter also presents a case study to illustrate the approach.

Chapter 5 introduces the use of formal methods for behavioral safety assessment. It first gives on overview of the formal methods for safety analysis focusing on model checking. Then, it presents an approach for automatically generating NuSMV programs from SysML models. This approach details the mapping between SysML

models and the NuSMV program and shows how to extract the relevant information from the SysML model to automatically generate the different modules of the NuSMV file. Model checking is then used to automatically verify safety requirements that are captured in temporal logic format. The use of formal methods is very useful since it gives an automated verification of the specified properties and also provides counter-examples when the properties do not hold. Integrating behavioral safety analysis allows to have a comprehensive safety assessment dealing with both compositional and dynamic aspects. The proposed approach is also illustrated with a case-study.

Chapter 1

State of the Art

1.1 Introduction

The complexity of systems is considerably growing; systems are getting larger, more integrated, and involving a huge variety of components and technologies [7]. Software components are more and more embedded for control purposes in systems that already contain a diversity of other components of different domains such as electronic devices, sensors, actuators and mechanical structure [7, 8]. Well specified and adapted methods and tools are needed to manage the development but also the validation of the design of such systems.

Several systems are considered safety critical, i.e., their malfunctioning and/or failure could lead to serious safety issues. The development of safety critical systems requires to check that the system behaves safely not only in nominal situations, but also under certain degraded situations [9]. Several safety analysis techniques and methods are developed for this purpose and widely used in industry. Safety analysis aims at enhancing the design and making the manufactured systems safer. The integration of safety analyses during the design is then crucial [7].

The objective of this thesis is to contribute to the integration of safety analysis with the systems engineering design process by introducing an integrated methodology. This chapter introduces the basic concepts about the two main topics of this thesis: systems engineering and safety analysis. Its aim is to provide the basic knowledge necessary to understand the remaining parts of this work. It is organized as follows. First, an overview of systems engineering with its common definitions and

standards is given in section 1.2. Then, a summary about safety analysis is given in section 1.3 including an overview of the most used safety analysis techniques in section 1.3.1, and an introduction to safety standards in section 1.3.2. Related work about the integration of Model-Based Systems Engineering (MBSE) and Model-Based Safety Analysis (MBSA) is then given in section 1.4. Finally, the chapter is concluded in section 1.5.

1.2 Systems Engineering

Systems engineering is a subject that attracts a lot of attention in the last years and it is often seen as an emerging discipline [4]. Indeed, it is a trendy subject nowadays in terms of formalizing educational programs as well as industrial projects. However, it's not that recent in practice since without systems engineering we wouldn't have had several systems like the pyramids or more recently, space shuttles and aircraft that became part of our daily life. These systems have certainly been thoroughly planned and developed with a global view. In other terms, they have been systems engineered, and the example of the pyramids shows that it already existed long time ago. More recently, the technological progress in the end of the 19th and the beginning of the 20th centuries lead to an industrial growth. Consequently, manufactured systems are becoming progressively complex offering more and more functionality. With this increasing complexity of engineered systems and the trend to have more and more large scale systems, deep thinking about systems design and development led to concepts like systems engineering and systems theory. Complexity overwhelms traditional approaches and new appropriate approaches are needed to manage this complexity and keep designing successful systems.

Since the key concept behind the application of systems engineering is complexity, let's first have a look at what is complexity, what are the causes behind increasing complexity, how it affects design tasks and how to deal with it.

1.2.1 Complexity

There are several attempts to define complexity in several fields. A definition was given to complexity by Nancy Leveson [10] highlighting the fact that complexity

prevents the behavior of complex systems to be " thoroughly planned, understood, anticipated, and guarded against " because of the presence of some *unknowns* in predicting system behavior. Even though this definition was given from safety perspective, it is well suited for the other aspects of engineering. The critical factor that differentiates complex systems from other systems is intellectual manageability. Models are then needed to help in managing the complexity.

Complexity is also defined by the SAE-ARP-4754A [11] as "an attribute of functions, systems or items, which makes their operation, failure modes, or failure effects difficult to comprehend without the aid of analytic methods".

Several factors can increase complexity of systems and thus make their operation more difficult to comprehend like, for instance, the increasing number of components and relationships, the interactions among system elements, particularly the feedback loops but also the use of new and/or sophisticated technologies [10, 12]. When the interacting components are from different domains and exchange different kinds of flows (discrete vs continuous, data vs object etc.), complexity is even more increased.

The traditional approach to remedy the intellectual manageability due to complexity is based on the "divide and conquer" principle. This approach consists in breaking down the complex system into distinct sub-systems or parts of lower complexity. The decomposition can be performed either on the structural aspect or on the behavioral one [10]. In this approach, each sub-system or component is designed and developed separately by domain specific engineers and all the components are then assembled to form the system under study. This approach is usually able to provide a solution to the problem. However, its main drawback is that only local optimization can be achieved at the component level, but no global optimization is performed at the system level. Unfortunately, the sum of these local components does not lead necessarily to a global optimum at the upper (system) level. This approach is consequently unable to guarantee an optimal solution.

Another drawback is that the decomposition, if not thoroughly performed, could increase the complexity or lead to erroneous results. On the other hand, if done adequately, the decomposition could help in finding a solution. However, the decomposition is only possible under some conditions. Indeed, not all the systems can be decomposed. Before proceeding to a decomposition, it is necessary to make

sure that the results of the analyses performed on the components, independently of the system, are not distorted by the extraction of the components from the system context. This is true if the following assumptions are achieved [1]:

- The system does not contain feedback loops or non-linear interaction between components;
- The components have the same behavior whether considered inside the system or separately;
- The components assembly principles are simple enough so that the interactions between the components can be considered separately from the behavior of the latter.

As an attempt to determine when the decomposition is feasible, systems theory has classified systems into several classes according to their degree of complexity. There are three classes of systems: systems of organized simplicity, systems of organized complexity and systems of non-organized complexity. Each of these systems has a specific way to be dealt with [1].

Systems that can be decomposed easily, i.e., where the analytic reduction can be applied are called of **organized simplicity**. This class contains systems whose components can be isolated for analysis purposes without inducing errors in the analysis results. This class is well suited for mechanical systems for instance. Other systems are complex and cannot be decomposed to allow the analysis of the components independently, but their behavior is regular and random enough to allow studying them statistically. This class of complexity is called by systems theoreticians **non-organized complexity**. Even though the systems of this class are considered complex systems, their study remains simple since it can be dealt with by statistic laws. In physics, this class is embodied in statistical mechanics.

The third class contains the systems of **organized complexity**. These systems are too complex for being analyzed completely and too organized (do not have random behavior) to be dealt with using statistics. This is the case for biological and social systems for instance. It is also the case for a major part of the systems conceived after second world war, mainly those with complex embedded software [1].

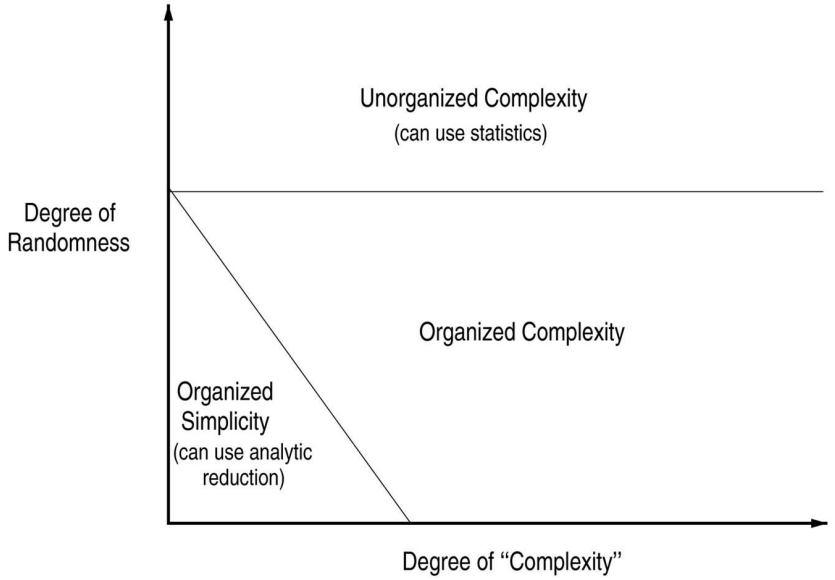


FIGURE 1.1: Three Categories of Systems [1]

The introduction of several disciplines interacting with each other inside the same system, and mainly the introduction of software and computer science lead to new types of dependency and non-linear interactions between components. Consequently, these systems cannot be decomposed and a systemic approach, considering the system as a whole, is compulsory. In the frame of this thesis we are mainly interested in this class of systems, that are multidisciplinary non-decomposable systems. We mainly focus on mechatronic systems that are within this class of systems. The complexity of modeling the behavior of mechatronic systems is due to the following two intertwined aspects [13]:

- The first aspect is the hybrid behavior modeling. As a consequence of their multidisciplinary and synergistic aspect, mechatronic systems often exhibit discrete behavior, time-continuous behavior, and hybrid behavior;
- The second aspect is the multiple domain integration: the various domains involved are interconnected with their proper specific characteristics.

Moreover, the two aspects of complexity are not independent. The association of hybrid behavior in various domains increases the complexity of behavioral modeling. Therefore, a system level design which considers the multi-disciplinary (including but not limited to mechanical, electrical/electronic and software) components simultaneously is needed at the early stages of the development of complex

mechatronic systems, hence the need for systems engineering approach.

1.2.2 Systems Engineering Definitions

This section aims at defining systems engineering to better identify its goals and processes. Several definitions are given in the literature, and each definition highlights a particular aspect. Some of the most relevant definitions are given hereafter.

In the American military standard Mil-Std-499A, 1974 [14], systems engineering is defined as:

" The application of scientific and engineering efforts to (a) transform an operational need into a description of system performance parameters and a system configuration through the use of an iterative process of definition, synthesis, analysis, design, test and evaluation; (b) integrate related technical parameters and assure compatibility of all physical, functional and program interfaces in a manner that optimizes the total system definition and design; (c) integrate reliability, maintainability, safety, survivability (including Electronic Warfare considerations), human and other such factors into the total technical engineering effort to meet cost, schedule and technical performance objectives."

In this definition, several key concepts appear such as **iterative process**, **compatibility** and the **global aspect** by the integration of a variety of factors like reliability, safety, human, etc.

This definition has been updated in the next version of the standard, Mil-Std-499B, 1993, to include broader concepts and focus on the **interdisciplinary** aspect. The definition of systems engineering becomes: "An interdisciplinary approach encompassing the entire technical effort to evolve and verify an integrated and lifecycle balanced set of system product and process solution that satisfy customer needs. Systems engineering encompasses: (a) the development, manufacturing, verification, development operations, support, disposal of, and user training for, system products and processes; (b) the management of the system configuration; (c) the translation of system definition into work breakdown structures; and (d) development of information for management decision making."

Another definition of systems engineering given in [15] is "an iterative process of top-down synthesis, development and operation of a real-world system that satisfies, in a near optimal manner, the full range of requirements for the system".

In this definition, the concept of **requirements** appears. Indeed, requirements management is a major point in systems engineering since the objective of this latter is the design and development of systems that satisfy a set of requirements. A good definition and analysis of requirements is of paramount importance for a successful system design. Requirements can be split into two major categories: functional requirements and non functional requirements. Functional requirements specify the different functions that the system must achieve. Non-functional requirements include different classes like performance, regulation, safety requirements to only cite a few. Requirements evolve during the whole process. They are progressively refined at different abstraction levels as system level requirements are declined into component level requirements and so on. Requirements engineering is a part of systems engineering that focuses on requirements management and deals with requirements derivation, consistency, and traceability [16].

Systems engineering also considers the triad cost, schedule and technical performance since the objective of all engineering projects is to provide a solution that achieves the set of requirements cheaper, faster and better.

The main characteristic of systems engineering is that it tackles the whole life-cycle of the system from the stakeholders need expression to the retirement or disposal of the system.

For a successful design of complex systems, systems engineering approach is required [17]. Systems engineering approach relies on a set of processes, methods and tools. A process is a logical sequence of tasks performed to achieve a particular objective. A process defines **WHAT** is to be done, without specifying **HOW** each task is performed. A method consists of techniques for performing a task, i.e. it defines the **HOW** of each task. A tool is an instrument that, when applied to a particular method, can enhance the efficiency of the task. It gives an answer to the question **WITH WHAT**. For an optimal result, it is necessary to formalize the approach by the definition of appropriate processes, methods and tools.

To formalize the processes, different systems engineering standards have been developed. The next section gives an overview of the most widely used standards.

1.2.3 Systems Engineering Processes - Standards

1.2.3.1 IEEE 1220

IEEE 1220 [18] focuses on the technical processes of systems engineering from the requirements analysis to the definition of the physical system. The standard describes in detail three main processes on which the systems engineering activities rely:

- Requirements analysis;
- Functional analysis;
- Allocation and synthesis.

Each one of these processes is validated before going to the next process. The standard also specifies the inputs and outputs of each process 1.2.

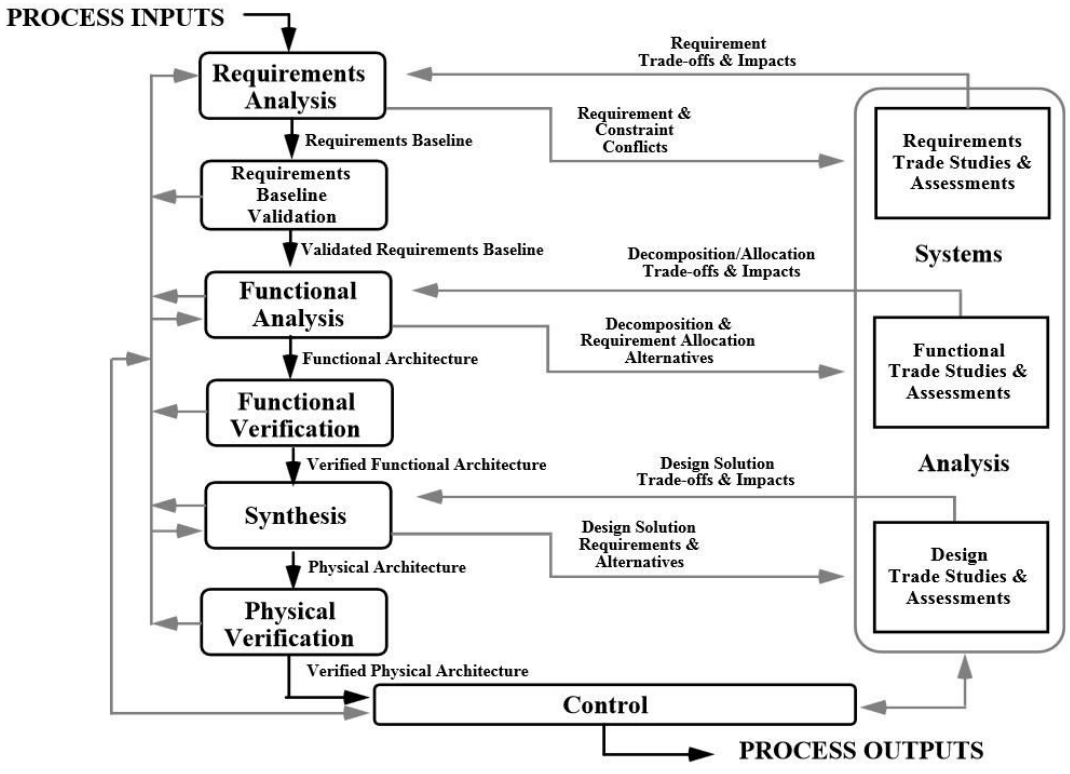


FIGURE 1.2: The IEEE 1220 Processes

1.2.3.2 EIA 632

EIA 632 [19] covers a larger scope than the IEEE 1220 by supplementing the system design processes with the contract (acquisition and supply) processes, the technical management processes, the product implementation processes as well as the technical evaluation processes. An overview of these processes and the way they are related to each other is given in Figure 1.3.

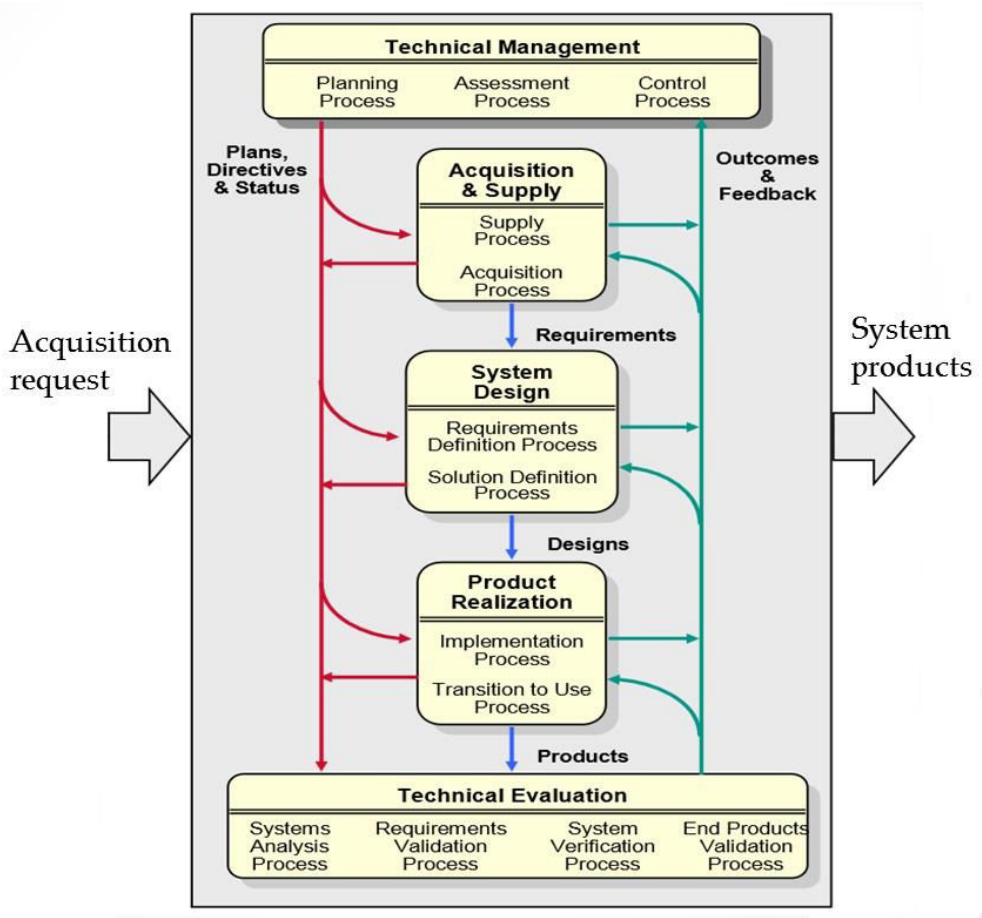


FIGURE 1.3: The EIA 632 Processes

1.2.3.3 ISO/IEC 15288

The ISO/IEC 15288 standard covers a larger scope than the two previous ones and tackles the whole system life-cycle processes [20]. It was inspired from the ISO/CEI 12207–AFNOR Z 67-150 (Typology of software life-cycle processes). It extends the technical processes to the whole system life-cycle, covering operations, operational maintenance and disposal at the end-of-life processes. It also covers

project processes, enterprise processes and agreement (acquisition and supply) processes (Figure 1.4).

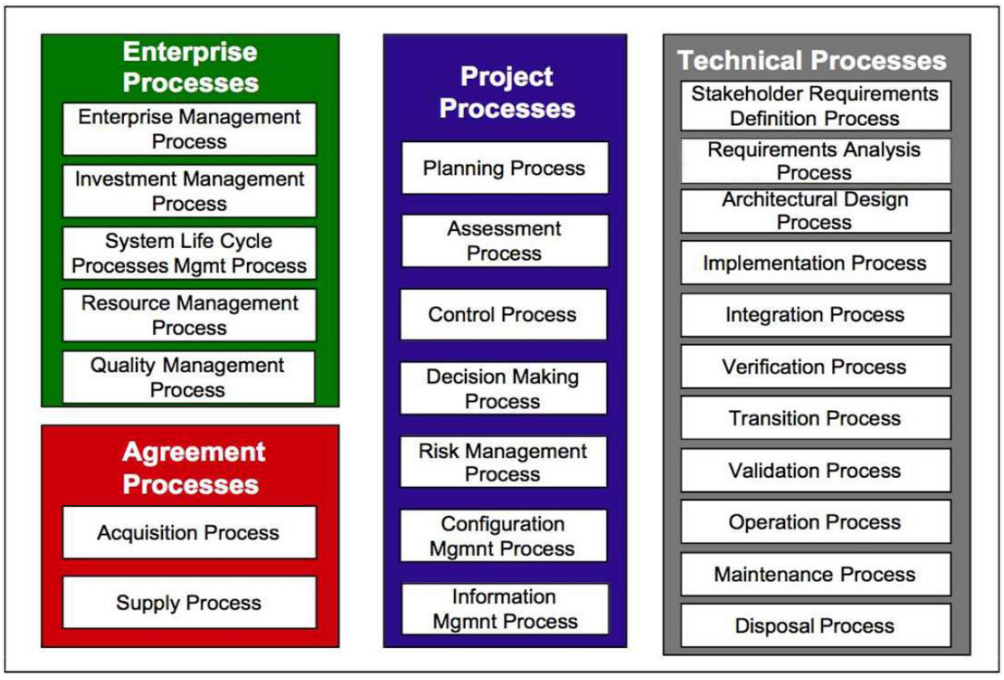


FIGURE 1.4: The ISO/IEC 15288 Processes

The three standards have the technical processes describing the system definition in common but only the ISO/IEC 15388 covers the whole life-cycle of the system (Figure 1.5).

A comparison between the different SE standards can be found in [21] and [20].

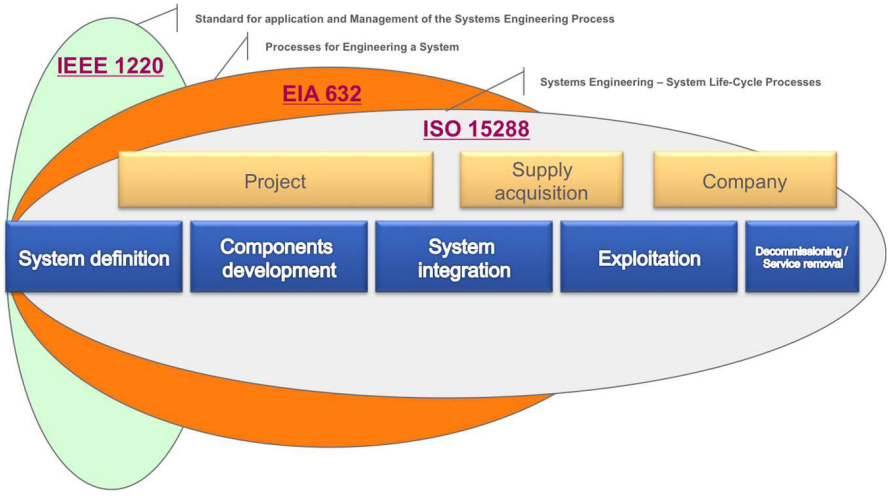


FIGURE 1.5: IEEE 1220, EIA 632 and ISO 15288 Standards

1.3 Safety Analysis

1.3.1 Safety Analysis Techniques and Methods

Safety analysis techniques and methods have the objective to assess the system safety during the design phase and ensure that the designed systems have satisfactory safety level. Several techniques and methods exist and they can be classified according to different criteria. For instance, they can be classified according to the type of models they are constructed upon, to the type of elements they are interested in (functions, components, etc.).

Safety analysis artifacts can be constructed from structural models as well as from the dynamic behavioral models of the system. In the first case, the safety analysis is qualified as compositional while in the second case they are qualified as behavioral [22]. Compositional safety analyses are based on the composition models of the system. They can be performed at the functional or component level. They only consider the static aspect of the system. On the other hand, behavioral safety analyses focus on the dynamic behavior of the system. For a comprehensive safety analysis however, the behavioral (dynamic) aspect should also be considered.

Safety analysis techniques can also be classified according to the life-cycle phase they are performed in. Some techniques can be performed since the preliminary design stages while others can only be performed in advanced stages when a detailed design has already been defined.

Different safety analysis methods exist and are used for different purposes and at different design stages. Among these methods we can cite Preliminary Hazard Analysis (PHA), System Hazard Analysis, Fault Tree Analysis (FTA), Event Tree Analysis, Failure Mode and Effects Analysis (FMEA), Functional Hazard Analysis (FHA), Petri Net Analysis or Markov Analysis. Each hazard analysis technique is a unique analysis methodology using specific guidelines and rules with an overall objective of identifying hazards, mitigating them, and assessing system residual risk. More detail about these hazard analysis techniques and methodologies can be found in a book written by Ericson [23].

To perform safety analyses, a number of interrelated analysis techniques and methodologies are usually used in a complementary way. The two most traditionally used techniques are Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA). FMEA aims to evaluate the effects of potential failure modes of components or functions, and eliminate these potential risks in the system design. FMEA is an inductive bottom-up safety analysis tool that identifies the failure modes of system functions or components and then determines their effects on the system level [23, 24]. Meanwhile, FTA, when used in a qualitative approach, is a top-down deductive analytic method in which the analysis starts from an undesired event called the *top level event* and then, the initiating primary events such as component failures, human errors, and external events are traced through Boolean logic gates to this top level event [23, 24]. FTA can also be used in a quantitative analysis and in this case, the probability of the top level event is evaluated based on the different probabilities of the leaf events of the fault tree [24]. Both of these techniques are compositional analyses based on the topology of the system and do not consider its dynamic behavior.

More detail about some of the techniques mentioned above is given in the next sections. As our study is mainly based on FMEA, FTA and Model checking for behavioral safety analysis, these techniques will be described in more detail in chapters 3, 4 and 5 respectively.

1.3.1.1 Failure Mode and Effects Analysis (FMEA)

Failure Mode and Effects Analysis (FMEA) is one of the first systematic techniques for failure analysis. It was elaborated in the 1950s by reliability engineers to study the problems that could arise from the occurrence of malfunctions in military systems. FMEA is considered as an inductive bottom-up tool for safety analysis [23, 25]. It aims at evaluating the effects of potential failure modes of components or functions on the system level, and then eliminating the identified potential risks in the system design. To perform an FMEA, each element of the system (function or component) is reviewed separately in order to identify its potential failure modes. The causes and effects of each failure mode are then determined. The FMEA is then presented as a table in which, for each function or component, the different failure modes, their causes and effects on both local and system levels are recorded. In an FMEA table, each line corresponds to an element and the

columns represent a set of properties of the failure mode like the causes, effects, etc. An example of FMEA table is given in Figure 1.6.

Component	Failure mode	Causal factors	Immediate Effect	Method of detection	System effect	Recommended Actions	Severity
Embedded MCU with Power Bridge							
MotRedCoder							
Mech Transmission							

FIGURE 1.6: Example of FMEA

The list of columns is not unique and can be adapted by each enterprise to better respond to its needs.

A more detailed version of the FMEA is the Failure Modes, Effects and Criticality Analysis (FMECA). This version develops three factors: *severity*, showing how serious the consequences of failures may be; *occurrence*, showing how frequently failures occur; and *detection*, showing how easily failures can be detected. Then the Risk Priority Number (RPN) is calculated by multiplying these three factors. FMECA is then performed by addressing problems from the biggest RPN to the smallest one [25].

1.3.1.2 Fault Tree Analysis (FTA)

Fault Tree Analysis (FTA) is a top-down deductive approach that aims to identify the root causes of an undesired event. As a deductive approach, FTA starts with a top-level undesired event, such as failure of a main engine, and then determines (deduces) its causes using a systematic, backward-stepping process [26].

In determining the causes, a fault tree (FT) is constructed as a logical illustration of how lower level events combine together to result in the upper level undesired event. The events contributing to the undesired event include component failures but may also include normal events if normal event combined with some failures may lead to the top level undesired event. They constitute the leaves of the tree and are linked together using logical operators. Different symbols are used to

indicate the different type of events and the type of relationships involved. Events are divided into several types: basic events, intermediate events, etc. Relationships are represented by logical gates, mainly AND and OR gates but other operators like Priority AND and Exclusive OR can also be used.

FTA allows both qualitative and quantitative analyses. Qualitative analysis is performed in order to identify the necessary and sufficient combination(s) of basic events that result in the occurrence of the undesired top event. These necessary and sufficient combinations are called minimal cut sets. The identification of minimal cut sets in a fault tree helps the designer to focus on the design weak points.

Quantitative analysis is performed to calculate the probability of the top event. The fault tree can also be quantified to provide useful information on the probability of the top event occurring and the importance of all the causes and events modeled in the fault tree [26]. The probabilistic computation for the quantitative analysis of fault trees can be performed by dedicated tools such as XFTA [27].

A fault tree can also be transformed into a success tree, the logical complement of a fault tree. A success tree shows how the undesired event can be prevented from occurring. The conditions provided in the success tree, if assured, guarantee that the undesired event will not occur. The success tree is then a valuable tool that provides equivalent information to the fault tree, but from a success viewpoint [26].

1.3.1.3 HAZard and OPerability Analysis (HAZOP)

HAZOP is an inductive, structured and systematic qualitative risk assessment tool [23]. It aims at identifying and analyzing potential hazards in a system as well as the operability problems that are likely to happen. It is applicable either for an existing or planned system or process and tackles the entire life-cycle of the system, i.e. from concept phase to decommissioning. It assumes that risk events are caused by deviations from design or operating intentions. The hazard identification process consists in identifying the deviations by comparing system parameters to a list of key guide words such as *more*, *less*, *no*, *reverse*, *late* and so forth, that are combined with process/system conditions or parameters such as *speed*, *flow*, *pressure*. Then, HAZOP analysis looks for hazards resulting from

identified potential deviations in design operational intent and the team decides whether the designed safeguards are satisfactory or additional actions are necessary to reach an acceptable level of risk. A HAZOP analysis is performed by a team of multidisciplinary experts in a brainstorming session under the leadership of a HAZOP team leader.

This technique can be used for both preliminary and detailed design. Although initially developed for the chemical process industry, the HAZOP analysis can be applied to any type of system and equipment, with analysis coverage given to subsystems, assemblies, components, software, procedures, environment, and human errors.

The main drawback of this approach is that its success heavily relies on the ability of the team in predicting deviations based on past experiences and general subject matter expertise.

1.3.1.4 Reliability Block Diagrams (RBD)

A Reliability Block Diagram (RBD) [28, 29] is a diagrammatic success-oriented method for describing a system with a focus on how the reliability of its components contributes to the success or failure of the global system. RBD model has been widely used due to its simplicity and is one of the most practical reliability modeling tools. An RBD model consists of an input point, an output point, and a set of blocks connected in parallel or in series between these two points. Each block represents a physical component that functions normally. When a component fails, the corresponding block is removed from the diagram. The whole system is operational if there is at least one path between the input and output points. When too many components are removed leading to interrupting the connection between input and output points, then the system is no longer operational. An example of RBD is given in Figure 1.7.

The main advantage of the RBD approach is its simplicity. Indeed, it can be easily used and understood by engineers from different disciplines as well as by project managers or any non-technical stakeholders. However, the main disadvantage of RBD is that it is unable to "capture the dependent and dynamic behaviors of large and complex system" [28].

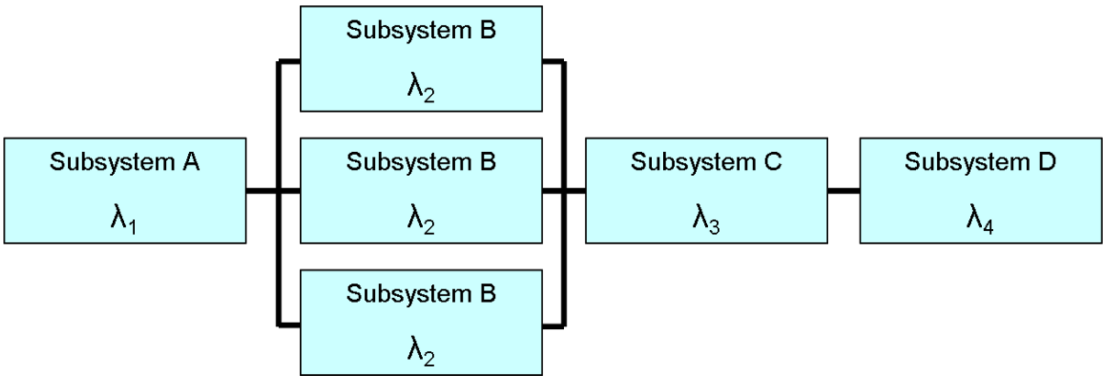


FIGURE 1.7: Example of RBD

1.3.1.5 Dynamic Reliability Block Diagrams (DRBD)

The rapid advances in computer-based technology allowed to embed more complex, dependent, and dynamic behaviors in mission critical systems to achieve more advanced functionality. As a consequence, reliability tools able to model dynamic behavior are needed. As RBDs were widely used, but did not allow reliability modeling of dynamic systems behavior, there were attempts to extend them. For this reason, a new reliability modeling tool called Dynamic Reliability Block Diagrams (DRBD), derived from RBDs is introduced to model dynamic relationships between system components [28, 30]. DRBDs are obtained by extending RBD with new constructs that allow the modeling of dynamic behavior and dependency between the components behaviors. As an example, the State Dependency (SDEP) bloc allows to model the dependency between spare components (Activation, Deactivation and Failed). A new DRBD controller component called SPARE (spare part) block is also defined to model the cold stand-by sparing [28].

1.3.2 Safety Standards

1.3.2.1 ARP4754

The ARP 4754 standard [11] entitled " Guidelines for Development of Civil Aircraft and Systems" is a standard of the Society of Automotive Engineers (SAE). It gives guidelines for the development of civil aircraft and systems with an emphasis on safety aspects. It is intended to be used in conjunction with the ARP 4761 [12] and is supported by other aviation standards like the DO-178.

1.3.2.2 ARP4761

The ARP 4761 standard [12] entitled "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment" is a standard of the SAE.

It is intended to be used conjointly with the ARP 4754 standard and is suitable only for airborne systems. This recommended practice defines a process using common modeling techniques to assess the safety of a system. It gives an overview on some safety analysis methods, then gives an example of application with a fictitious aircraft braking system. The methods covered in this standard are:

- Functional Hazard Assessment (FHA)
- Preliminary System Safety Assessment (PSSA)
- System Safety Assessment (SSA)
- Fault Tree Analysis (FTA)
- Failure Mode and Effects Analysis (FMEA)
- Failure Modes and Effects Summary (FMES)
- Common Cause Analysis (CCA), consisting of:
  - Zonal Safety Analysis (ZSA)
  - Particular Risks Analysis (PRA)
  - Common Mode Analysis (CMA)

1.3.2.3 IEC 61508 and derivatives

The IEC 61508 [31] entitled "Functional safety of electrical/electronic/programmable electronic safety-related systems". The introduction of software for control purposes by the 1980s, introduced new sources of errors in the systems that the current safety analysis techniques were unable to treat. Indeed, it is almost impossible to prove that software is correct, and even if it faithfully responds to its specification, the difficulty of getting the specification correct is also well known [32].

The IEC 61508 was developed to remedy to this situation. The standard is made up of seven parts [32]:

- Part 1 (General Requirements) defines the activities to be carried out at each stage of the overall safety life-cycle, as well as the requirements for documentation, conformance to the standard, management and safety assessment;
- Part 2 (Requirements for Electrical/ Electronic/ Programmable Electronic (E/E/PE) Safety-Related Systems) and Part 3 (Software Requirements) interpret the general requirements of Part 1 in the context of hardware and software respectively. They are specific to phase 9 (E/E/PE safety related systems realization) of the overall safety life-cycle.
- Part 4 (Definitions and Abbreviations) gives definitions of the terms used in the standard;
- Part 5 (Examples of Methods for the Determination of Safety Integrity Levels) gives risk-analysis examples and demonstrates the allocation of safety integrity levels (SILs);
- Part 6 (Guidelines on the Application of Parts 2 and 3) offers guidance on the application of Parts 2 and 3 as mentioned the title;
- Part 7 (Overview of Techniques and Measures) provides brief descriptions of techniques used in safety and software engineering, as well as references to sources of more detailed information about them.

1.4 Integration of Safety Analysis within Systems Engineering

1.4.1 Model-Based Systems Engineering (MBSE)

The first aim of systems engineering is to offer a help for the design and management of complex systems. As seen in section 1.2.1, the main characteristic of complex systems is that their behavior cannot be "thoroughly planned, understood, anticipated, and guarded against" [10]. This makes the use of models necessary during the design. Indeed, the traditional engineering processes used to rely heavily on documents. The main drawback of document-based approaches is that a major part of the effort is given to documents management rather than engineering [33]. It is laborious and time-consuming to classify, update and find

the needed documents. It's also a challenge to manage the different versions of a document and to make sure that everyone is working on the latest version. In addition, text based approaches are inefficient at finding errors and stress points in the design. They are not adequate for testing performance and comparing competing solutions [34]. Models are more easily explored and updated than documents. They are more expressive to describe systems and more likely to be understood in an unambiguous way than text-based descriptions. Simulating models also offers an easier way to perform trade-offs and comparison between alternative designs. Another major advantage of using models is that traceability between the different views and between models of different levels of abstraction is easily established. This emphasis pushed the systems engineers to prefer the use of models and led to what we call Model-Based Systems Engineering or MBSE for short.

In MBSE approaches, different models are used to model the system (and its different parts). A model is an abstraction of the real system and only represents a part of the system characteristics. This is well-expressed by the saying "All models are wrong but some are useful" by George Box. However, the abstraction allows to build lighter models that are easier to understand and less time-consuming during simulation. According to the intended use of the model, only some characteristics of the system such as timing, process behavior, or various performance measures are of interest. The appropriate type of model that focuses on the subset of the total system characteristics should then be selected [33]. This also helps managing the knowledge sharing and the management of intellectual property between different companies that work together on the same product and/or project.

Generally, a system can be viewed through different perspectives: an external perspective showing the system context and its relationships with its environment, a behavioral perspective showing the dynamic behavior of the system and the way it responds to events, or a structural perspective showing the system architecture. These perspectives were given by Sommerville in his book about software engineering [35] but they also apply for any other kind of system. In a more recent edition of his book [36], he also added the interaction perspective where the focus is placed on the interactions between the system and its environment or between the system components. Through these different perspectives, various models can be developed during the design phase - for example: a data-flow model which shows how data is transferred and processed at different stages; an architectural model which shows the composing sub-systems and their interrelationships; or a

stimulus/response model (also known as a state/transition model) which shows how the system reacts to internal and external events [22]. Using models helps to better manage complexity by separating different view points and different levels of abstraction allowing to provide the exact amount of information to the appropriate persons at the appropriate stage of the design. In addition, the use of models instead of text drifts the effort from documents management to engineering tasks. There are two general categories of models : representation models and simulation models [33]. Simulation models have the advantage that they can be executed to verify system properties.

In such Model-Based Systems Engineering (MBSE) approach, different modeling tools and languages can be used according to the different domains involved in the system, the level of detail, the system aspects to be modeled, etc. For instance, we can use IBM Doors for requirements modeling; CATIA, ProEngineer, etc. for Computer Aided Design (CAD); Modelica for complex physical systems; VHDL-AMS for mixed signal systems and integrated circuits; Matlab Simulink for dynamic systems, etc.

This thesis focuses primarily on conceptual models. In particular, we focus on early design where initial requirements are defined and progressively refined and where abstract functional models of the system are being produced - and traced to the corresponding requirements - to describe functions, their dependencies and abstract behavior. For these reasons, SysML, the systems modeling language is chosen as a supports for system models.

SysML is a semi-formal modeling language adapted to systems engineering from UML 2.0 that enhances the traditional top-down approach [34]. It is as considered the most accomplished attempt to implementing MBSE to replace the text-based approach. More detail about SysML semantics is given in Chapter 2.

1.4.2 Model-Based Safety Analysis (MBSA)

Several model-based safety analysis methods are developed to formalize the work and automate some steps of safety analysis [37]. Most of these works are based on formal methods and mainly model checking. The application of model checking

in safety analysis has been studied in many researches [22, 38–41] for safety requirements verification and/or automatic fault trees generation. Two different well known tools can be used for model checking: FSAP/NuSMV-SA and AltaRica.

FSAP/NuSMV-SA [39] is an automated analysis tool that consists of two components: FSAP (Formal Safety Analysis Platform), which provides a graphical user interface, and NuSMV-SA, which is based on the NuSMV model checker as safety analysis engine. The tool provides some predefined failure modes that can be injected by safety engineers in the initial model of the system to create a so-called extended system model. The safety requirements are expressed in temporal logic and can be subsequently verified using the NuSMV model checking verification engine. The model checker is used as a validation tool, or a powerful fault tree analysis tool.

AltaRica [38, 42, 43] is a formal specification language that was designed to specify the behavior of complex systems. The process takes system AltaRica models (composed of nodes that are characterized by their reachable states, in and out flows, events, transitions and assertions) as input and generates fault trees and model checker verification results as output.

The recent work of Sharvia [22] dealt with the integration of the Compositional Safety Analysis (CSA) and the Behavioral Safety Analysis (BSA). The first part is carried out with HiP-HOPS (Hierarchically Performed Hazards Origin and Propagation Studies), a safety analysis technique presented in [44]. In this part, system failure models such as Fault Tree Analysis and Failure Mode and Effects Analysis are constructed by establishing how the local effects of component failures combine as they propagate through the hierarchical structure of the system. CSA gives preliminary information about state-automata that represent the transition between normal and failure states of the system. Next, in the BSA, model checking can be carried out on these behavioral models in order to verify automatically the satisfaction of safety properties. So the CSA and BSA could be effectively combined to benefit from the advantages of both approaches.

Joshi et al. in their report [40] presented a Model-Based Safety Analysis approach in which the nominal (non-failure) system behavior captured in model-based development using Simulink is augmented with the fault behavior of the system. In their work, temporal logic is used to formalize informal safety requirements, and the model checker NuSMV is used to validate these requirements.

More detail about these works can be found in the related work of Chapter 5.

1.4.3 MBSE and MBSA Integration: Related Work

In this section a review of the related work about the integration of MBSE and MBSA is given.

Laleau et al. in [45] tried to combine SysML requirement diagrams and the B formal specification language. Since requirements in SysML are textual, the SysML requirement models are firstly extended to represent some concepts in the goal-oriented requirement engineering approach, such as expectation, elementary or abstract goal for requirement classes and milestone, and/or refinement for relationship between requirements. Then, derivation rules are proposed to translate the SysML goal models into B specifications. By doing so, a more precise semantics of SysML goal models is given, narrowing the gap between the requirement phase and the formal specification.

Also regarding requirements, Dubois [46] proposed to directly include system requirements in the design process but the separation with the proposed solutions as required by safety standards is achieved by isolating the following triplet: requirement models, solution models and validation and verification models. A SysML profile respecting safety standards called RPM (Requirement Profile for MeMVaTEX) has been developed. The requirement stereotype of SysML is replaced by the MeMVaTEX requirement, by adding various properties such as verifiable, verification type, derived from, satisfied by, refined by, traced to, etc. So, the traceability is assured between requirement models, between requirement and solution models, and between requirement and Verification and Validation models by using these properties. These V&V models have also been exploited in the thesis of Guillerm [16].

Another approach to integrate SysML and safety analysis is the use of the common modeling Eclipse framework. Thomas and Belmonte in [47] give an example with the independent tool called "Obeo Designer" that provides several viewpoints to model the different aspects of the system. A Safety Viewpoint is also provided to implement classical risk analyses. The interoperability of this modeling tool and the SysML model is achieved through the Eclipse Modeling Framework. To make the integration possible, the authors used the open source SysML Topcased

editor. Safety elements can reference SysML model elements since they are both expressed in the same framework. Furthermore, the Topcased GenDoc plugin can also be used to generate safety documentation from the two models. In this approach, SysML is not extended with a safety profile. In a more recent work [48], a translation from Obeo Designer's Domain Specific Language for FMEA (Failure Mode and Effects Analysis) and PHA (Preliminary Hazard Analysis) into AltaRica is added to enable formal verification. However, no real system has been studied yet in order to prove the scalability of the method.

David et al. [49, 50] worked on the generation of an FMEA report from system functional behaviors written in SysML models, and on the construction of dysfunctional models by using the AltaRica language in order to compute reliability indicators. In their methodology called MéDISIS, they start with the automatic computation of a preliminary FMEA. The structural diagrams, namely Block Definition Diagram (BDD) and Internal Block Diagram (IBD), and the behavioral diagrams such as Sequence Diagram (SD) and Activity Diagram (AD) are analyzed in detail to give an exhaustive list of failure modes for each component and each function, with their possible causes and effects. Then the final FMEA report is created with help from experts in the safety domain. To facilitate a deductive and iterative method like MéDISIS, a database of dysfunctional behaviors is kept updated in order to rapidly identify failure modes in different analysis phases. The next step of their work is the mapping between SysML models and AltaRica data flow language, so that existing tools to quantify reliability indicators such as the global failure rate, the mean time to failure, etc. can be used directly on the failure modes identified in the previous step.

Philipp Helle in [37] presents an integration process of MBSA in a SysML-based MBSE. In this work, an extension os SysML allows to include safety related information into the system model allowing the systems engineer to take some light decisions without the help of safety expert. A Java program, called Safety Analyzer, is also implemented that retrieves the system model to extract relevant information. The Safety Analyzer can then provide, as outputs, the minimal cut-set for each failure case and system alternative as well as RBD representing this cut-set.

Garro et al. [51] developed RAMSAS, a model-based method for system reliability analysis that combines SysML and the Simulink tool allowing the verification of

reliability performance of the system through simulation. A formal verification method was not used in this research for safety assessment.

Among these works, only some of them are interested in the integration of safety analyses within a SysML-based process which is the objective of this work. Among works dealing with SysML, few of them addressed the whole process. Laleau [45], Dubois [46], and Guillerm [16] mainly focused on requirements.

David [50] presented a more complete approach integrating the automated generation of safety artifacts. This work requires accessing to an external data base to import dysfunctional behavior. The generation of fault trees also requires the use of an external language: AltaRica.

In our work, the safety artifacts are extracted directly from SysML system models. In this way, we reduce the model-to-model transformations and thus the possibility of data losses during successive transformations. In addition, we update the system model with the results of safety analyses. In this work we also address behavioral safety analysis with model checking.

1.5 Conclusion

This chapter is an introductory chapter to this thesis. It quickly introduced the main concepts that will be tackled in this work. First, it highlighted how challenging is the design of nowadays multidisciplinary complex systems. Then, it showed the need of well established and adequate processes, methods and tools for the design and validation of complex systems. Systems engineering, and more precisely MBSE is considered as the new paradigm for complex systems design. The chapter also reviewed the main definitions and standards relevant to systems engineering and MBSE that will be the basis of our work.

From another viewpoint, the complexity of these systems makes the assessment of their safety also challenging. Several techniques, methods and standards exist for safety assessment and analysis and the most important ones are discussed in this chapter. These analyses however, are usually performed with independent tools and models separately from systems engineering processes. This results in errors and delays in the whole design process since it is hard to maintain safety analyses consistent with system design.

To ensure consistency and shorten time allocated to system development and safety analyses, there is a need to integrate MBSE and MBSA. Several works have already tackled this issue and are discussed in this chapter.

Chapter 2

SafeSysE: A Safety Analysis Integration In Systems Engineering Approach

2.1 Introduction

As highlighted in the previous chapter, nowadays man-made systems are getting more complex, and involving an increasing number of components from different disciplines (mechanical, hydraulic, electronic, electrical, software, etc.) that are interacting together in a synergistic way and exchanging different types of flows. The high dependence between the components makes the traditional approach of splitting the design into different domains inadequate. Instead, there is a need for the designers from different domains to collaborate together around a system model in order to make sure that the specific constraints of all the domains are accounted for. For this purpose, an MBSE approach is required in order to manage the complexity, enhance consistency and allow modeling and simulation of the whole system. As the different collaborators will work on the same model, a unified language is needed. Building a system-level model, in a unified and unambiguous language, is a key point for a successful mechatronic design. System-level model is the reference each contributor should refer to during the system design, at anytime, to have unique, up-to-date data that is shared by everyone. However, it is necessary that everybody can easily find the information needed. Then, a multi-view model with different representations of the main system is necessary. These

different views shall be linked together with traceability links and be consistent with each other. SysML, the semi-formal systems modeling language, aims at providing a unified standard for "specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities" [3]. It also allows a multi-view model and building traceability links. This relatively recent language is already widely used in both industrial and academic worlds like in [7, 37, 52–54] to cite only few works. For all these reasons, SysML is chosen as a support for system modeling in the early design stages in this work.

Number of systems are safety critical. Safety critical systems are subject to very precise requirements on their performance, as well as their dependability, safety, correctness, etc. [7]. Several safety analysis techniques are established in order to assess and manage the potential risks arising from failures or malfunctions. These analyses however, are based on independent tools and performed separately by safety engineers. The extraction of information from the system model is usually done manually. As a consequence, these analyses are error-prone and time-consuming. During safety analysis, the design usually continues to evolve and thus, safety studies are done for obsolete versions of the design model. Ensuring consistency between safety analyses and system design is thus a very hard task. To be efficient and correctly exploited by the system designers, safety analyses must be performed since the early design stages to influence the design choices without having recourse to late and costly changes. They should also be done rapidly enough to keep consistent with design, and, of course, should be error-less. To respond to these requirements, safety analyses should be integrated into the design process. To reduce error-proneness and development time, an automated generation of safety artifacts via model-to-model transformation approach is needed. The system model should then be extended to include relevant information about safety. For this purpose, new profiles are developed to extend SysML semantics.

In this chapter, we propose a methodology of **Safe**ty integration in **Sys**tems **E**ngineering approach named **SafeSysE**. This methodology, we believe, contributes to solving the problems mentioned above. Based on a SysML system model, the methodology provides a common reference system model to all the contributors that takes into account the constraints of all the domains. It is noteworthy however, that this is a high level model useful at the early design stages and it shall be complemented by domain specific models at lower detail levels. The use of a

common abstract level is useful to provide preliminary design parameters of the different components while taking into account the whole system properties. Based on this model, the methodology developed in this work automates some steps of the safety artifacts generation via automated model exploration and model-to-model transformation. This helps to reduce both development time and error-proneness.

This chapter is organized as follows. Before detailing the methodology, the chapter first presents the semantics of UML and SysML and demonstrates the adequacy of SysML as a modeling language for systems engineering projects in section 2.2. Then, the developed safety and mechatronic extended modeling profiles are presented respectively in sections 2.3 and 2.4. Next, the SysML-based systems engineering methodology that will be the support for the integrated process is presented 2.5. Our integrated approach, SafeSysE is presented in section 2.6. Finally, the chapter is concluded in section 2.7.

2.2 SysML, a UML Profile

As software programs became larger and more complex, there was a need to design them in a way that enables scalability, security, and robust execution under stressful conditions. There was also a need to structure them in a way that facilitates their maintenance and reuse. UML was developed by the Object Management Group (OMG) to help specifying, visualizing, and documenting models of software systems, including their structure and design, in a way that meets all of these requirements [55]. Since its creation, UML was rapidly adopted in software engineering and became the de facto standard software modeling language. It was recognized to contribute to the success of several projects. Even though it was initially set up for software engineering, UML was flexible for modeling multifaceted systems and was also used for systems engineering [7]. Despite its successful use for some systems engineering projects, UML is still weakly accepted by systems engineers due to its overly software-oriented semantics [7]. It also lacks some important concepts for systems modeling, namely the representation of continuous flows to model physical flows (material, energy or information) exchanged within the components. It also provides no real management of requirements during the design process. This led the OMG to create a new language, more adapted for systems engineering while keeping the benefits of UML. This language is the OMG Systems Modeling Language named SysML.

The first SysML specification was voted and adopted by the OMG in July 2006. Since that time, several tools implemented the new language and the use of SysML is growing.

Like UML, SysML shows the ability to model large systems. It also offers a common language for systems engineers to replace the wide range of modeling languages and techniques currently used to model complex systems [7] like EFFBD, Composable Objects (COB), APTE, etc.

The following subsections will deal with the semantics of UML and SysML and discuss the adequacy and limitations of each language for systems modeling.

2.2.1 The Unified Modeling Language (UML)

The Unified Modeling Language (UML) [5] is a general-purpose graphical modeling language used to specify, visualize, construct, and document the artifacts of a software system. Although initially set up for software engineering, UML can be used for several application domains. It is used to understand, design, browse, configure, maintain, and control information about systems during their design and captures decisions and understanding about such systems [56]. It is intended to support all development methods and life-cycle stages. The main objective of UML is to unify past experience about modeling techniques and to incorporate the best practices of software engineering into a standard approach [56]. UML includes semantic concepts, notation, and guidelines. "It offers rich modeling capabilities with static, dynamic, environmental, and organizational parts. It is intended to be supported by interactive visual modeling tools that have code generators and report writers. The UML specification does not define a standard process but is intended to be useful with an iterative development process. It is intended to support most existing object-oriented development processes".[56].

UML is an object-oriented language, i.e. it is based on the concepts of object, class, inheritance and extensibility [34]. It supports multiple views with varying levels of abstraction. The ability to support multiple levels of abstraction is crucial for systems and software engineering since it enables top-down development with progressive refinement from the need expression to the solution definition.

An object is a basic building block able to receive and send messages as well as processing data. It can be either a component or an actor. A class, in object-oriented terminology, stands for a grouping of related variables or functions. It is one of the key concepts for supporting multiple levels of abstraction. Different instances of the same class can be created, each instance having specific values of class parameters. This process known as inheritance (and also generalization) promotes reuse.

The vocabulary of the UML encompasses three kinds of building blocks [57]: things, relationships and diagrams.

2.2.1.1 Things in UML

"Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things" [57].

There are four kinds of Things in UML :

- Structural things are the mostly static parts of a model, representing elements that are either conceptual or physical. The basic structural things are class, interface, collaboration, use case, active classes, components, node;
- Behavioral things are the dynamic parts of UML models. The basic behavioral things are interactions and state machines;
- Grouping things are the organizational parts of UML models. Packages are the basic grouping things;
- Annotational things are the explanatory parts of UML models.

2.2.1.2 Relationships in UML

Relationships are the basic relational building blocks of UML. They are used to specify relationships between things in models.

There are four relationships : Dependency, Association, Generalization, Realization.

- Dependency: a dependency is a semantic relationship between two things which means that a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing). Graphically, a dependency is represented by a dashed line that can be directed and may also include a label.
- Association: an association is a structural relationship that describes a set of links i.e., connections among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts. Graphically, an association is rendered as a solid line, possibly directed. It occasionally includes adornments such as label, multiplicity, and role names.
- Generalization: a generalization expresses a specialization/generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent). Graphically, a generalization relationship is rendered as a solid line with a hollow arrow head pointing to the parent.
- Realization: a realization is a semantic relationship between classifiers. Graphically, a realization relationship is rendered as a cross between a generalization and a dependency relationship.

There are also variations on these relationships, such as refinement, trace, include, and extend.

2.2.1.3 Diagrams in UML

A diagram is the graphical presentation of a set of the model elements, rendered as a connected graph of vertices (things) and arcs (relationships). Diagrams are drawn to visualize a system from different perspectives. A diagram is a representation of a particular view of the elements that make up a system and their relationships. The same element may appear in all diagrams, only in a few diagrams (the most common case), or in no diagrams at all. In theory, a diagram may contain any combination of things and relationships. In practice, however, a small number of common combinations arise, which are consistent with the five most useful views that comprise the architecture of a software intensive system [57]. UML specification defines two major kinds of diagrams : structure diagrams and behavior diagrams. UML diagrams are represented in Figure 2.1.

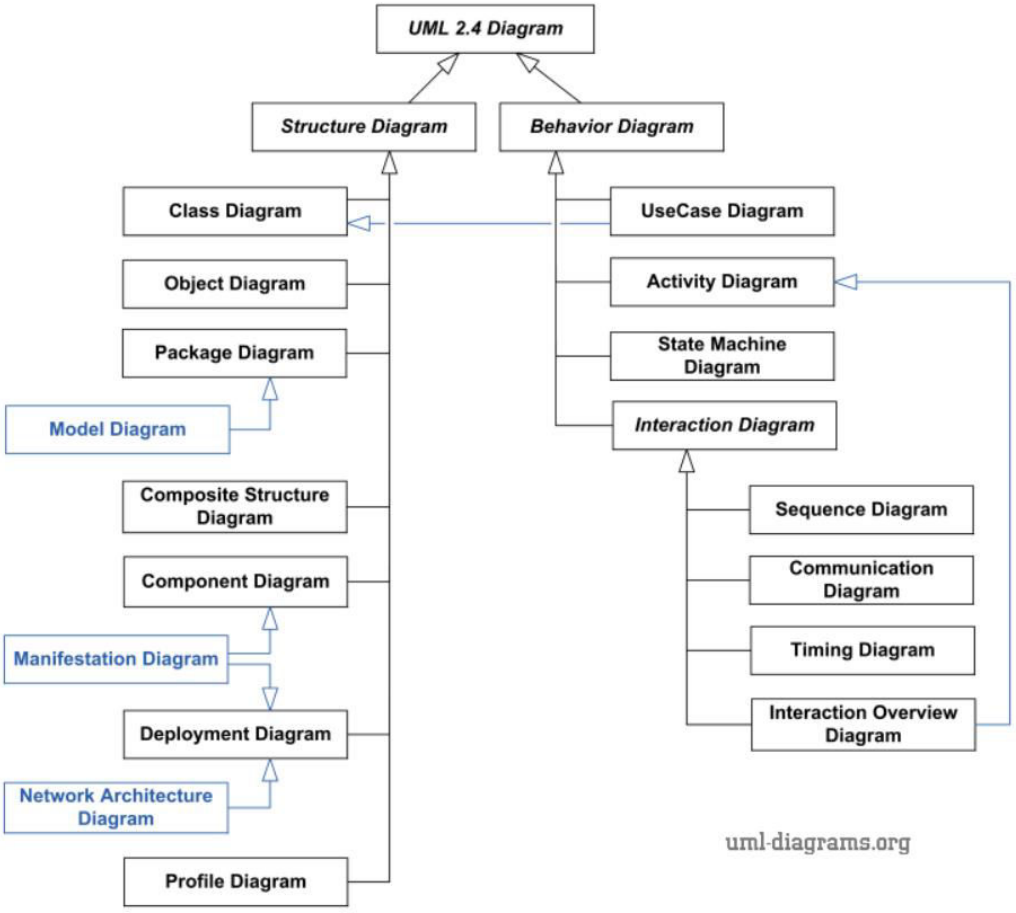


FIGURE 2.1: UML Diagrams [2]

2.2.2 UML Extension Mechanisms

As seen in the previous section, UML has core concepts that cover common case applications but may be not very accurate for some specific domains. There are two reasons for this. The first one is that trying to be exhaustive will lead to a very complicated language and it will be very hard to master all the concepts. The second reason is that there is such a variety of domains that it is practically impossible to build a comprehensive language covering all the existing domains. To cope with this limitation, extension mechanisms are used to add new concepts and notations. Extensions are intended for particular application domains or programming environments. They can be obtained by adding new constraints, tagged values, and stereotypes to build new profiles. As a generic UML extension mechanism, a profile is a collection of stereotypes, stereotype attributes and constraints applied to specific UML modeling elements (classes, activities, attributes, etc.) in order to customize UML for specific domains and platforms. For this purpose,

UML vocabulary is extended by adding stereotypes to create new domain-specific modeling elements derived from generic UML ones. Several profiles are derived by the OMG from UML like SysML (Systems Modeling Language) [3], MARTE (Modeling and Analysis of Real-time and Embedded systems) [58], and UPDM (Unified profile for DoDAF/MODAF) adapted for systems modeling, real time modeling and supporting DoDAF/MODAF architecture frameworks respectively. Another profile adapted for mechatronic systems modeling was also created and labeled Mechatronic UML. These profiles are created to respond to specific modeling needs not achievable in UML.

MARTE is the UML 2.0 OMG standard for Real-Time Embedded systems modeling and Analysis (RTEA). It replaces the former UML SPTP profile for Schedulability, Performance and Time Specification. It deals with both software and hardware aspects and provides non-functional property modeling such as performance and scheduling. It also adds time features, defines concepts for software and hardware platform modeling, and provides quantitative analysis [59].

MARTE was efficiently used in several industrial projects from different domains. Three examples of industrial use of MARTE in Real Time Embedded Systems (RTES) design are described in [60]. The first one deals with architectural modeling and configuration, applied to integrated control systems (ICSs) which are heterogeneous systems-of-systems. In these systems, software and hardware components are integrated to control and monitor physical devices and processes, such as process plants or oil and gas production platforms. The use of MARTE was mainly for capturing software and hardware components interactions, enhancing consistency between software and hardware modeling and, finally, enabling automated configuration and configuration reuse.

The second experience deals with model-based robustness testing, using also stereotypes from Robustprofile from Simula Research Laboratory [61]. The project aimed at supporting automated, model-based robustness testing of a video conferencing system from CISCO Systems Inc.

The last experience deals with Testing RTES using MARTE environment models on a large and complex seismic acquisition system with tens of thousands of sensors and actuators in its marine environment. Models were used to generate an environment simulator, test cases, and obtain test oracles.

MARTE is also a UML profile for AADL (Architecture Analysis and Design Language), a Domain-Specific Language (DSL) that deals with the hardware platform and the physical environment of intensive embedded software systems. It can be used to model application tasks and communication architectures, thus allowing modeling and analysis of coupled software and hardware RTES aspects.

Mechatronic UML is a specific profile for mechatronic systems. It is derived from the safety-critical software development domain with the main objective of bringing model-based design, domain testing and simulation, and formal analysis to the mechatronics area [62]. The aim behind this is to guarantee highly safety-critical system properties. The profile restricts the usage of UML to certain types of diagrams and extends these diagrams to be able to model hybrid and self-adaptive systems. The use of formal semantics to model the components allows formal analysis. This profile aims at being more efficient than approaches such as SysML that does not adequately support modeling of time, and MARTE that does not provide the needed architectural abstraction for hardware [62].

The fourth profile is UPDM (Unified Profile for DoDAF/MODAF) [63]. DoDAF (Department of Defense Architecture Framework, US), MODAF (Ministry of Defense Architecture Framework, UK) and NAF (NATO Architecture Framework) use this profile to define architecture frameworks that are domain specific and define practices for creating, interpreting, analyzing and using architecture descriptions, as described in ISO/IEC/IEEE 42010 [64].

2.2.2.1 Limitation of UML for Systems Modeling

UML was firstly used for software development and consequently it has overly software-oriented semantics. This results in a week acceptance by systems engineers unfamiliar (and even having some fair/hostility) with object oriented terms and concepts like class, inheritance, etc. However, this is not the only obstacle for using UML in systems engineering. Since, in software engineering, behavior is mainly discrete, UML lacks of constructs for continuous behavior. It does not explicitly support modeling continuous functions and flows and does not offer any clear expression of physical flows of energy, material and information passing between the system components. Consequently, it is not fully appropriate for complex multi-disciplinary systems design. It's more appropriate for software intensive systems. This was a major obstacle for persons who tried to use UML for systems

modeling as reported in [7]. Managing some system aspects like continuous flows needed proper stereotypes to be developed which is opposite to one of the most important reasons for UML: unification and standardization of modeling. There will be as many different ways to model continuous flows and functions as the number of persons trying to build such models. This also prevents the reuse of such models with commercial tools.

These lacks led the OMG (Object Management Group) to derive a new profile specified for systems engineering, the SysML language. It was constructed as a subset of UML, completed by additional modeling possibilities specific for systems engineering.

Some of these diagrams are integrated in SysML without any change (package, state machine, use case, and sequence diagrams), some others are modified (class diagram, activity diagram and composite structure diagram) and the remaining diagrams were suppressed because considered irrelevant for systems modeling.

2.2.3 SysML, the Systems Modeling Language

SysML is a profile of UML dedicated to systems engineering [3, 65]. It has been built to bring together the successful tools and methods used in engineering into a unique and standardized language. It is based on the experience and lessons gathered from the use of UML 2.0 and includes successful engineering graphical representations such as Enhanced Functional Flow Block Diagram (EFFBD) [66].

Like UML, SysML is a semi-formal modeling language. Semi-formal languages are usually based on graphical modeling with a more or less accurate semantics, i.e. they can have elements of ambiguity [67]. At the opposite to formal languages, semi-formal languages are polysemic which means that an element can have more than one meaning. This makes them subject to fierce criticizes from mathematicians and formal modeling lovers. This property however, makes semi-formal languages more flexible and easier to use, especially at a high level of abstraction. Indeed, “starting the study of a complex system using formal models appears to be hardly achievable in practice” [68]. The use of semi-formal language, mainly if they are based on graphical formalism, makes the models easier to explain and understandable by all the persons involved may they be domain engineers or commercial and management decision makers. This justifies the use of languages such

as SysML to cover the first phases of specification using a graphical modeling that promotes the knowledge sharing between the involved engineers.

SysML is designed to provide simple but powerful constructs to model a wide range of systems engineering problems [3]. For a better acceptance by systems engineers, object-oriented vocabulary was hidden in SysML and replaced by general vocabulary. Still in the same perspective, some concepts entirely dedicated to software engineering were removed. Consequently, only some UML diagrams are part of SysML. On the other hand, and to cope with the limitations of UML, new concepts that are crucial for systems engineering were added. SysML is then constructed as a subset of UML 2.0 complemented by additional concepts offering more modeling possibilities [7]. As a result, SysML is particularly effective in specifying requirements, structure, behavior, allocations, and constraints on system properties to support engineering analysis [3]. It has inherited UML abilities, notably the advantages of object-oriented methods for complex system design and of its standardized aspect. This new profile, SysML, helps performing several design tasks such as reliability study [7], complex and embedded systems modeling [69] etc. It is also more and more adopted by industrialists such as Valeo, a supplier of automobile systems [54, 70] and Airbus, a designer and integrator of aeronautic systems [71] to cite only a few examples of SysML deployment.

In this section, SysML semantics are detailed. SysML contains nine diagrams instead of thirteen diagrams in UML. The nine diagrams are partitioned into 3 categories : Requirement Diagram, Behavior Diagrams and Structure Diagrams. Figure 2.2 shows the different diagrams in SysML.

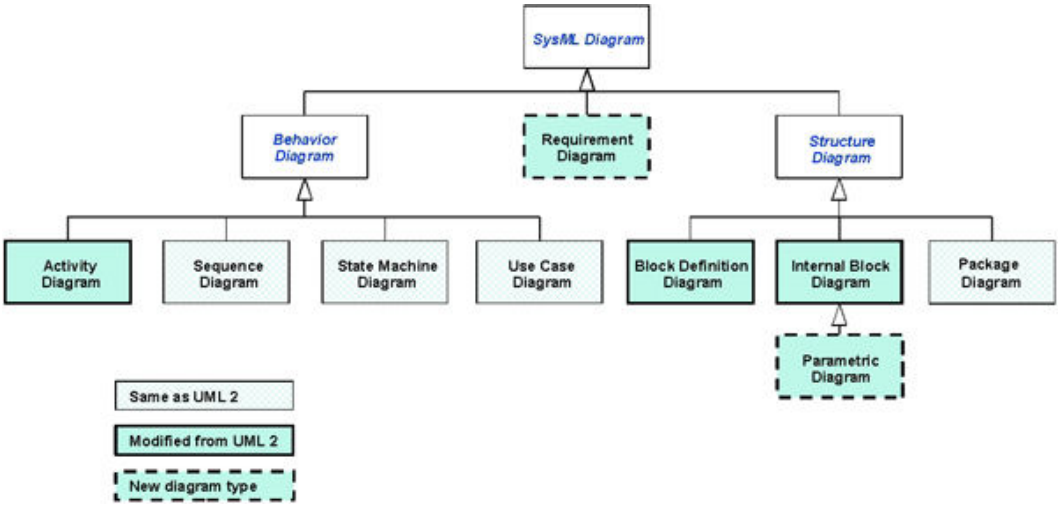


FIGURE 2.2: SysML Diagrams [3]

The next sections deal with the three categories of diagrams respectively.

2.2.3.1 Requirements Modeling with SysML

Requirement managing includes requirements capture, requirements analysis (defining critical requirements, making trade-offs between conflicting requirements, etc.) and finally allocation (allocating requirements to use cases, components, test cases, etc.).

In UML, requirements are mainly modeled with use cases. However, even if use cases are adequate to model functional requirements, they are not able to model non-functional requirements [72]. In systems engineering, there are several non-functional requirements such as reliability, performance, regulation and so on. SysML integrated new constructs to model requirements. A requirement stereotype, which is a class stereotype including an open list of properties (that are attributes of the stereotype), is created. The elementary properties of a requirement are the text definition and the id which are both strings. The text definition describes the requirement that has to be met by the system in natural language. In some cases, it may also include a reference to external source like a regulatory standard. Other properties like source, priority, verification, etc. can be added to the requirement via the extension mechanism. In the same way, requirements categories such as functional requirements, interface requirements or safety requirements, etc. can be introduced [68, 72].

Several relationships are also defined to trace the requirements together or with other model elements 2.3.

- **Namespace Containment**

  The **namespace containment** (it can also be referred to as composition) describes the fact that a requirement is contained in another requirement [72]. It helps structuring the requirements and means that the composed (or master) requirement is realized if and only if all the component requirements (or sub-requirements) are realized.

- **Derive Requirement Relationship**

  A **derive requirement** relationship describes that a requirement is derived from another requirement [72]. Requirements derivation usually results from

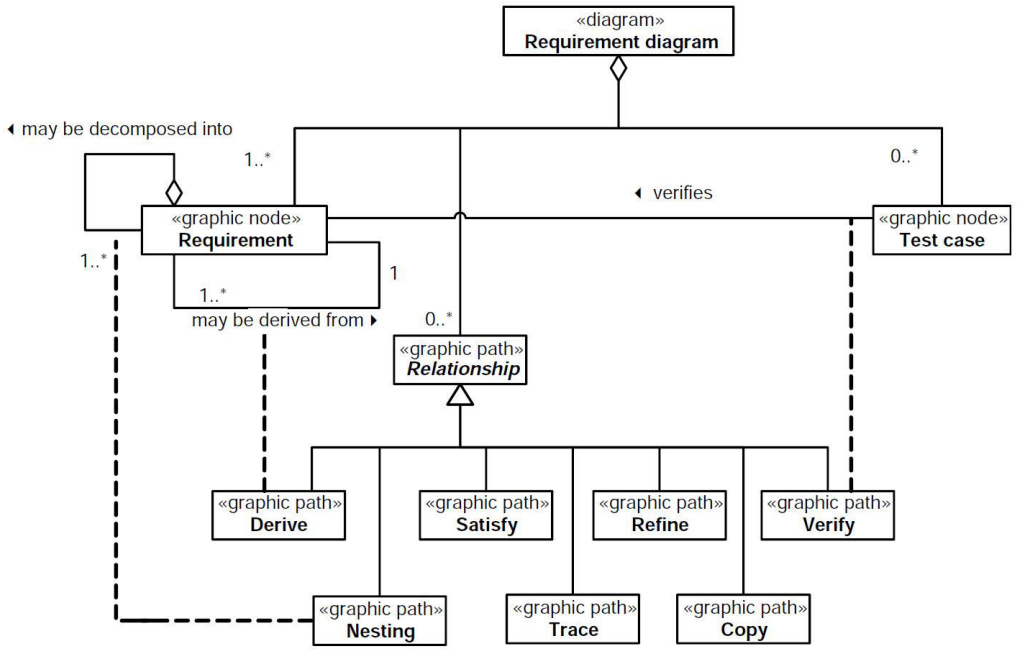


FIGURE 2.3: Partial Meta-Model of the Requirements Diagram [4]

an analysis of the source requirement. For example, a business requirement can result in one or more technical requirements. The derive relationship is only allowed between requirements.

- **Satisfy Relationship**

  The **satisfy** relationship describes that a design element satisfies (or realizes) a requirement. This helps evaluating the impact of the design changes on the requirements and vice-versa. A requirement can be satisfied by several design elements. In this case, the satisfy relationship does not specify if the design element fully or partially satisfies the requirement [72]. This information can be added by a comment or note in the model.

- **Copy Relationship**

  The **copy** relationship connects a requirement with another requirement and describes the fact that a requirement is a copy of another. This enables the reuse of the same requirement in different contexts. In this case, the copy is maintained consistent with the original requirement, i.e. the id and the name may be different but the text description is write-protected in the copy and automatically updated to be identical to the original requirement if it is modified [72].

- **Verify Relationship**

The **verify** relationship connects a requirement with the test case that is used to verify it [72]. Together with the test case stereotype, the verify relationship allows specifying how the requirements have to be verified early in the design. This avoids misunderstanding or bad interpretation of the requirements by the persons who, later, will verify the requirements. As for the satisfy relationship, the verify does not specify the completeness of the test case with regard to the requirement and a comment can be used to add this information [72].

- **Refine Relationship**

  The **refine** relationship specifies that a model element describes the properties of a requirement in more detail [72]. Sometimes, the text of the requirement is not sufficient for describing the requirement in detail. In this case, the requirement can be refined by other model elements (eg. a use case to refine a requirement).

- **Trace Relationship**

  The **trace** relationship describes a general relation of traceability between a requirement and another model element [72]. The aim of this relationship is to specify a traceability between the two elements without specifying the reason of it.

Requirements and their relationships can be presented in a graphical form within Requirement diagrams. This is meaningful when the purpose is to focus on a particular set of requirements and their relationships. When viewing a large number of requirements on a requirement diagram however, it is very hard to depict and relate requirements. It is not practical to visualize the several hundreds or thousands of requirements that a project can have in graphical form. SysML also offers a table notation to represent the requirements in a more visible way. The requirement tables are adaptable and offer the possibility to focus on particular aspect by choosing the requirement properties to visualize (id, text, relationships like satisfaction, derivation etc) as well as the analyzed packages of the model.

2.2.3.2 Structure modeling with SysML

SysML provides four structural diagrams: the Block Definition Diagram (BDD), the Internal Block Diagram (IBD), the Parametric diagram (Par) and the Package diagram (Pkg).

Block Definition Diagram (BDD)

A Block is the basic structural element in SysML, and corresponds to a class in UML.

Blocks describe a system as a collection of parts, each of which plays a specific role in a particular context. They describe parts of the structure of a system [72]. A block can be either a logical or a physical unit. Blocks can also reference other blocks they are bound with (association). The attributes of a block provide information about the block itself. Quantifiable properties of a block are described by the block value properties. This is very useful to define the parameters (performance, reliability, etc) of the system and the components.

As a result, a block has three main properties that are respectively *part property*, *reference property* and *value property*.

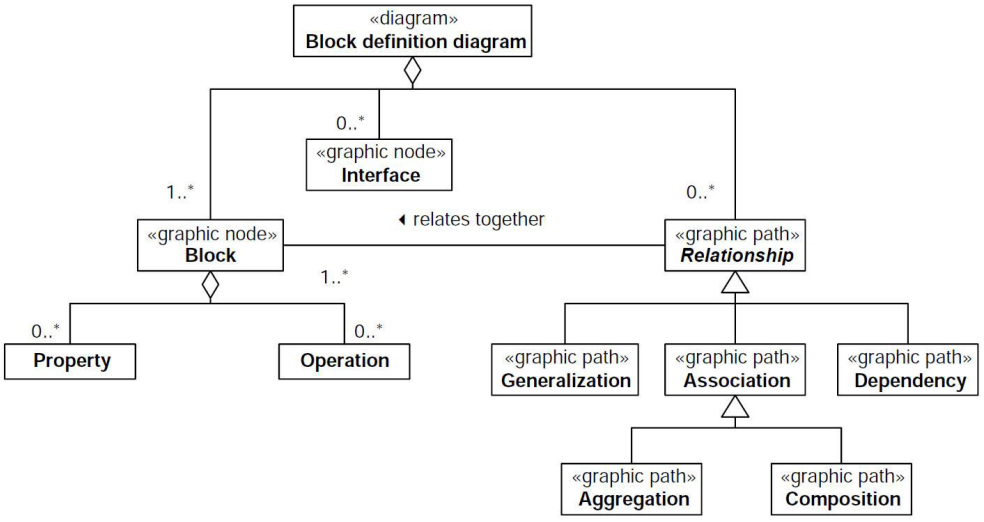


FIGURE 2.4: Meta-model of the Block Definition Diagram (BDD) [4]

In addition to the purely static structure, a block can also describe the behavior it executes with operations [72]. Figure 2.4 represents the meta-model of the Block Definition Diagram.

Internal Block Diagram (IBD)

An internal block diagram is a modification of the UML composite structure diagram. It describes the internal structure of a block by representing the interfaces and connections between parts of a block. It shows how the block properties (parts and references) are interconnected. In addition to the internal exchanges among components, it also represents the exchanges of the block with its environment (contributing systems and/or users). Ports allow to model the interfaces. There are two kinds of ports : standard ports and flow ports. An optional item flow can also be added on the connector to specify the flow.

- **Standard ports**

  A standard port specifies the services that either are required or provided by the part.

- **Flow ports**

  Flow ports are the interaction point of a block through which objects can flow into and out of the block. Usually, objects flow in a particular direction. For this reason flow ports are directed. Input flow ports, denoted with the keyword “in”, allow objects to flow into the block. Output flow ports, denoted with the keyword “out”, allow objects to flow out of the block. There are also bidirectional flow ports, denoted with the keyword “inout” and they allow objects to flow in both directions. The communication path between two blocks is represented by a connector connecting the flow ports of these blocks.

- **Item flows**

  An item flow is an optional element that can be added on the connector to describe the specific objects that are transported. Figure 2.5 shows the different elements that can be found in an Internal Block Diagram.

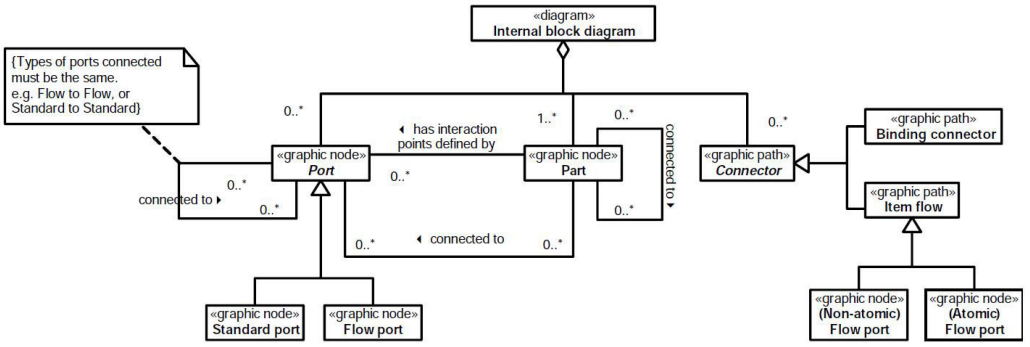


FIGURE 2.5: Meta-model of the Internal Block Diagram (IBD) [4]

Parametric Diagram

The parametric diagram is a new diagram in SysML that is not in UML. The parametric diagram aims at identifying the main system parameters (block value properties) as well as their relationships seen as constraints. Integrating parametric relationships in the design allows taking into account performance or reliability aspects. It is also useful for instance to compare alternative solutions with regard to particular criteria like response time, global mass, reliability, etc. [72]. The relationships can be physical laws like the Ohm law relating the voltage to the current and resistance or Newton's law relating the force to the mass and acceleration.

- **Constraint blocks**

  These relationships are captured in **constraint blocks** that describe constraints on system structures and the parameters required. A constraint block contains a set of constraints (the mathematical relationships) and the parameters of these constraints. A constraint block is built context-independent and can be used in different contexts by binding the constraint parameters to the model parameters.

Even if it allows the use of parametric diagram to capture the relationships, SysML is not intended to solve these equations and does not offer any solver for this purpose. On the other hand, it recommends the implementation of exchange formats allowing the communication to external solvers for the resolution of the equations. In this case, the equations shall be written using the syntax of the solver. Figure 2.6 represents the different elements that can be found in a parametric diagram.

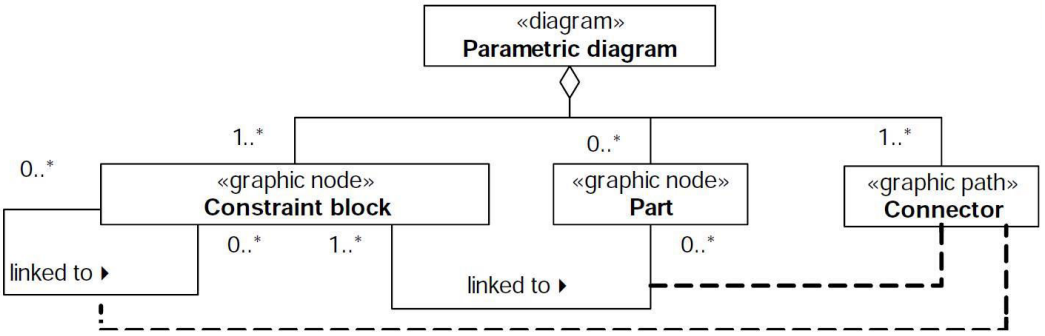


FIGURE 2.6: Meta-model of the Parametric Diagram [4]

Package Diagram

The last diagram used in the structure modeling is the package diagram. This diagram represents the organization of a model in terms of packages that contain modeling elements (Figure 2.7).

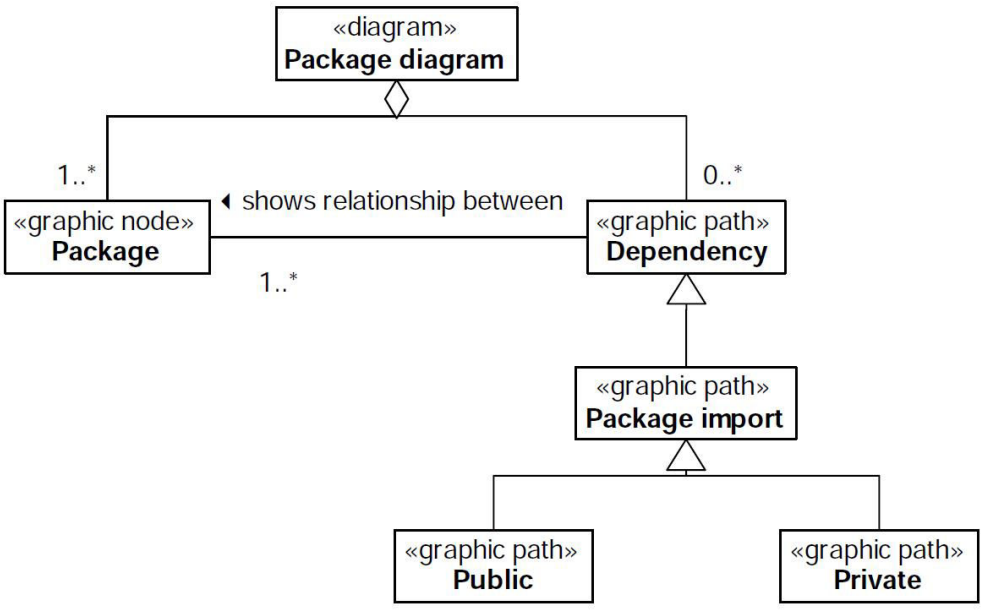


FIGURE 2.7: Meta-model of the Package Diagram [4]

2.2.3.3 Behavior Modeling with SysML

In addition to the static aspects, SysML also offers the possibility to model dynamic aspects with behavior diagrams. SysML provides several diagrams to model different behavior kinds and thus offers a comprehensive description of behavior

that helps to reach a complete specification of system [73]. It covers flow-based behavior, where behavior is detailed in terms of the flow of inputs, outputs and control with **activity diagrams**. **State machine diagrams** support event-based behavior expressed in terms of response of blocks to internal and external events. Functionality of system, in terms of the services it provides to potential users, is represented by **use cases**. Message-based behavior used to model service-oriented concepts is also supported in SysML by **sequence diagrams**.

Activity Diagram (ACT)

Activity modeling emphasizes the inputs, outputs, sequences, and conditions for coordinating other behaviors [65]. Activity diagrams include semantics for precisely specifying the system behavior in terms of the flow of control, inputs and outputs. An activity diagram represents a controlled sequence of actions that transforms inputs into outputs [73]. Activity diagrams in SysML, are extended with flow-based behavior modeling and can be used in a similar way to the Functional Flow Block Diagram (FFBD) that has been widely used for modeling systems. They describe the functional breakdown in a hierarchical way with different detail levels and thus, can provide the functional architecture of the system. Activity diagrams are made up of three basic elements : activity nodes, activity edges and regions [4] (Figure 2.8). There are three main types of activity nodes: *activity invocation*, *object* and *control node*. Control nodes offer rich semantics to model the sequencing of behavior in activity diagram. They are presented in terms of sets of two complementary nodes that are:

- The *Initial node* and the *Final node*. The *initial node* shows the starting point of an activity diagram. There are two types of final nodes: the *Activity final node* that shows where the activity diagram ends and the *Flow final node* that shows where a particular flow ends.
- The *Join node* and the *Fork node*. The *Fork node* allows a flow to be split into several parallel routes while the *Join node* allows the flows to join again in a later point in the diagram. By default, all the flows are necessary in the join flow but logical relationships can also be specified.

- *Decision node* and *Merge node*. *Decision node* allows a flow to branch off into a particular route according to a condition. The *Merge node* allows flows to merge back into a single flow.

Activity edges are the paths that connect activity nodes to each other. There are two main activity edge types: control flow and object flow. The first one specifies when, and in which order, the actions within an activity will execute. The second one specifies the objects (physical items or information) that flow between activity nodes.

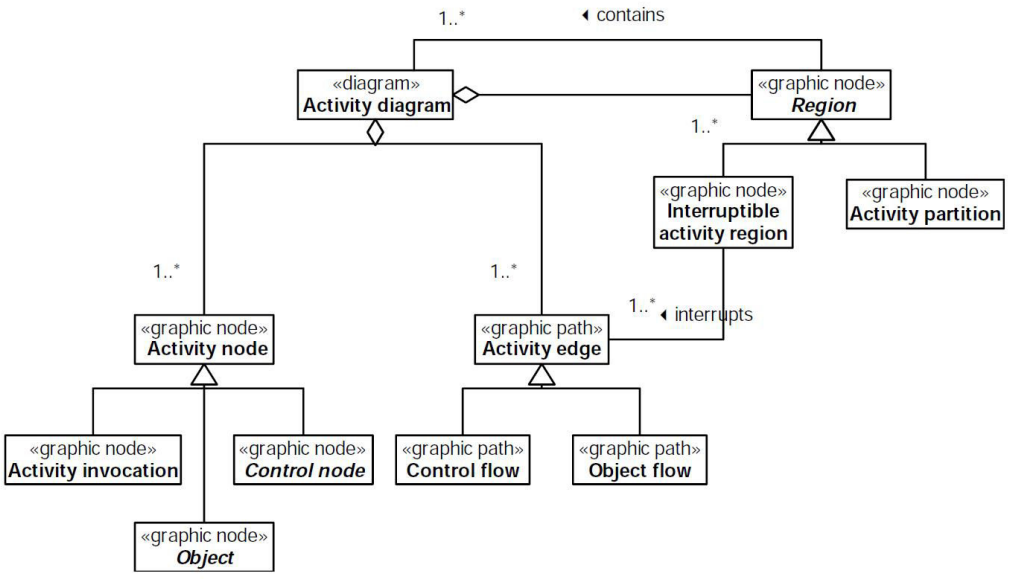


FIGURE 2.8: Meta-model of the Activity Diagram [4]

State Machine Diagram (STM)

The state machine package defines a set of concepts that can be used for modeling discrete behavior through finite state transition systems. The state machine represents behavior of a system in terms of its states and transitions. The activities that are invoked during the transition, entry, and exit of the states are specified along with the associated event and guard conditions. Activities that are invoked while in the state are specified as "do Activities", and can be either continuous or discrete. States can be either atomic or composite. A composite state has nested states that can be sequential or concurrent [65].

State machine diagrams are used to identify different states (or operating modes) of a system. Establishing such diagrams helps to define the transition from one

state to another. Each mode (state) is then treated separately and a comprehensive description of the behavior of system is established. Figure 2.9 gives a meta-model of state machine diagram.

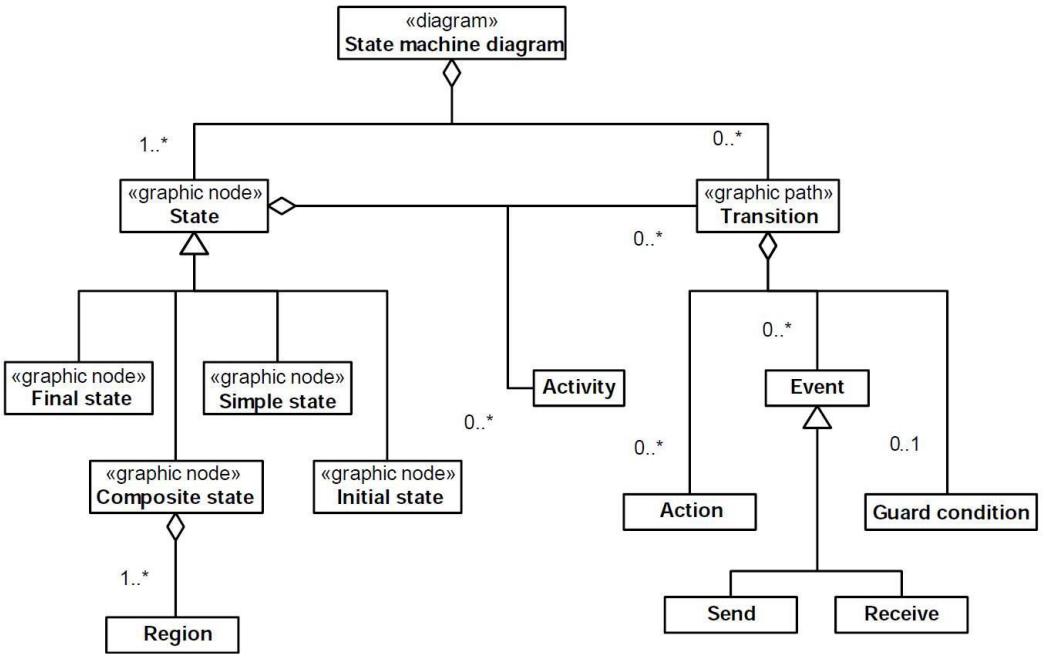


FIGURE 2.9: Meta-model of the State Machine Diagram [4]

Use Case Diagram (UC)

The use case diagram is directly integrated from UML. Use case diagrams represent system functionality in terms of how a system (or a subject) is used by external entities (such as actors) to achieve a set of goals. In other words, it represents the set of services provided by the subject to the selected actors [65].

A use case can also be viewed as functionality and/or capabilities that are accomplished through the interaction between the subject and its actors. Use case diagrams include the use case and actors and the associated communications between them. Actors are external to the system and they may represent users, systems, and or other environmental entities. They may interact either directly or indirectly with the system [65].

The association between the actors and the use case represent the communications that occur between the actors and the subject to accomplish the functionality associated with the use case. The subject of the use case can be represented

via a system boundary. The use cases that are enclosed in the system boundary represent functionality that is realized by behaviors such as activity diagrams, sequence diagrams, and state machine diagrams [65].

The use case relationships are "communication," "include," "extend," and "generalization." Actors are connected to use cases via communication paths, that are represented by an association relationship. The "include" relationship provides a mechanism for factoring out common functionality that is shared among multiple use cases, and is required for the goals of the actor of the base use case to be met. The "extend" relationship provides optional functionality (optional in the sense of not being required to meet the goals), which extends the base use case at defined extension points under specified conditions. The "generalization" relationship provides a mechanism to specify variants of the base use case. Figure 2.10 gives a meta-model of the use case diagram.

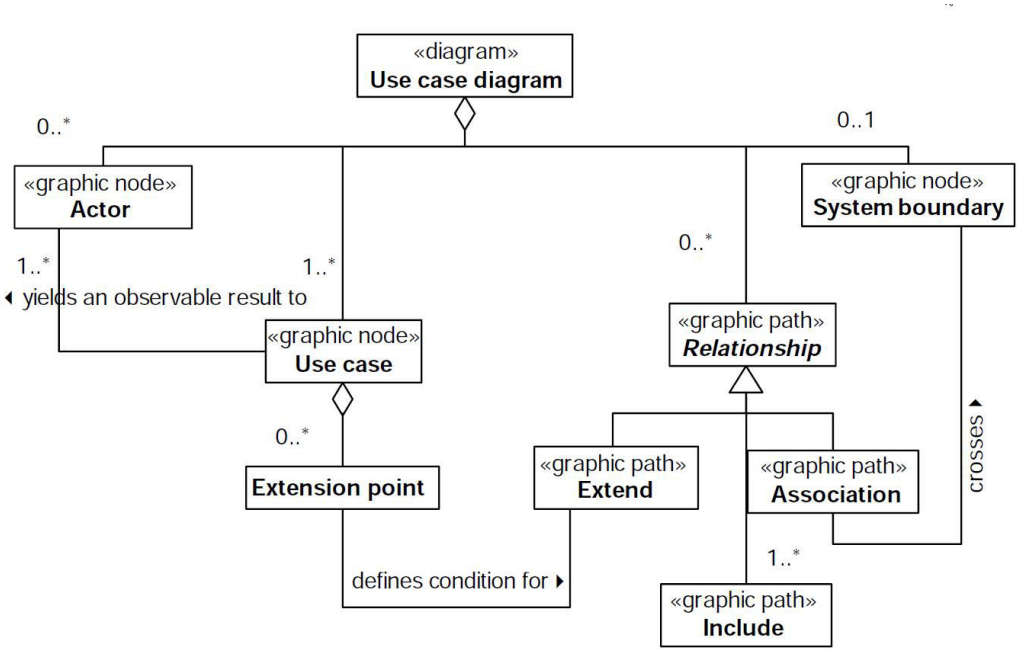


FIGURE 2.10: Meta-model of the Use Case Diagram [4]

Sequence Diagram (SD)

Sequence diagrams are typically used to model scenarios [4]. In a sequence diagram, only the pertinent aspects of the particular situation described by the scenario are highlighted. Each of these aspects is represented as an element known as a 'Life line'. A 'Life line' in SysML represents a participant in an interaction and

refers to an element of the model, such as a block, a part or an actor. Sequence diagrams model interactions between life lines, showing the messages passed between them with an emphasis on logical time or the sequence of messages (hence the name). The interactions (or sequences) occur in the order of their appearance in the diagram, i.e. from the top to the bottom. Sequence diagrams can be used at different levels of systems hierarchy. At the system level they represent interactions between system and its environment. At a lower level of detail, they represent interactions between different components of system. Sequence diagrams have a very rich syntax [4]. They describe the flow of control between actors and systems (blocks) or between parts of a system [65]. Combined fragments can be added to offer the possibility to model some particular aspects like loops or parallel sequences. A combined fragment is defined by an interaction operator (alt for alternatives, par for parallel, loop for a loop, etc.) and corresponding interaction operands. For more detail please refer to the SysML specification [65].

The meta-model of a sequence diagram is given in Figure 2.11. It shows the diagram elements and their relationships.

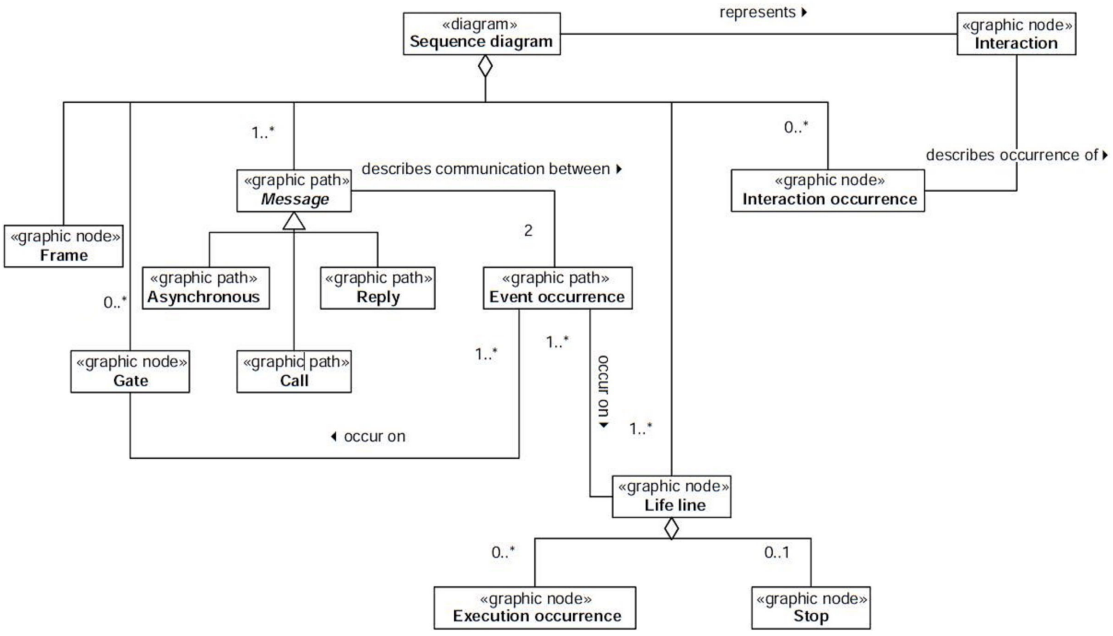


FIGURE 2.11: Meta-model of the Sequence Diagram [4]

2.2.3.4 Transverse Constructs in SysML

To ensure the consistency between the previous aspects of a system, SysML contains a set of traceability links. Some of these relationships can be traced between

requirements as they can link requirements to other model elements as shown in section 2.2.3.1. Other relationships can link model elements to each other. Allocation can be defined between the system functions (in the behavior part of the model) and the components (in the structure part of the model) that will achieve these functions. These relationships summarized in Figure 2.12 enhance the consistency of the model and facilitate the evaluation of the impact of design changes.

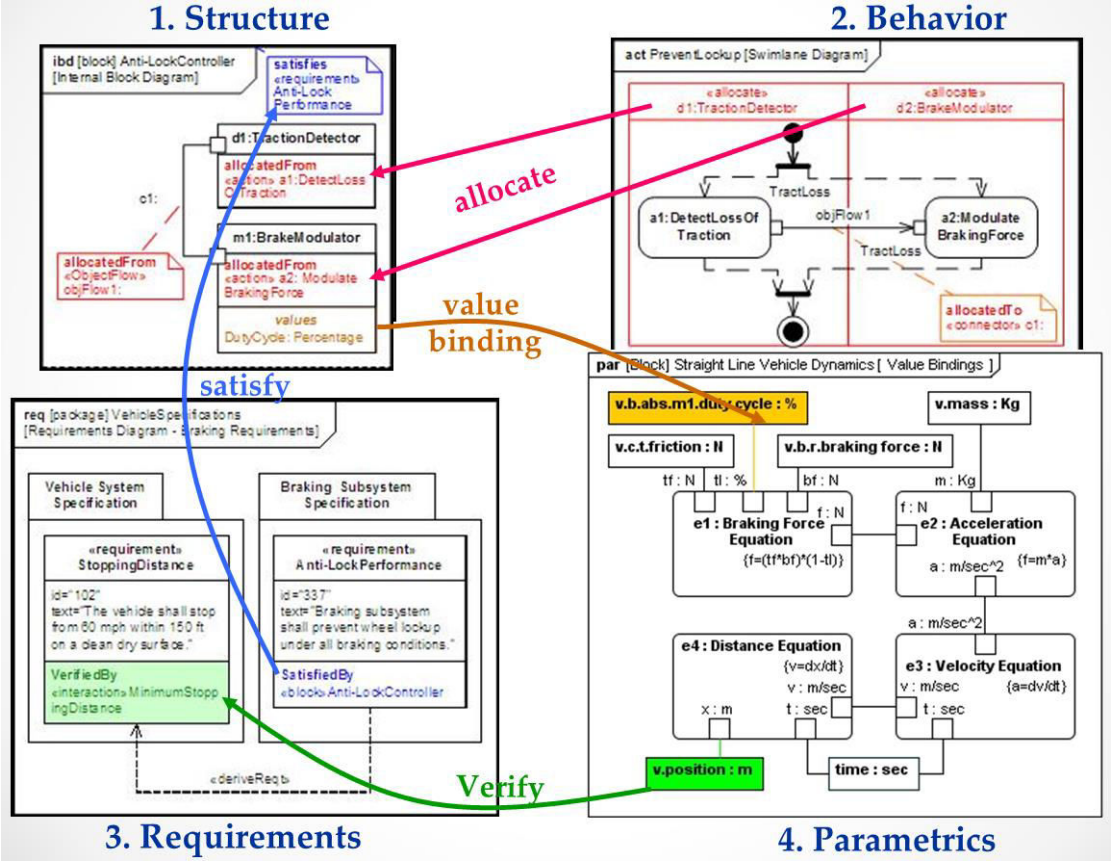


FIGURE 2.12: Consistency in SysML [5]

We have stated that SysML offers a wide range of constructs that respond to the main needs of modeling in systems engineering. The additions made to SysML compared to UML brought key concepts for systems engineering. Requirements added a better way to model needs and specifications in a textual form, which is the usual way for systems engineers to express the requirements. It also allowed to model non-functional requirements like performance and safety requirements. The introduction of continuous functions filled a large gap in UML since it enables modeling the physical flows. Value types allow adding a dimension and a unit to data types in UML which is very useful in checking the consistency of dimensions in the models.

If these extensions are sufficient to model a large variety of systems, some domains require specific concepts that are not integrated in SysML. For this reason, SysML, as a profile of UML, kept the same extension capability and mechanisms than UML. This offers the possibility to SysML users to enlarge their use of the language by extending its semantics with domain specific concepts. We have considered this opportunity to enrich SysML semantics with some constructs relative to safety and to machatronic systems modeling. In the following, the Safety Profile and the Mechatronics Extended Modeling Profile developed in this thesis are presented.

2.3 SysML Safety Profile

During the design phase, the designers (systems or domain engineers) may have relevant information concerning safety, especially if they are integrating new concepts or innovating technology. In this case, they are recommended to transmit these data to safety experts. In the same way, it is important for a safety expert to feed back safety analysis results to systems engineers to take into account these results in the system design. In order to integrate safety information directly into SysML models, we have used the extension mechanism of UML to create a so called Safety Profile. A profile allows adaptation or customization of UML meta-models to a specific platform, domain or method through stereotype and tag definition concepts [55]. The stereotype is the primary extension construct that extends an existing meta-class. A stereotype may have properties that are referred to as tag definitions. The safety profile is intended to allow capturing safety aspects in the system model. In our case, the Safety Profile is built from stereotypes and tag definitions that represent artifacts useful for the safety analysis techniques we selected for our integrated process, i.e. FMEA, FTA and behavioral safety analysis. The constructs added to enable FMEA building are failure modes, causal factors, system effects, probability, severity, etc. The relationships between this information are modeled by a class diagram given in Figure 2.13. For instance, this diagram specifies that:

- One or more functions are allocated to one component;
- Each function or component can have one or more Failure Modes;
- A failure mode is caused by one or more causal factors;

- A failure mode generates effects both at local and system level (Immediate Effect and System Effect respectively);
- etc.

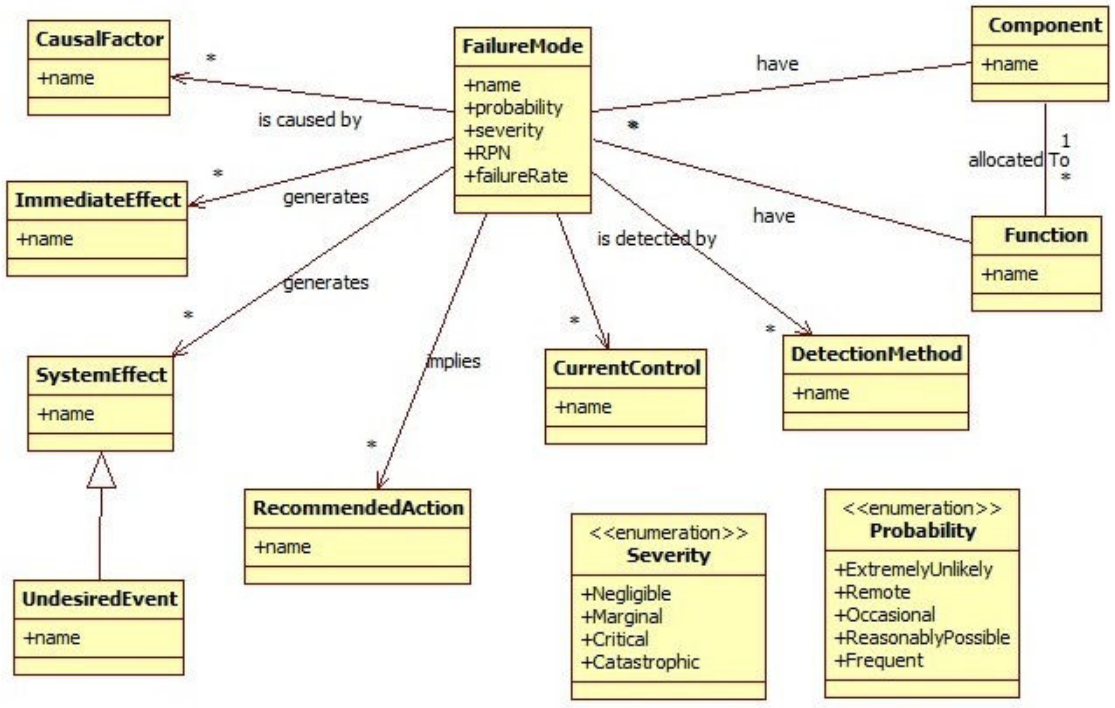


FIGURE 2.13: Class Diagram for FMEA Artifacts

Since in our methodology, a system function is represented by an activity, it is straightforward to consider Function as a stereotype extending the Activity meta-class. A system component is a SysML block, so the Component stereotype will extend the Class meta-class of UML. Because each activity may have several parameters and each class may have several attributes, we propose to use Parameter and Attribute as extended meta-classes for FailureMode stereotype. By doing so, we can represent the fact that each function and each component may have different failure modes. The other information about a failure mode such as rate, severity, causal factors, detection methods, etc. can be simply considered as the tag definitions of the Failure Mode stereotype. Figure 2.14 gives the profile diagram of our Safety Profile. This profile diagram models a simple redundancy mechanism as well as dysfunctional behavior information. It is also noted that there is no unique solution for the safety profile. We prefer to use a simple and efficient solution that allows us to represent all needed information while not overloading the XML Metadata Interchange file generated from the SysML model.

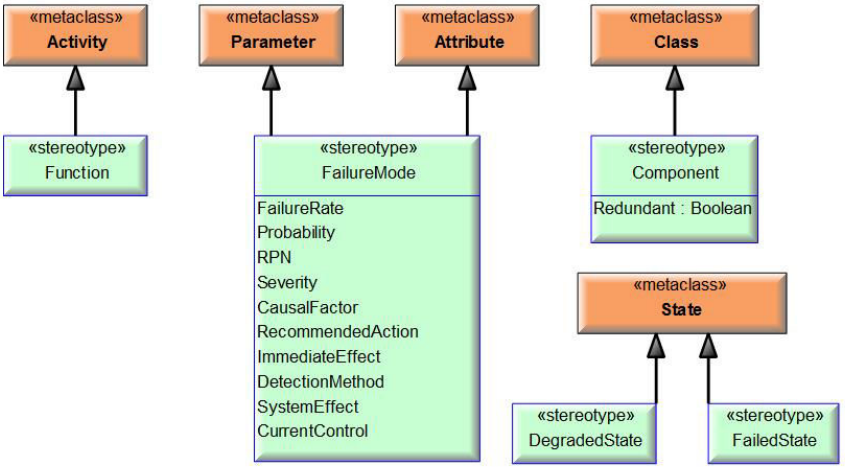


FIGURE 2.14: Safety Profile Diagram

2.4 Mechatronic Extended Modeling Profile

2.4.1 Need for Extended Modeling for Mechatronic Systems

The synergistic aspect of mechatronic systems leads to numerous linking paths and different types of flows exchanged between the system components. Connection components such as wires and tubes are used to transfer the exchanged flows among components. These interaction components, in addition to increasing system cost and weight, add a supplementary cause of failure in the system since their own failures may lead to system level malfunctions or failures. The choice of connection components can for instance affect the probability of failure occurrences [74]. Thus, if they are not accounted for in the right way, a system could be considered as safe (i.e. satisfying safety requirements) while it is not or vice versa. However, in traditional system architecture modeling, connection components, despite their importance, are not modeled, and thus they are not systematically taken into account in safety analysis since this latter is based on the structural model of the system.

Another issue that strongly affects mechatronic systems design is the multi-physical coupling. Indeed, vibration, heat or electromagnetic fields emitted by some components or by the environment could affect the other components in their neighborhood leading to malfunctioning or even failures. This could have considerable effects on system safety. However, the impact of the multi-physical couplings

on safety analysis is taken into account only by the safety expert during safety analysis. But, system designers usually have a better knowledge on the system components, their properties and the technologies in use. Consequently, integrating the multi-physical properties within the system model by system designers may reduce errors and omissions during safety analysis.

These two aspects, i.e. connection components and multi-physical flows, must be accounted for as early as possible in the design process. They should be integrated at the system-level modeling to avoid, as far as possible, late design changes that are time consuming and very costly.

In the following, we propose an extended modeling including the connection components as well as the multi-physical flows that highly impact the functioning of mechatronic systems. This modeling, we believe, will contribute to a more comprehensive safety analysis.

2.4.2 Extended Modeling for Multi-disciplinary Systems

Usually, during the design phase, the system structural models only contain the main components (those achieving a specific system function) but do not consider connection components (such as wires) that connect these main components and transfer the exchanged flows among them. However, as we have pointed up above (see section 2.4.1), connection components are critical in mechatronic / multi-disciplinary systems mainly for safety reasons. For this reason, we are convinced that connection components must figure in the system architectural model in the same way as the other components.

In SysML, physical components with different hierarchical levels (system, component, element, etc.) are represented by *Block* stereotype. In the Internal Block Diagram (IBD), the connection component between two components, represented as blocks, is represented by the *connector* stereotype, which is a path between these two blocks. Paths in SysML cannot have the same attributes as blocks (for instance, "value types" representing physical properties and/or performance parameters). To better emphasize the fact that connection components are as important as the other components, we suggest to add new blocks to model connection components instead of using the only *connectors* of SysML. These blocks may have a different stereotype to distinguish them from the other components.

Indeed, both types of components are different since, unlike the other components, connection components do not need to be allocated to functions.

The second aspect that we think it is very important to add in the system model is the multi-physical flows that can disturb the functioning of the systems and thus have an impact on the system safety. As a first step to model the potential multi-physical flows, we use the already existing flow ports in SysML to which we added a new stereotype to distinguish them from the remaining ports. A different graphical appearance is also used. Multi-physical ports have a larger size and have different colors; the red is used for thermal flow, the blue for electromagnetic flow and the grey for vibration. As components may be either aggressors or victims of multi-physical flows, a distinction is made between emitted and received (multi-physical) flows. An input port means that the component is sensitive to this kind of multi-physical aggression and must be protected from potential emissions. An output flow means that the component emits the flow in question and may be aggressive to its surroundings. Aggression from the environment is also modeled with ports on the boundary of the system. If there is no source of one particular multi-physical flow, then there should be no input port of this type.

The profile diagram of the Mechatronic Extended Modeling Profile is given in Figure 2.15. It shows the added stereotypes. Two stereotypes, the "Component" and the "ConnectionBlock" are added to extend the UML metaclass "Class", and they represent respectively the component and the connection component. A component can have different stereotypes specifying the domain i.e. mechanical, software etc. To each kind of component specific failure modes can be added automatically and integrated in the safety analyses. The stereotype "Multi-PhysicalPort" is also added as an extension to the UML metaclass "Interface" to model multi-physical ports.

The additional data integrated in the system model using the Safety Profile and/or the Mechatronic Extended Modeling Profile will be added to the FMEA generated automatically and used by the safety expert during his assessment.

The two profiles developed in this work help integrating relevant information in the system model to be used in the safety analysis. In the next sections, first a SysML-based design methodology will be presented. This methodology will be the support for an integrated approach including safety analyses within the design.

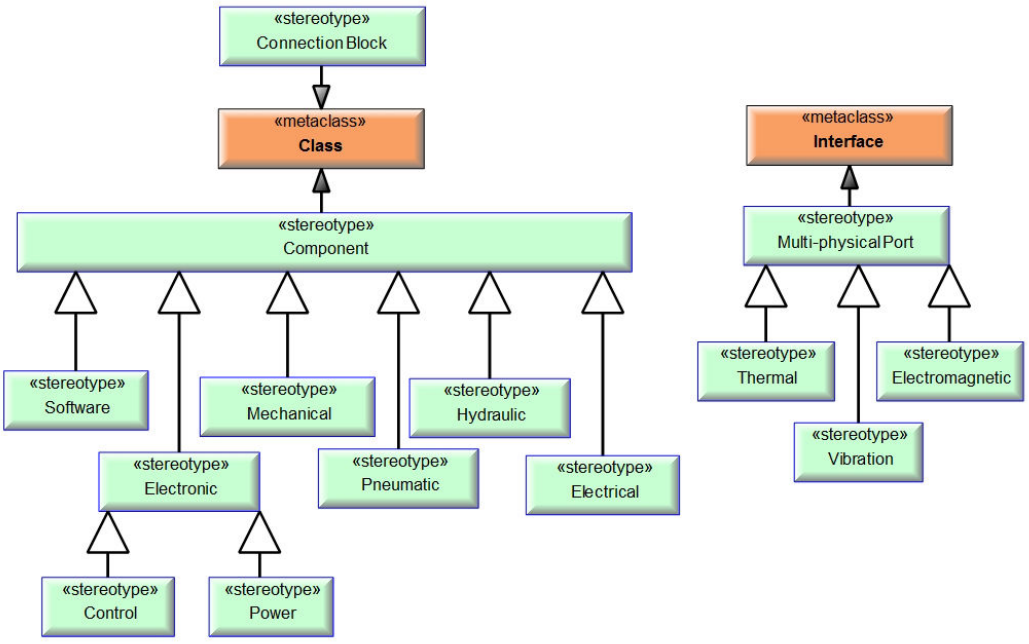


FIGURE 2.15: Profile Diagram of the Mechatronic Extended Modeling Profile

2.5 SysML-Based Systems Engineering Methodology

In Chapter 1, the main systems engineering standards and their respective processes were introduced. The benefits of a model-based approach was also presented and the adequacy of SysML as a support language for MBSE in early design stages was demonstrated. However, the availability of standards and tools is not sufficient for the successful development of complex systems. Only an appropriate methodology can lead to an optimal use of the available tools by providing a set of practices, procedures and rules to follow during the design phase. As long as we know, few methodologies were developed to describe how to achieve the appropriate processes of the available standards with SysML and most of them focus on embedded software systems design.

Other domains, like mechanical engineering, had their own approaches with powerful methodologies like APTE [1], FAST [2], SADT [3], etc. These methodologies, although relevant and very useful, are not implemented by tools allowing to build a consistent (multi-view, multi-level) model for complex systems. They were not integrated neither in the proposed SysML-based methodologies found in literature. On the other hand, some SysML diagrams can support close modeling mainly to APTE and SADT. This led us to developing our own methodology with SysML trying to take benefit of this language and the model based approach and to be compliant with the standards processes while not loosing the key concepts we have been using for several years.

The proposed SysML-based methodology is a top-down approach that, starting from a need expression progressively defines a solution to satisfy this need. It aims at assisting the designer in the system design phase to have a consistent modeling. It also attempts to guide the designer facing the variety of diagrams in SysML (given without any methodology) by giving a sequence of use of these diagrams in different design stages. The methodology is a two-phase modeling process. It begins with a *black-box* phase where the system of interest is considered as a black box, meaning that its internal structure is still not defined. This modeling phase, only gives an external point of view of the system aiming at defining a consistent set of requirements the system must satisfy, and that will be the baseline for the next phase. The second phase, called *white-box* phase is an internal point of view of the system where the internal architecture is progressively determined. Each analysis phase is made up of several steps. In each step, one or more SysML diagrams are used to describe a specific point of view of the system. The sequence of these diagrams contributes progressively in the emergence of a consistent set of requirements for the first phase and system architectures for the second one.

More detail about the two phases and how SysML contributes in performing each phase will be given below in sections 2.5.1 and 2.5.2 respectively. Even though these steps are given in a sequential way, in each step we can go back to previous steps either because a requirement is modified (which is current in industrial

[1] APTE is a French method for functional analysis and value analysis created by Gilbert Barbey in the 60s. It is based on graphs to define the need to which the system answers and identify the functions in a systematic way. More detail can be found in the URL: http://methode-apte.com

[2] FAST is the acronym of Function Analysis System Technique.

[3] SADT is the acronym for Structured Analysis and Design Technique. IT is a systems engineering and software engineering methodology for describing systems as a hierarchy of functions

projects), or due to the occurrence of changes in the market such as the appearance of new technologies or regulations or simply because the designer reminds aspects that have not been accounted for.

2.5.1 Black-Box Phase: Requirements Definition and Analysis

Bad requirements specification could lead to significant cost and schedule overruns, failures to deliver all of the functionality specified, and systems that do not have adequate quality [75]. The aim of the *black-box* analysis is to build a comprehensive and consistent set of requirements to minimize expansive design changes due to bad specification. This is made through a sequence of steps with potential iterations between them. First, the system mission and objectives are determined. Then, the whole life-cycle is identified in order to take into account the constraints of each phase of the life-cycle. The system boundary specifying what exactly is to be developed shall also be thoroughly identified since the early stages in the design process. For each phase of the system life-cycle, the system context including the interactions the system has with its surrounding shall be considered. Based on this, the external interfaces supporting these interactions shall be defined. Then, the external behavior (with regard to the user) of the system is modeled through the user operating modes, the services (or use cases) provided by the system and the functional scenarios.

Requirements emerge from a series of the modeling activities of *black-box* phase and are traced to the model elements that helped in their identification and specification. Each of these modeling activities deals with a specific aspect of the system, i.e. the mission, context, services etc. These modeling activities are described hereafter.

Initial Requirements and Global Mission

Here, the global mission (or function) of the system is identified. It is usually issued from textual, oral or partial definitions of the user needs. The main mission may be given in a hierarchical decomposition into sub-functions.

It is captured in SysML as one or more requirements with possible sub-requirements. A Requirement diagram (Req) represents these initial requirement(s) and their relationships to each other. Containment links are used to link the main requirement to its sub-requirements. Afterwards, other requirements will be identified and will progressively be captured in the system model.

Identifying the System Life-Cycle

In order to have a comprehensive set of requirements, all the system life-cycle phases as well as the stakeholders and enabling systems in each phase shall be identified, and the corresponding requirements captured. Indeed, government regulations are imposing more restrictions on different life-cycle phases of systems to take into account environmental aspects (by ensuring a recycling ratio or imposing a reduced amount of emissions for instance), safety measures (by imposing safety levels) etc. Systems built without respecting regulatory constraints may be refused the marketing authorization. Modeling the life-cycle also allows available manufacturing, assembly, testing, transportation and all other means needed during the whole system life to be considered in the design to avoid getting aware too late of problems like unavailability of transportation means for very large systems etc.

In SysML, a State Machine Diagram (STM) is used in order to define the different stages of the whole life-cycle of the system and the transition conditions from one stage to another. Each stage of the life-cycle is represented by a "state" element in SysML. Each phase of this life-cycle may then be detailed in further diagrams, all along the different steps of this modeling process.

System Context Modeling

It is very important to define the boundary of the system at the beginning of the study to identify what is within the boundary and what is outside it, in other words, what exactly has to be developed. Boundary definition prevents from doing extra work by developing things outside the boundary, or delivering partially what is actually needed by excluding some parts. Then, for each phase of the life-cycle, an exhaustive list of the external interfacing elements (stakeholders,

external systems and actors) and their interactions with the studied system are specified.

A SysML Block Definition Diagram (BDD) represents the system and its interactions with external elements. This use of the SysML BDD is partially similar to the interaction graph in APTE method. These interactions shall be considered during the design phase and specified within the requirements. Let's note that, in SysML, an actor (represented by the stick-man icon) specifies a role played by an external entity that interacts with the subject. This external entity is not necessarily a human.

The External Interfaces Modeling

In order to define more precisely the interfaces between the system of interest and the actors that interact with it, an internal block diagram (IBD) is created to complete the context BDD.

The context diagram helps in identifying, for each actor, whether external interfaces are needed or not.

In SysML, ports are used to model interfaces. Identifying the interface ports at this early stage prevents from forgetting some of them later when identifying the internal structure. The ports created here are included in the model and automatically populated later on in the design process. Consistency is thus ensured both with previous steps in the black-box phase as well as with the next white-box phase. Future changes of these ports will also be automatically propagated back to the black-box models.

The User Operating Modes

For each phase of the life-cycle (most importantly for the main operating phase corresponding to the system use), the user operating modes have to be determined and interconnected to each other using a State Machine Diagram (STM). Each operating mode is represented as a state of the system and transitions specify the conditions to go from one operating mode to another. The system operating modes detail the usage of the system during its operating stage of its life-cycle.

The "operating modes" state diagram is then linked to the corresponding state of the life-cycle.

The Services Provided by the System

For each user operating mode, each services provided by the system to the end-user is modeled by a Use Case. The dependency relationships among use cases as well as the actors that are involved in each use cas are represented in a Use Case Diagram (UCD). To ensure consistency with the system operating modes, each use case is linked with the relevant operating mode (i.e. state).

The Functional Scenarios

For each service (or use case), a sequential description of functional scenarios may be defined with one or more sequence diagrams (SEQ). Sequence diagrams are strongly coupled to the UC they detail. In a sequence diagram, the interactions between the system and its context (external actors and/or systems) are detailed. The same actors that are linked to the use case shall be on the corresponding sequence diagram. Internal operations may also emerge from this diagram and will be used in the functional model during the white-box analysis.

Requirements specification

All the information captured within the previous diagrams shall be elicited in the model database as requirements. The set of requirements has to be structured with the appropriate links among: *derive*, *refine* and *contain* relationships. These relationships can be displayed in a requirement diagram that summarizes the initial requirements (representing the global mission) as well as the requirements derived from the other diagrams of the black-box analysis like the UCD for the system services or the interactions of the BDD context.

Requirements Traceability

In order to be able to trace all requirements, and ensure consistency of the black-box phase, some relationships are specified in some additional requirements diagrams. The following traceability links can be used:

- *"refine"* between Requirements and UC;
- *"allocate"* between Requirements and roles in BDD context;
- *"satisfy"* between Requirements and external ports (IBD).

2.5.2 White-Box Phase: Architectures Definition

In the previous phase, an external point of view analysis was established in order to identify a comprehensive set of requirements and specifications of the system. A baseline is now available to go further and identify different candidate solutions. This modeling phase is conducted with an internal point of view on the studied system, in order to model its internal structure and behavior, with respect to the set of requirements (REQ) specified during the black-box analysis.

In the *white-box* analysis, the system architecture is progressively identified. First a hierarchical model of the system functions breakdown is established. Then, based on the functional model, components are allocated to functions to synthesize candidate logical architectures. By logical architecture, we mean an architecture based on general classes of components and not specific fully defined components. Based on trade-offs and further simulations with external tools, physical architecture is defined by allocating the optimal physical (existing COTS or designed) components to the logical ones. During this phase, new requirements may emerge and shall be traced to the black-box requirements and to the related model elements.

Functional Architecture Definition

In this step, the functional requirements identified in the black-box analysis (from use cases and operations identified on sequence diagrams)are translated into a coherent description giving the functional architecture of the system. Systems

functions are represented by means of activities, and shall be linked to the corresponding requirements. The functional architecture is defined with activity diagrams (ACT), with a top-down breakdown of the main function of the system into sub-functions. Activity diagrams show the progressive transformations of input flows into output flows. Both object flows and control flows are supported. External PINs must be consistent with the external ports of the previous external interfaces IBD.

The Logical Architecture Definition

- Logical breakdown and allocation to functions

  In this step, system functions are allocated to logical components. By logical component we mean a general class of components, mainly a kind of technology: motors, gears... etc. Thus, a set of logical components is chosen in order to achieve all functions specified in the functional breakdown. Allocations between activities and components can be displayed in a Block Definition Diagram (BDD). In this diagram, one or more functions (activities) are allocated to one logical component. At this step, several candidate logical structures can be proposed and then compared. Moreover, and to ensure consistency with black-box analysis, the operations identified in sequence diagrams can be allocated to relevant components.

- Requirements to Logical Components Traceability

  Once the logical components are identified and allocated to activities, component level requirements are derived from system-level requirements and traced to the corresponding components. The logical components shall satisfy these derived requirements. *Satisfy* relationships are used to show this dependency and are displayed in a new requirements diagram (REQ). New requirements can also emerge from the choice of the logical components and shall then be added to the requirements database and linked to the corresponding components.

- Logical Architecture

  The logical components are now identified, but the way they interact is still not specified. In this step, the internal interactions between the components

are given. An internal block diagram (IBD) displays the system architecture with the interactions among components and the different flows they interchange with each other.

The physical Architecture Definition

After having checked with simulation (Modelica, Simulink, HIL...) that the logical architecture is relevant to fulfill the set of requirements, it is time to choose physical components (components off-the-shelf (COTS), machined or molded parts, e.g. instances with suppliers references) and allocate them to logical components.

An internal block diagram (IBD) allows specifying the interactions (flows) between the physical components. In this case, consistency shall be maintained by allocating the physical components and ports to the corresponding logical ones and justify if new components or ports are added.

2.5.3 Methodology Discussion

The proposed methodology gives a way to translate progressively user needs (or an initial set of requirements) into a feasible, well defined solution that takes into account the different aspects and constraints concerning the whole life-cycle of the system. A black-box analysis firstly aims at identifying a comprehensive set of requirements. A white-box analysis is then performed on order to define the system behavior and structure. It should be bared in mind that it is question of an iterative process. Some diagrams may reveal information that wasn't considered in previous steps. The designer shall then go back to add the missing information and check all the impacts of this change on the model. The sequential aspect presented here aims at making the understanding easier for the reader. The proposed methodology addresses the following problems. First, it gives a kind of road-map to help persons intending to use SysML choosing the right diagrams (among the variety of available diagrams) for the right purpose. It also highlights how to build a consistent model and maintain consistency between the different views of the system during all the design phase. Finally, the solution is defined progressively based on a comprehensive set of requirements thoroughly determined and a hierarchical functional modeling.

The presented methodology however, is not comprehensive and shall be extended with additional support processes. It is kind of the basic skeleton on which other engineering processes can be added. For instance, it does not yet offer processes for ranking and prioritizing requirements, establishing trade-offs and performing safety analyses.

Other works on metrics are also developed by our team to help the designer to take decisions and trace his choices [76] and will be integrated within this design methodology.

For a seamless process, some tools, like Model Center for instance, are developing bridges between SysML and other modeling and simulation tools. This is a good point that the authors are exploiting to further extend the current methodology.

In this thesis, we are mainly interested in the integration of safety analysis and a proposal of integrated process has been developed and is presented in the next section.

2.6 SafeSysE: The Integrated Process

In this section, the integrated process of systems engineering and safety analysis is presented through a set of steps. In Figure 2.16, an activity diagram describes this integrated process and the sequencing as well as the exchanges between the different steps. Swim-lanes are used to make a distinction between systems engineering and safety analysis activities (or processes). The integrated process starts with a requirements definition and analysis process with, as a starting point a set of initial requirements describing the need. Then, the different steps including design activities and safety analyses are performed successively. Data stores are used to model the storage of the different artifacts issued from each activity.

2.6.1 Step 1: Requirements Definition and Analysis

This step is the initial step of the design methodology presented in section 2.5 where system functionalities as well as its external interfaces are described by a set of requirements. Several SysML diagrams such as use case diagrams and block definition diagrams for the system context can be used to help in the identification

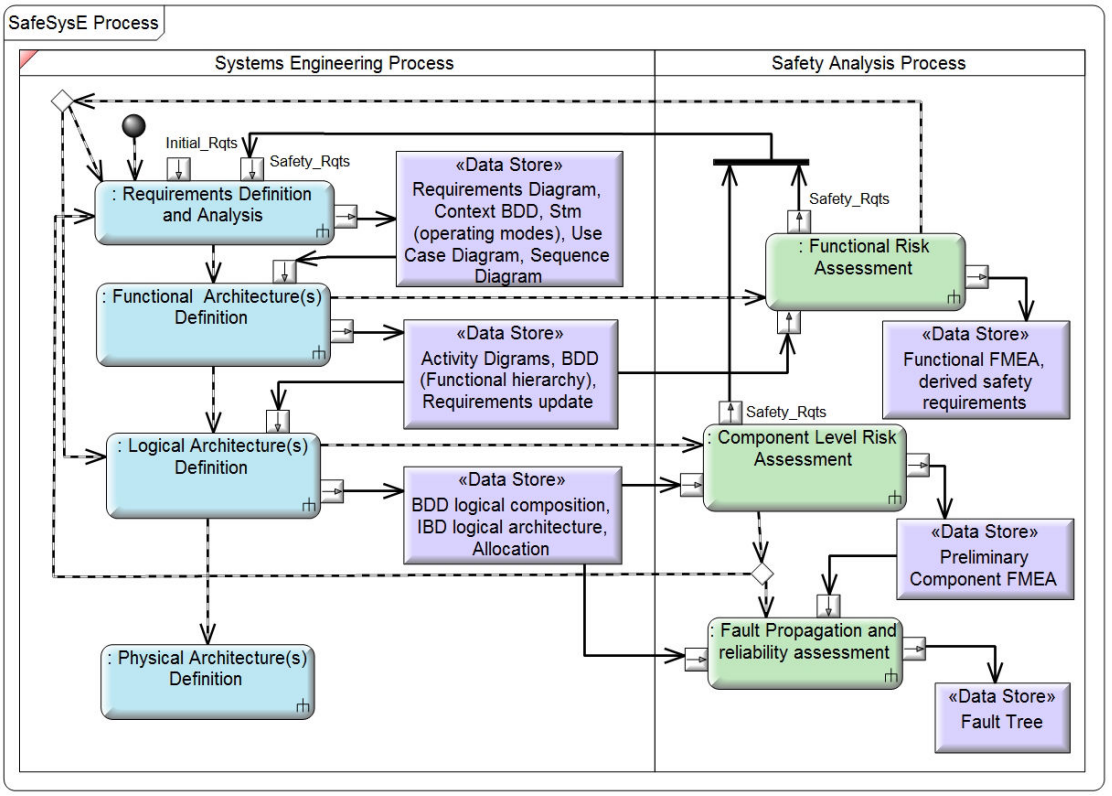


FIGURE 2.16: SafeSysE Integrated Process

of these requirements. Since, the requirement definition and analysis is not in the main scope of this thesis, please refer to section 2.5 ot to our paper about SysML-based systems engineering methodology in [77] for more detail. Furthermore, we will illustrate this step through the Electromechanical Actuator (EMA) case study found in Appendix A.

2.6.2 Step 2: Functional Architecture Definition

Based on the functional requirements identified in step 1, one or more functional architectures are identified during this step. The final result is a hierarchical model of the breakdown of the system main function(s) into sub-functions. In SysML, functions are represented by activities and the functional breakdown is modeled through a set of activity diagrams, each activity diagram representing the breakdown of a given function (activity) into sub-functions. Activity diagrams also show the progressive transformation of input flows into output flows.

2.6.3 Step 3: Functional Risk Assessment

In this step, a functional Failure Mode and Effects Analysis (FMEA) is used to identify potential hazards caused by failures and their effects. In this work, a prototyping tool has been developed to automatically generate partial FMEA based on the XML Metadata Interchange XMI [78] file generated from the SysML model. The generated FMEA data-sheet contains the list of functions and a generic list of failure modes. The safety expert then performs the analysis and completes the FMEA with the relevant data. All these new safety information are then updated into the SysML model via the safety profile extension explained in section 2.3 and the developed tool. The gap between safety analysis and design modification is shorten, thanks to this integrated model. At the end of this step, safety requirements are derived and added to the set of requirements. The rule is that for each failure mode with hazardous effects, at least one safety requirement is added. Design changes can be done from this early design stage at the functional level to eliminate or reduce identified risks. Risk effects mitigation can be obtained by eliminating or modifying high risk functions, adding new fault tolerance mechanisms like diagnosis and reconfiguration functions, etc. Each time that the functional architecture is modified, the FMEA shall be updated to take into account the new changes. The previous steps iterate until a satisfactory solution is identified and the final results are stored in the system model.

More detail about the functional FMEA generation will be given in chapter 3.

2.6.4 Step 4: Logical Architecture Definition

Once the functional architecture is defined taking into account the results of the safety analysis in step 3, one or more logical architectures are defined by allocating components to functions. A Block Definition Diagram (BDD) describes the components of the system and an Internal Block Diagram (IBD) describes the interactions between the components. The logical architecture defined at this step already takes into account safety aspects since it integrates the results of the functional safety assessment performed in step 3.

2.6.5 Step 5: Component-level Risk Assessment

When the structure of the system is defined, the safety analysis results are updated and a component level risk assessment is performed. For this purpose, a component FMEA is generated from the XMI file like in step 3 for the functional FMEA. To ensure consistency with previous safety analysis, the generated FMEA, in addition to the components, contains in front of each component the functions allocated to the component as well as the failure modes identified at the functional level as a reminder. The safety expert then identifies the failure modes at the component level and performs FMEA analysis. If there are identified risks with unacceptable level, then these risks shall be eliminated or reduced to an acceptable level by performing changes to the design. Once again, these safety data are saved back in the same SysML model using the safety profile developed in this work. If design changes are performed (by going back to previous steps), a new FMEA is generated to assess the new architecture. In this case, the previous results are also automatically generated as they are stored in the model and the safety expert updates the FMEA without loosing his previous work.

More detail about the component FMEA generation will be given in chapter 3.

2.6.6 Step 6: Fault Tree Analysis

The final step is the fault tree analysis. Fault trees are used for both qualitative and quantitative analyses. In our approach, fault trees are automatically generated from SysML IBDs describing the system architecture. Information from the previous FMEA analysis is taken into account to create fault tree with specific failure modes. Fault trees can be generated in a graphical form for qualitative analysis purposes like fault propagation studies and critical paths identifications. They can also be generated in an appropriate format for existing fault tree analysis tools. For more details about fault tree generation please refer to Chapter 4.

2.7 Conclusion

In this chapter, the ability of SysML to support MBSE for early design stages is demonstrated. Indeed, in addition to its rich semantics to model the most common

aspects of systems, SysML offers extension mechanisms to add new semantics to enlarge its use to specific domains. In our case, an extended modeling enabling to take into account the particular features that characterize multidisciplinary systems was needed. Including multi-physical interactions (i.e. thermal and EMC) among the system or with its environment as well as the interconnection components is very useful for safety analysis. An extended modeling for mechatronic systems was proposed in this chapter. Moreover, a safety profile is proposed to integrate safety-relevant properties in the system model to facilitate the automatic generation of safety artifacts discussed in the next chapters.

Finally, an integrated process including system engineering and safety analysis is proposed. The integrated process provides the safety experts with FMEA and FTA consistent with the system behavior and architecture, thus allowing them to deal with an up-to-date and consistent modeling of the system and its safety aspects.

Chapter 3

FMEA Generation from SysML Models

3.1 Introduction

Failure Mode and Effects Analysis (FMEA) is a reliability tool widely used in safety analysis. In addition to being mandatory in several safety critical domains, FMEA is very useful since it allows an early identification of risks. It focuses particularly on risks resulting from potential failure modes. Early identification of risks avoids the embarrassment of discovering problems very late, which requires complicated and costly correction processes. Consequently, it allows reducing the cost and time for design changes and thus for the whole development process. To take full benefit of the FMEA, it is critical to conduct it at early design stages. However, as the design continues to evolve, the reliability analysis should be continued concurrently with the design process. FMEA should then be updated to take into account design changes. Different FMEAs can be performed during the whole development process beginning with a functional FMEA and then performing one or more component FMEAs with different levels of detail as long as the design process progresses.

Despite its advantages, an FMEA can be a source of delay in the project. Indeed, it can be laborious and very long to perform mainly when done manually for complex systems. In addition to being resource and time consuming, when done manually, it is also error-prone since it requires the extraction of the whole list of the items

to be analyzed as well as a detailed understanding of the system functions and design.

To cope with these problems, the system models should be expressive enough to ensure a good understanding of the system functioning and facilitate the extraction of the information required for the establishment of the FMEA. As stated previously, SysML is a good candidate for system modeling. It supports systems modeling semantics and can easily be extended to support the integration of reliability (FMEA) semantics. For instance, the safety profile developed in this work enables modeling safety relevant concepts.

An optimal reduction of time and errors during the FMEA elaboration would be an automatic generation of the FMEA from system models. However, unless the system model is charged with all the information required for the FMEA, it can unfortunately not be fully automated. Integrating all the information relative to safety in the system model requires the safety expert to collaborate closely with the design team.

Automating some steps of the FMEA generation is then a good balance allowing to reduce development time and error proneness while maintaining the role of safety expert in completing and reviewing the generated FMEA. The automated steps in this work are mainly the extraction of the list of items to be analyzed (functions or components) and any important information available in the system model such as the input/output flows of each item or any safety relevant information added via the use of the Safety Profile.

Another problem when performing FMEA is that it is a hard task to maintain consistency with the system model. In our work, as the preliminary FMEA is automatically generated from the system model, the consistency is then ensured. Information added by safety experts to complete the generated FMEA will then be sent back to update the SysML model via the Safety Profile. Consistency is also ensured between functional and component FMEAs since the component FMEA integrates the functions of each components and, for each function it includes the functional failure modes. The safety engineer has to translate the failure modes identified at the functional level into the component level.

This chapter deals with the integration of FMEA within the design process. First it gives an overview on the FMEA technique, its purposes, the different steps of performing an FMEA as well as the different kinds of FMEA in section 3.2.

The related work about the automatic generation of FMEA is given in section 3.3. Then, the automatic generation of functional FMEA and component FMEA proposed in this work are presented respectively in sections 3.4 and 3.5. Both sections are illustrated with the EMA case study. The chapter is finally concluded in section 3.6.

3.2 Failure Mode and Effects Analysis (FMEA)

Failure Mode and Effects Analysis (FMEA) is a popular tool for reliability and failure mode analysis [79]. It was one of the first systematic techniques for failure analysis. It was created in the 1950s by reliability engineers to better study and prevent the consequences of malfunctions and failure in military systems and evaluate the impact of these failures on system reliability. It was then utilized in several other domains and is now a common practice for most, if not all, safety critical projects. It can also be required by regulations or customers.

The FMEA method is a disciplined bottom-up evaluation technique that can be applied at any level of detail in the design. It is applicable to functions, components, assemblies or sub-systems. FMEA is frequently used in analyzing hardware and processes but it can also be used in software to evaluate the effects of software functions failures.

The aim of the analysis is to identify potential unacceptable reliability, safety or operation conditions resulting from the identified failure modes. This requires a deep understanding of system functions and design. Consequently, for a successful failure modes analysis, the FMEA report should be performed by a mixed multi-disciplinary team including at least safety experts and system designers. If unacceptable risks are identified, then a list of prioritized corrective actions to eliminate or reduce these risks is provided.

Risks cannot be completely eliminated and then it should be brought to a level ALARP (As Low As Reasonably Practicable). Practicable means what is possible to do, for instance reducing frequency of occurrence and/or the consequence of the event. Reasonable means to have a balance between the cost, time, trouble, etc. resulting from reducing the risk and the benefit from eliminating or reducing that risk. This means that risk is tolerable if it is demonstrated that there's a big disproportion between the cost of further risk reduction on one hand and the

resulting risk reduction benefit on the other. If the quantified risk of injury is insignificant compared to the measures needed to mitigate the risk, then no action needs to be taken. "However, the greater the risk, the more likely it is to be reasonably practicable to go to substantial expense to do something about it" [80].

These actions imply new iterations in the design process to change the design in order to eliminate the risk or bring it to acceptable levels. Performing FMEA requires time and manpower resources. To be efficient, these efforts should be performed as early as possible during the design process so that they influence design choices and decisions in real time, thus avoiding complicated and costly late changes. If the problems are not discovered and solved early enough, they could be discovered at very late stages like the production stage or even the product warranty phase. The later the corrective actions occur, the more complicated and costly they are.

In addition, the FMEA provides a good support document capturing recommended design changes and their reasons. "Time and resources for a comprehensive FMEA must be allotted during design and process development, when design and process changes can most easily and inexpensively be implemented" [23].

3.2.1 FMEA Process

FMEA is performed in three main steps. First, the process identifies the potential failure modes of a system function or component during its life-cycle. Then, it determines the effects of each failure mode. Finally, the criticality of the effects is evaluated [79].

The MILSTD-1629A standard gives a detailed description of the steps of performing an FMEA within the other design activities as follows [81]:

- First, define the system to be analyzed giving its internal and interface functions, expected performance, etc.;
- Construct models or diagrams that illustrate the operations, interrelationships and dependencies of system functional entities;

- Identify all potential failure modes for each item and the effect of each failure mode on the function or item itself, on the system in study and on the system mission;
- Evaluate the worst potential consequence of each failure mode and assign a severity class category. There are four severity categories consistent with the MIL-STD-882D standard [82]: catastrophic, critical, marginal and minor;
- Identify failure detection methods;
- Identify corrective actions to eliminate the failure or control the corresponding risk;
- Evaluate the effects of corrective actions on the system.

As a result, the FMEA is delivered as a table. Standards like the Military standard MIL-1629A specify the table contents, but other forms adapted from the standard can be found in the literature. The rows of the worksheet contain the analyzed entities, their failure modes and the corresponding information relevant to the FMEA. A list of columns contains the items properties to be integrated in the analysis.

In the following, some semantics about functional FMEA and component FMEA are given respectively in sections 3.2.2 and 3.2.3.

3.2.2 Functional FMEA Semantics

In a functional FMEA, the analysis focuses on the different functions the system performs to achieve its mission. The starting point for a functional FMEA is the functional architecture of the system defined by the designers (in our work this architecture is defined in the *Functional Architecture(s) Definition* step of the integrated methodology in section 2.6). The functional architecture describes the list of the different functions the system must perform to achieve its mission as well as the input and output flows of each function. Consequently, it describes how the system inputs (flows received by the system from its surrounding, i.e. users and contributing systems) are progressively transformed into outputs (the flows or services delivered by the system to its surrounding).

The FMEA then analyzes the potential failure modes of each function in order to identify the causes and effects of each failure mode. The list of functions is then inserted in the FMEA. For each function, the potential failure modes are identified. Then, for each failure mode, the causes and effects are determined and then the criticity of the failure mode is evaluated. If the failure mode has negative effects at the system level, then the safety expert tries to identify a list of corrective actions. This list is prioritized according to the criticity of the failure modes.

The FMEA table contains several columns, each of them describing a particular aspect relative to the function potential failures. Definitions of the different terms found in an FMEA are given below.

A functional FMEA table is given in Figure 3.1.

Id	Function	Failure Mode	Causes	Local Effects	System Effects	Severity	Occurrence	Detectability	Criticity	Detection Method	Corrective Action

FIGURE 3.1: Example of Functional FMEA Table

- **Failure mode**

  A failure mode is defined in [23] as "the manner by which an item fails; the mode or state the item is in after it fails. The way in which the failure of an item occurs".

  The first step of conducting an FMEA is the identification of the different failure modes of each item. The failure modes can be determined in different ways. They can for instance be determined based on existing data provided from past experience on similar products. Some standards can also collect such data from the experience of several industrialists and provide a list of classified functions with their respective failure modes. When such data is available, it can be of a big help since it allows taking benefit from past experience and preventing some mistakes. However, this is not always the case, mainly for totally new products. In this case, the knowledge of the safety expert with the FMEA team is the only source of information to

rely on. A systematic approach can be also used as a complement based on generic guidelines usually provided by the standards. Indeed, at the functional level, failure modes are quite generic and can be resumed in the following list [50], [81]:

- The function fails to execute;
- The function executes but not with the required performance (either with superior or inferior performance);
- The function is not executed at the required time (too late or too soon);
- The function is stopped during its execution;
- The function should stop but it still runs;
- The function runs in an intermittent way;
- Other failure modes proper to the function.

- **Failure Cause**

  A failure cause is a process or mechanism responsible for initiating the failure mode. The possible processes that can cause function failure include physical failure, design defects, manufacturing defects, environmental forces, and so forth [23].

- **Failure Effects**

  The failure effects are the consequence(s) a failure mode has on the operation, function, or status of an item and on the system. The failure effects can be noticed at the local level and at the system level [23].

  The "Local Effects" column identifies the most immediate and direct effect of the indicated failure mode. This is the low-level effect that occurs on the next item in the design.

  The "System Effects" column identifies the consequences of the failure mode at the system level.

- **Detection Method**

  This column identifies how the specific failure mode might be detected after it has occurred and before resulting in any serious consequence. If a method of detection is possible, it may be used in the mitigating design.

Id	Component	Function	Failure Mode	Causes	Local Effects	System Effects	Severity	Occurrence	Detectability	Criticity	Detection Method	Corrective Action

FIGURE 3.2: Example of Component FMEA Table

- **Recommended Actions**

  This column identifies methods for eliminating or mitigating the effects of the potential failure mode.

3.2.3 Component FMEA Semantics

The component FMEA focuses on the system components rather than the system functions. It is performed when the design process is more advanced and when the system components are already defined. It is linked to, and should be consistent with the functional FMEA. Indeed, since the components are allocated to the system functions they achieve, their failure modes correspond to the functional failure modes but expressed in more detail taking into account the physical aspects. The semantics of component FMEA are quite similar to those of functional FMEA with the difference that component FMEA is focused on components rather than functions and thus is closer to the technical aspects. All term, i.e. failure modes, causes, effects etc., are more specific and linked to the chosen technology. In the Component FMEA the columns are then slightly different from the functional FMEA, and for instance, the column "Component" is added.

An example of component FMEA is given in Figure 3.2.

3.2.4 Failure Mode Effects and Criticality Analysis

A more detailed version of the FMEA is the Failure Modes, Effects and Criticality Analysis (FMECA). FMECA is quite similar to FMEA except that it adds a criticality evaluation for each failure mode. FMECA is based on three factors: *severity*,

showing how serious the consequences of failures may be, *occurrence*, showing how frequently failures occur and *detection*, showing how easily failures can be detected. Then the Risk Priority Number (RPN) is calculated by multiplying these three factors.

$$RPN = (probability\,of\,occurrence)\,.\,(severity\,ranking)\,.\,(detection\,ranking)$$

The RPN is an index for reliability that helps in prioritizing the corrective actions; problems are addressed from the biggest RPN to the smallest one. Failure modes with catastrophic effects but very low occurrences may have a low RPN. However, these failure modes should be at the top of the list of failures to deal with despite their low RPN.

3.2.5 Advantages and Limitations of the FMEA

FMEA is a valuable reliability tool for analyzing potential individual failure modes and providing basic information to reliability prediction. This information including potential failure modes can be used in fault tree analysis (FTA) for instance. However, for safety purposes, FMEA technique is limited since it considers only single item failure and is unable to deal with combination of failures of different items. In real life however, accidents generally result from a combination of failures. FMEA does not neither consider hazards arising from events other than failures (e.g. timing errors, high voltage, etc.) [23].

Consequently, FMEA should not be the only hazard identification technique. It should be used in conjunction with other complementary tools like FTA.

3.3 Related Work about Automated FMEA Generation

FMEA is a valuable technique for predicting system reliability and evaluating the impact of potential failure. However, it is time and resource-consuming and error prone. To manage these drawbacks, several works attempted to automate some steps of generating FMEA. This section discusses some of these works.

David et al. in [83] carried out a study on the automated generation of FMEA from UML functional models. At that time, SysML was too recent and not yet widely implemented and used. As UML is firstly dedicated to software modeling, they faced some problems to model systems, mainly the problem of flow representation that is not supported in UML. This constituted a limitation because the study of the physical flows is crucial for fault propagation studies (to determine system level effects of a particular failure mode) and for deducing common failure modes. They had to create their own stereotypes to manage this lack of representation in UML.

When SysML was more widely used, David et al. [49, 50] worked again on the generation of an FMEA report from system functional behaviors written in SysML models. With the use of SysML, the problem with modeling flows was solved. They also worked on the construction of dysfunctional models by using the AltaRica language in order to compute reliability indicators. In their methodology called MéDISIS, they start with the automatic computation of a preliminary FMEA. The structural diagrams, namely Block Definition Diagram (BDD) and Internal Block Diagram (IBD), and the behavioral diagrams such as Sequence Diagram (SD) and Activity Diagram (AD) are analyzed in detail to give an exhaustive list of failure modes for each component and each function, with their possible causes and effects. Then the final FMEA report is created with help from experts in the safety domain. To facilitate a deductive and iterative method like MéDISIS, a database of dysfunctional behaviors is kept updated in order to rapidly identify failure modes in different analysis phases. The next step of their work is the mapping between SysML models and AltaRica data flow language, so that existing tools to quantify reliability indicators such as the global failure rate, the mean time to failure, etc., can be used directly on the failure modes identified in the previous step.

The European COMPASS (Comprehensive Modelling for Advanced Systems of Systems) project [1] tackled the safety analysis of Systems of Systems (SoS). In this project [84], a safety profile was developed to annotate the SysML model of the SoS by safety relevant information. The annotated SysML model is then processed by the external HiP-HOPS tool to automatically generate FMEA tables.

[1] http://www.compass-research.eu/

3.4 Automated Functional FMEA Generation in SafeSysE

3.4.1 Functional FMEA Generation : Implementation

Functional FMEA aims at identifying potential failure modes and their potential causes and local and system level effects since the functional stage in the design process. It can be performed once the functional architecture of the system is established. In this work, we propose an automatic generation of a preliminary functional FMEA from the functional model in SysML. As seen in section 2.5, we made the choice to model the functional breakdown by SysML activity diagrams. A progressive breakdown of the system functions with several levels of detail is performed. The decomposition is stopped once the designer is able to allocate components to the functions in an appropriate manner (respecting the rules mentioned in section 2.5). This results in a tree like hierarchical representation of the functions. In our study, we consider all the leaf functions in the FMEA generation because we consider that the failure modes of the higher level functions are the result of the failure modes of the lower level ones.

The starting point of the FMEA is the list of entities to be analyzed. In the case of a functional FMEA, this corresponds to the list of the system functions.

As stated in the design methodology, the functional breakdown of the system mission into sub-functions is modeled with the activity diagram, used as an EFFBD. In this way, in addition to the functions list, we have the input/output flows of each function and we can see the way the system progressively transforms inputs into outputs. This kind of information is very important to understand the system functioning but it is also very important for safety analysis as it helps in predicting failure propagation.

For the automatic generation, first the SysML model is exported into an XMI [2] file [78]. Since the most common usage of XMI is to exchange metadata for UML models, it is straightforward for us to work with XMI files generated from a SysML

[2] The XML Metadata Interchange (XMI) is an Object Management Group (OMG) standard for exchanging metadata information via Extensible Markup Language (XML). The last one is a markup language that defines a set of rules for encoding documents in a format that is widely used for the interchange of data over the Internet, thanks to its simplicity, generality and usability.

modeling tool such as Artisan Studio. A program, that we called SafeSysE Tool is developed in this work to help in performing safety analyses. This tool interacts with the SysML through the generated XMI file. In this way, our program works directly on the XMI file and is thus tool-independent.

In an XMI document, after the header section containing information about the versions of the standards and the tool that created it, we have the UML and SysML sections that describe the model itself. Figure 3.3 shows an excerpt from an XMI file where we can see how data is organized in a tree structure. Each node of the tree is an XML element and is written with an opening and closing tag. An element can have one or more XML attributes such as xmi:type, xmi:id and name. Each attribute has a value and the id of an element is unique.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version = "2.1" xmlns:xmi = "http://schema.omg.org/spec/XMI/2.1" xmlns:uml = "
http://www.omg.org/spec/UML/20090901" xmlns:sysml = "http://www.omg.org/spec/SysML/20100301/SysML-profile"
xmlns:StandardProfileL2 = "http://www.omg.org/spec/UML/20090901/StandardProfileL2" >
  <documentation xmi:type = "xmi:Documentation">
    <exporter>Artisan Studio</exporter>
    <exporterVersion>7.4.4</exporterVersion>
  </documentation>
  <uml:Model name = "EMA Reduced" xmi:id = "_1ea6af89-9583-446f-a9d4-6c15e80b7a14">
    <packagedElement xmi:type = "uml:Package" xmi:id = "_33e4937e-3177-47c4-9681-690ae433803e" name =
    "EMA Reduced">
      <packagedElement xmi:type = "uml:Package" xmi:id = "_ec5c18ce-3b5f-44fa-a897-e7a6ecfc6e87" name =
      "Behavior">
        <packagedElement xmi:type = "uml:Package" xmi:id = "_18e1e6d4-ec5a-499d-a9ea-44d90346f9a1" name
        = "Activity">
          <packagedElement xmi:type = "uml:Package" xmi:id = "_a6f696d2-e18a-48dd-96ba-6ffec8bf0bc7"
          name = "Arch-V1">
            <packagedElement xmi:type = "uml:Activity" xmi:id = "_fe034956-08e2-415b-a0da-559920a41171"
            name = "ControlAileronIncidence-V1" isReentrant = "false">
              <ownedParameter xmi:type = "uml:Parameter" xmi:id =
              "_f5cac098-c639-4eff-a4a4-3b46771281e8" name = "ElecPwr">
              </ownedParameter>
              <ownedParameter xmi:type = "uml:Parameter" xmi:id =
              "_6357131d-1b83-447b-818a-c347a2e71e2d" name = "PilotInstructions">
              </ownedParameter>
              <ownedParameter xmi:type = "uml:Parameter" xmi:id =
              "_d1e62180-8d7a-4ebc-a4d1-5688a04d70f9" name = "MechPwr" direction = "out">
              </ownedParameter>
              <ownedParameter xmi:type = "uml:Parameter" xmi:id =
              "_d4e93226-1c16-4c16-bdb0-297c6a432d56" name = "F_Back_to_CtrlU" direction = "out">
              </ownedParameter>
              <ownedBehavior xmi:type = "uml:Activity" xmi:id = "_5b0ed19f-3917-45b2-bb8d-f91b48efed7d"
              name = "Control and command" isReentrant = "false">
                <ownedParameter xmi:type = "uml:Parameter" xmi:id =
                "_a23814f0-3f93-4ee3-b6b8-b4b8a95b3a74" name = "PilotInstructions">
                </ownedParameter>
…
      </packagedElement>
```

FIGURE 3.3: Excerpt from an XMI File

SafeSysE Tool parses the XMI file by using *Beautiful Soup* [3], a Python library that provides methods to navigate, search and modify a parsed tree. For each

[3]http://www.crummy.com/software/BeautifulSoup/

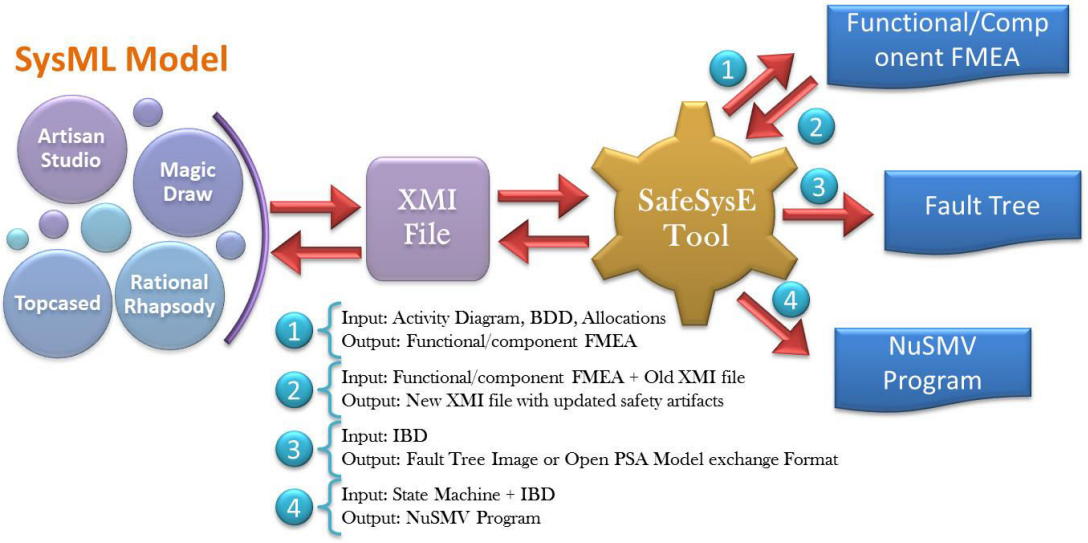


FIGURE 3.4: SafeSysE Tool

functionality, the program builds a graph with nodes and edges containing all information needed for the generation of safety artifacts. Figure 3.4 shows the program's main functionalities with the corresponding input and output data. The FMEA generation functionality is detailed hereafter. The other functionalites will be explained in detail in the corresponding chapters.

Functional FMEA Generation

The first step using SafeSysE Tool is to select the XMI file corresponding to the model to analyze. SafeSysE Tool analyzes the given model and displays the different packages of the model. The user can choose to generate a functional FMEA for a particular package that represents a specific functional architecture solution. If the selected package does not include a functional breakdown, then an error message is generated and the user is asked to choose a new package. The algorithm realizes the following two main steps:

1. Activity Diagram Extraction:

   From the node corresponding to the chosen package in the parsed tree, the program extracts all the information related to activity diagrams and stores them in a graph. A node in the graph can be an activity, an action, etc which is made up of its identity (id), name, type and the activity it belongs to. If the node is an action node, input and output pins are also collected as nodes in the graph. The edges of the constructed graph represent all possible relationships between nodes.

2. XLS File Generation:

   This step generates an .xls file corresponding to the FMEA worksheet by creating columns and adding information, when available, for each function in the given columns. For the functional FMEA, the column headers are : "Function", "Function failure mode", "Causal factors", "Immediate effects", "System effects", "Recommended actions" and "Severity" but this list can be modified if we want to add other information. The pre-filled columns are described hereafter:

   - Function: Functions (represented by activities) are extracted from the graph built in the previous step "Activity Diagram Extraction". Through the edges connecting different nodes, information about input and output pins as well as predecessor and successor activities is used to fill the other columns.
   - Function failure mode : The list of generic functional failure modes is saved in a configuration file. By default, the configuration file contains the following failure modes: "Fails to perform", "Performs incorrectly

(degraded performance)", "Operates inadvertently", "Operates at incorrect time (early, late)", "Unable to stop operation", "Receives erroneous data", and "Sends erroneous data". This list can be customized by editing the configuration file. These generic failure modes are automatically inserted into the failure mode cells of each function in the automatically generated functional FMEA. If any other specific failure modes of the system functions are noticed, they can be added later by safety experts directly in the FMEA worksheet.

- Causal Factors : In the "Causal Factors" column, the input and output parameters of the current activity/function are inserted. This does not correspond to the final information required in the functional FMEA but they are added to help the safety expert to be exhaustive in finding all possible causes of failures.
- Immediate effects : As the causal factors, we pre-fill the immediate effect cells by the upstream and downstream activities which are direct predecessors and successors of the current function, respectively. It means that a failure mode of a function can cause immediate effects for the functions that are related with the current function by flow controls.

Once the preliminary worksheet is generated, the safety expert then completes the FMEA by adding the relevant information. The FMEA team first checks the list of the automatically generated failure modes and removes irrelevant failure modes and/or adds new failure modes that have not been considered if any. Then, they add the causal factors of each failure mode. As failure can propagate with the system flows, the failure in one function can be caused by a failure of the upstream functions. For instance, because of failure, the upstream function could send a wrong flow causing an overload and consequently leading to a failure of the function in question. In some cases, a function can also fail because of the failure or degraded operating of the downstream functions.

The next step is analyzing each failure mode and determining the failure effects at the local and system levels and the possible corrective actions to eliminate or reduce the risks caused by each failure mode.

Finally, the severity of each failure mode is assessed in order to identify critical functions and prioritize the list of corrective actions.

3.4.2 Functional FMEA Generation: Case Study

In this section, the approach is illustrated with the Electro-mechanical Actuator (EMA) presented in Appendix A. After the requirements definition and analysis process, the functional analysis of the EMA is performed. A functional breakdown is obtained with (nested) activity diagrams in SysML. This resulting functional architecture of the EMA is given in Figure 3.5.

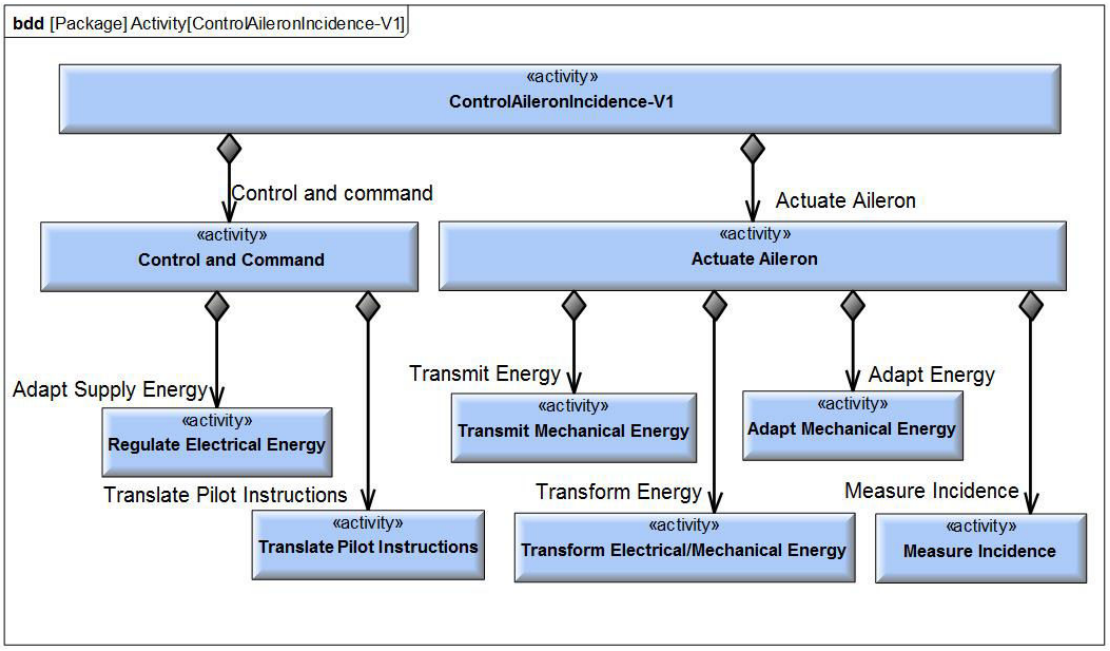


FIGURE 3.5: Functional Architecture of the EMA

Once the functional architecture is determined, the next step according to our methodology is the functional safety analysis using the functional FMEA technique. In order to perform this step, the SysML system model is exported into XMI format and then automatically explored to extract the list of system functions and automatically generate the partially filled preliminary FMEA. For the EMA, the preliminary FMEA automatically generated based on the functional architecture of Figure 3.5 is given in Figure 3.16.

Based on this preliminary FMEA and on a good understanding of the system functioning, the safety expert or the FMEA team completes the analysis to generate the final FMEA. The completed functional FMEA of the EMA is given in Figure 3.17.

As seen in this FMEA, several functions are critical because their failure could have catastrophic effects. Corrective measures are then required to reduce the risk. This can be obtained by allocating components with very high reliability to achieve these functions but also by performing some changes at the functional level by modifying or removing the critical functions if they are not essential to the system functioning or adding fault tolerance mechanisms. In this case, we decided to perform modifications at the functional level by adding an new function of "Internal Diagnosis" to the system. This function collects measures of some critical parameters of the different other functions and aims at identifying potential functional degradation in the system that could be caused by a failure and could lead to other failures. In case of abnormal behavior, this function will inform the "Control and Command" function that will inform the pilot and adjust the outputs it provides accordingly if needed.

Again, it is reminded that this example is chosen to illustrate the methodology but we were unable to obtain any industrial information about it because of confidentiality. As we don't have a wide knowledge of the real system, and we have no return of experience, the analyses provided here express our point of view and may be criticized for lack of accuracy. However, they allowed us to test our approach.

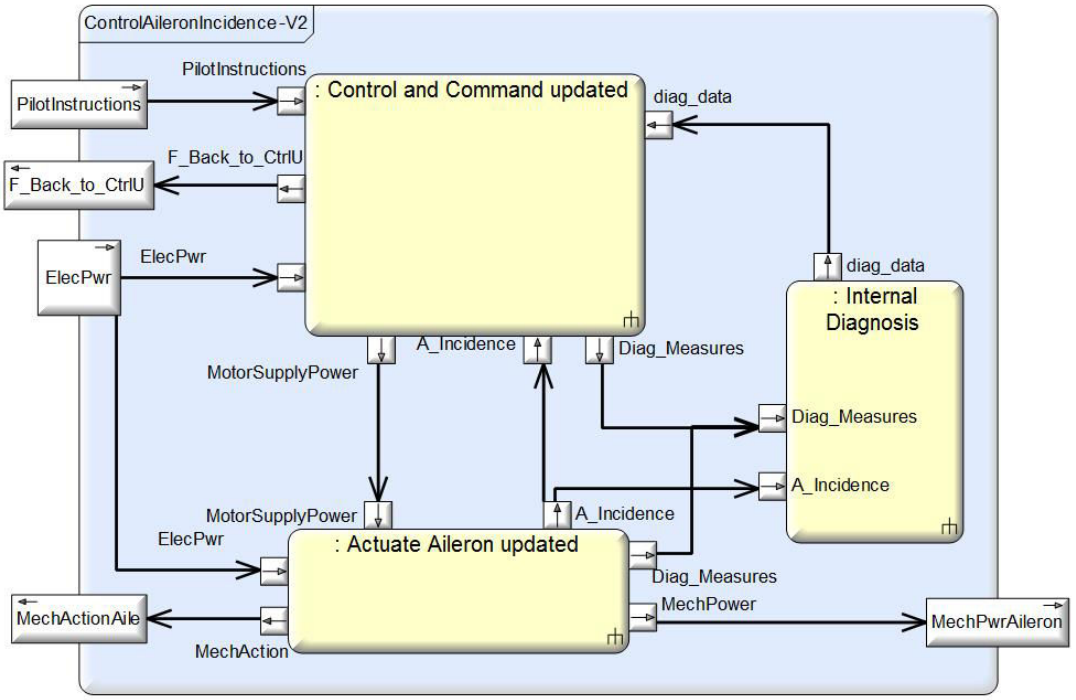


FIGURE 3.6: Updated Functional Decomposition

The updated activity diagram of the top-level function "Control Aileron Incidence" is given in Figure 3.6.

The new added function implies modification of the other functions since they have to provide this function with monitoring data. The updated activity diagrams of the "Control and Command Updated" and "Actuate Aileron Updated" functions are given in Figures 3.7 and 3.8 respectively.

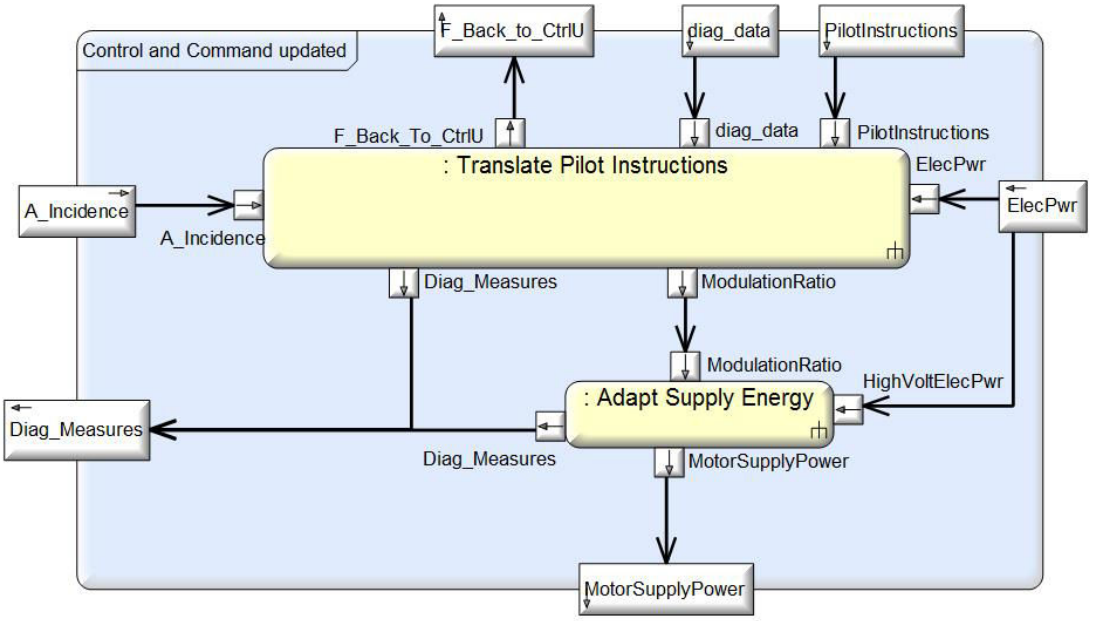


FIGURE 3.7: Updated Functional Decomposition of Control Command

The resulting functional hierarchy after this design iteration is given in Figure 3.9.

As the functional architecture of the system has been modified, a new iteration should be done in the functional FMEA to integrate the new function, analyze the impact of its failure modes and also assess that this new function does not impact the already established safety level.

When no more change needs to be done at the functional level, it's time to move to the next step of the methodology : the *Logical Architecture Definition*. The logical architecture is obtained in two steps. The first one is the identification of the components and allocating them to the functions. The next step then consists in identifying the communication among components. Once logical architecture is defined, the next task is to analyze it according to safety aspects. This leads to the component FMEA generation that is the subject of the next section.

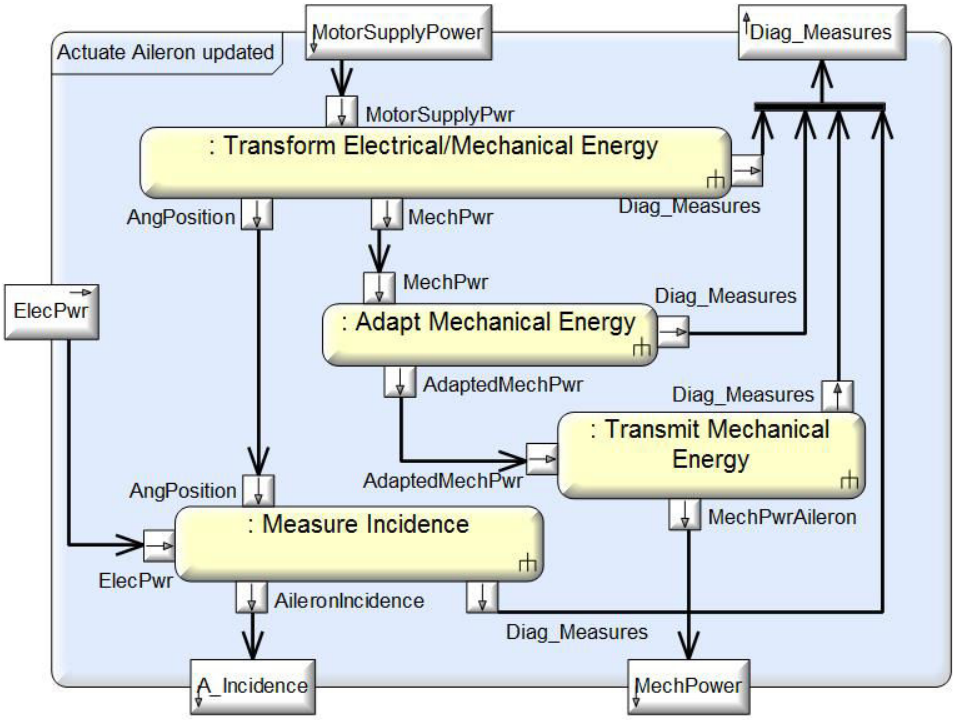


FIGURE 3.8: Updated Functional Decomposition of Actuate Aileron

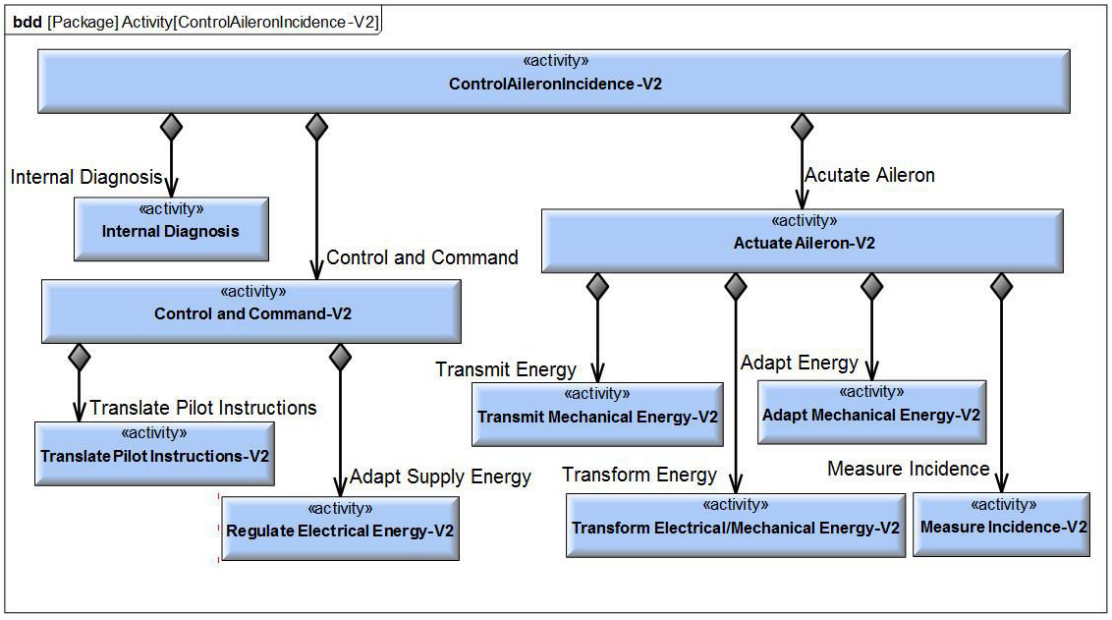


FIGURE 3.9: Updated Functional Hierarchy

3.5 Automated Component FMEA Generation in SafeSysE

3.5.1 Component FMEA Generation : Implementation

Component FMEA aims at identifying potential dysfunctional behavior of each system component and evaluating its impact on the system global behavior. The first step when performing a component FMEA is to delimit the boundary of the study, i.e. the desired level of detail. A component FMEA can only be performed when the system structure is known, i.e. components are identified and allocated to functions.

The elaboration of a component FMEA is quite similar to the functional FMEA since it is based on the same steps. The main difference is that component FMEA focuses on the components while functional FMEA focuses on the system functions.

The component FMEA generation is based on the structural models of the system. In our case, it is generated from SysML BDD and IBD. The first diagram provides the list of system components while the second one provides the interactions among components. Knowing the way in which the components interact and the different flows exchanged among them is very useful for the safety analysis. Indeed, they describe the way in which the flows propagate among the system. When errors occur, they also propagate in the same way. As a result, the interactions among components help in identifying potential causes and effects of failures.

For the automated generation, a new XMI file is generated from the SysML model. This XMI file contains the latest version of the system including the list of functions, the list of components and the allocation links between them. It also contains the information about the input and output flows of each component and the connections among the ports that give the communication paths among components. The XMI file also contains the results of the safety analysis performed at the functional level, i.e. the list of functional failure modes of each function. As the developed safety profile allows to include some information about safety, mainly component failure modes as well as their causes and/or effects, this information is also available in the XMI file. The generated XMI file is then automatically explored by SafeSysE Tool to identify the list of components as well as all the relevant information relative to each component that is useful for the generation of

the component FMEA. We have implemented a function that, giving the id of the activity, returns the classes (or more specifically the blocks) that implement the activity via the "allocated to" relationship in the SysML model. This information is useful for the link between the functional and the component FMEA.

3.5.2 Component FMEA Generation: Case Study

3.5.2.1 Traditional Modeling

In this section, the component FMEA generation is illustrated with the EMA example. The component FMEA generation is based on the structural models of the system. In our case, it is generated based on the SysML structural diagrams, i.e. BDD and IBD. Prior to the FMEA generation, we shall first build the structural model for the case study. The logical structure of the EMA is obtained by the allocation of components to the functions identified in the functional architecture definition. The final functional breakdown is obtained as a result of iterations between the design modifications and the safety analysis until no more modifications are possible to reduce the identified risks or that the modifications are not considered reasonable in terms of the balance between the risk reduction and the investment needed. The final functional hierarchy is given in Figure 3.9.

Three components are allocated to these functions as shown in the BDD representing the system structure given in Figure 3.10. As this diagram is performed after safety analysis, it is different from the one presented in Appendix A. The main difference is that the *Internal Diagnosis* function added as a result of the safety analysis is allocated to the *Embedded MCU with Power Bridge* component. In addition, we can notice that the inputs and outputs of the functions are modified to include the flow exchanges with the added function. The interactions among the components are detailed in the IBD given in Figure 3.11.

A preliminary FMEA is automatically generated from the IBD an extract of which is given in Figure 3.12 containing the list of components. To ensure the consistency of the component FMEA with respect to the functional FMEA, the functions allocated to each component and their failure modes are also added in the preliminary component FMEA. The safety expert then associates the functional failure modes into the corresponding component failure to obtain the final FMEA.

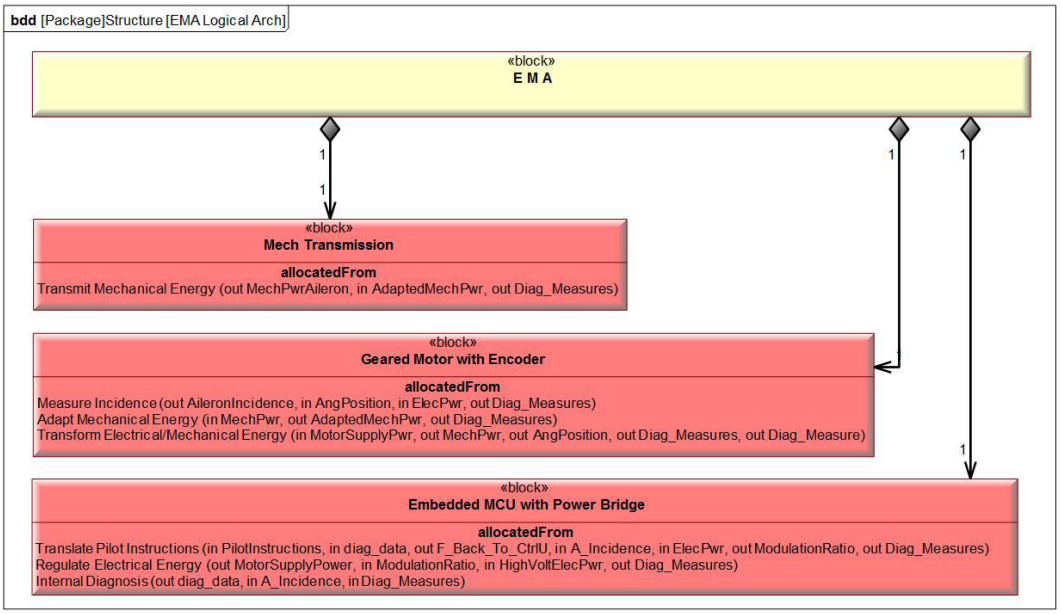


FIGURE 3.10: EMA Logical Structure with Functional Allocation

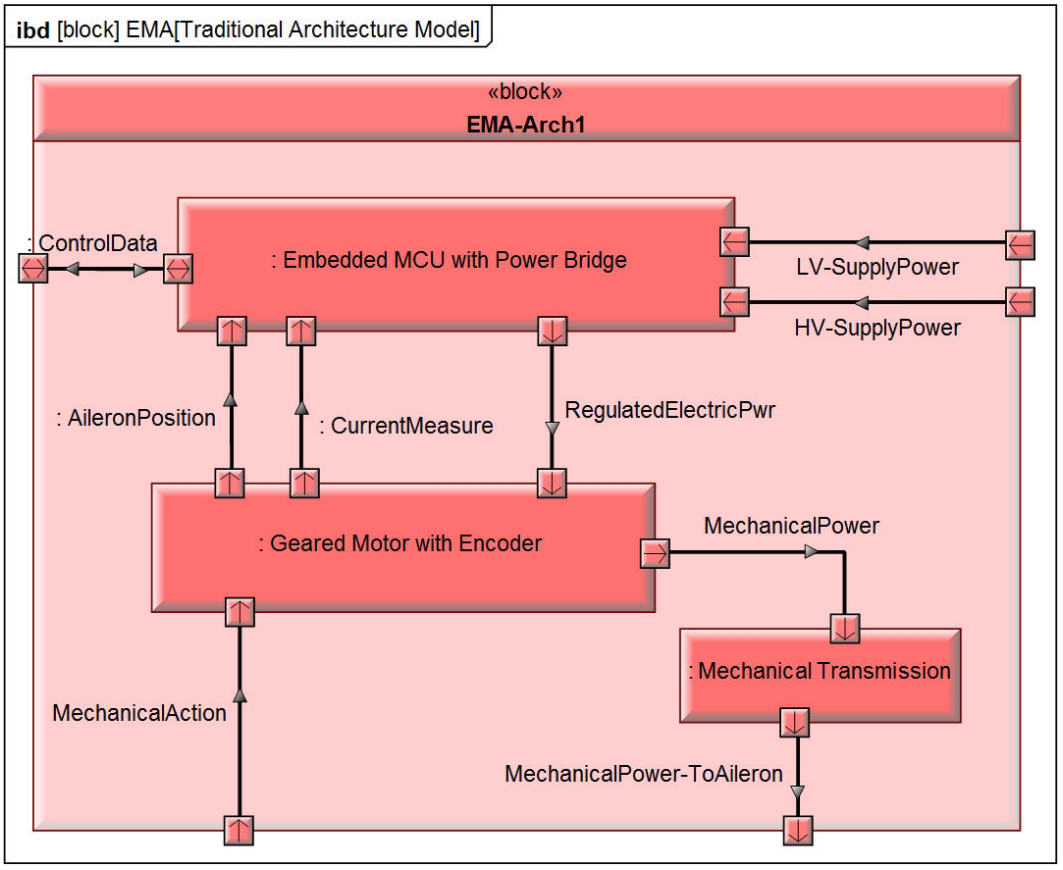


FIGURE 3.11: EMA Internal Architecture

Component	Function	...	Failure mode	Causal factors	Immediate effect	...	System effect
Embedded MCU with Power Bridge	Regulate Electrical Energy	...			in: Geared Motor with Encoder in: External out: Geared Motor with Encoder	...	
	Translate Pilot Instructions	...			in: Geared Motor with Encoder in: External out: Geared Motor with Encoder	...	
Geared Motor with Encoder	Adapt Mechanical Energy	...			in: External in: Embedded MCU with Power Bridge out: Mech Transmission out: Embedded MCU with Power Bridge	...	
	Measure Incidence	...			in: External in: Embedded MCU with Power Bridge out: Mech Transmission out: Embedded MCU with Power Bridge	...	
	Transform Electrical - Mechanical Energy	...			in: External in: Embedded MCU with Power Bridge out: Mech Transmission out: Embedded MCU with Power Bridge	...	

FIGURE 3.12: Extract of Preliminary Component FMEA

3.5.2.2 Mechatronic Extended Modeling

As mentioned in section 2.4, connection components as well as multi-physical flows must be taken into account in order to have a complete safety analysis for mechatronic systems. The Mechatronic Extended Modeling Profile allows to model both aspects.

The resulting Internal Block Diagram of the EMA with the mechatronic extended modeling including connection components is given in Figure 3.13. In this case study, several components are added : a **DataBus**, a **FeedbackBus**, a **PowerBus**, a **PowerSupplyHarness**, and three mechanical couplings named respectively **MC Wing Mot: MechCoupling**, **MC Mot Trans: MechCoupling** and **MC Trans Aileron: MechCoupling**. When the connection components are modeled by blocks, the flow ports on each component will represent the interface allowing it to interact with other components by exchanging different kinds of flows. The connectors (i.e. connection paths in SysML) will indicate that the components are linked.

This modeling is closer to the real world systems where connection components are materialized like the other components. However, as we said previously, integrated

components can exist by combining several components into a unique component and getting rid of the connection components consequently. This would lead to new and more complex components achieving more functions with less connections at the system level.

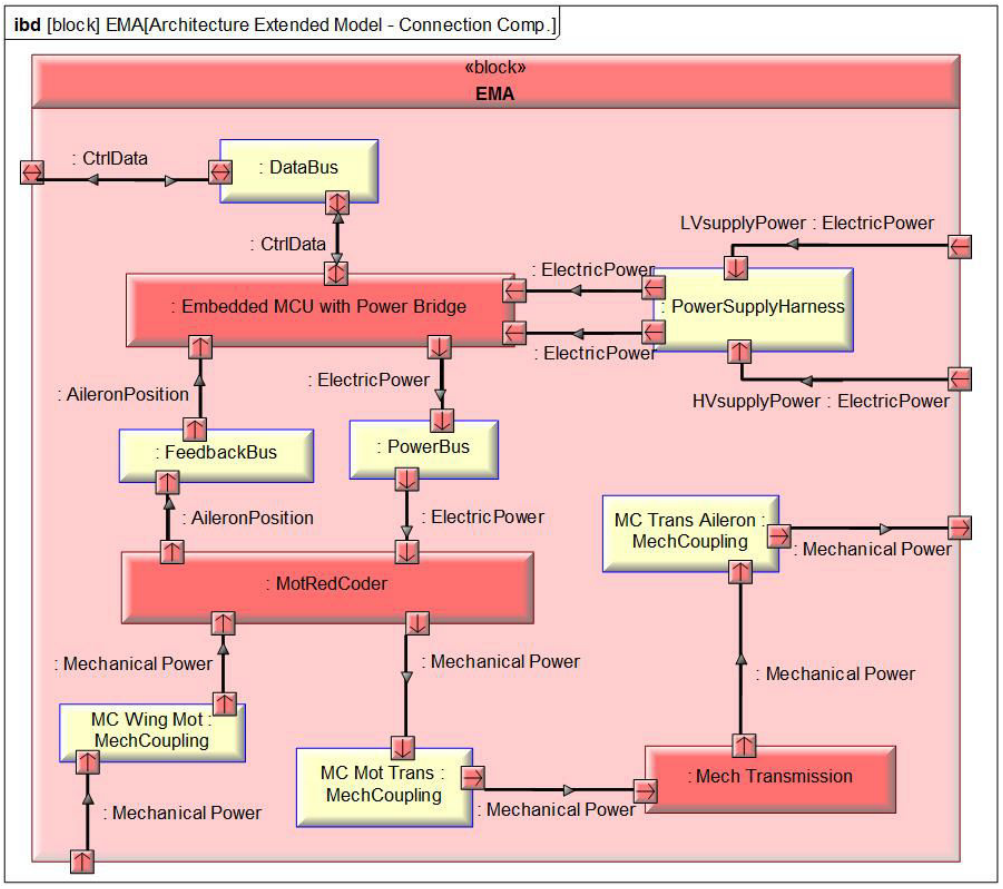


FIGURE 3.13: EMA Architecture Extended with Connection Components

The mechatronic extended model of the EMA including both connection components and multi-physical flows is given in Figure 3.14

An extract of the FMEA automatically generated based on this extended modeling is given in Figure 3.15. In this table, all the components, including connection components are automatically added. For each component, the multi-physical flows that are likely to affect its behavior are automatically added in "Causal Factors". To help the safety expert during the assessment of the immediate effects of a failure, the components directly linked to each component are also automatically added in the FMEA.

The final component FMEA is joined as Appendix at the end of this dissertation.

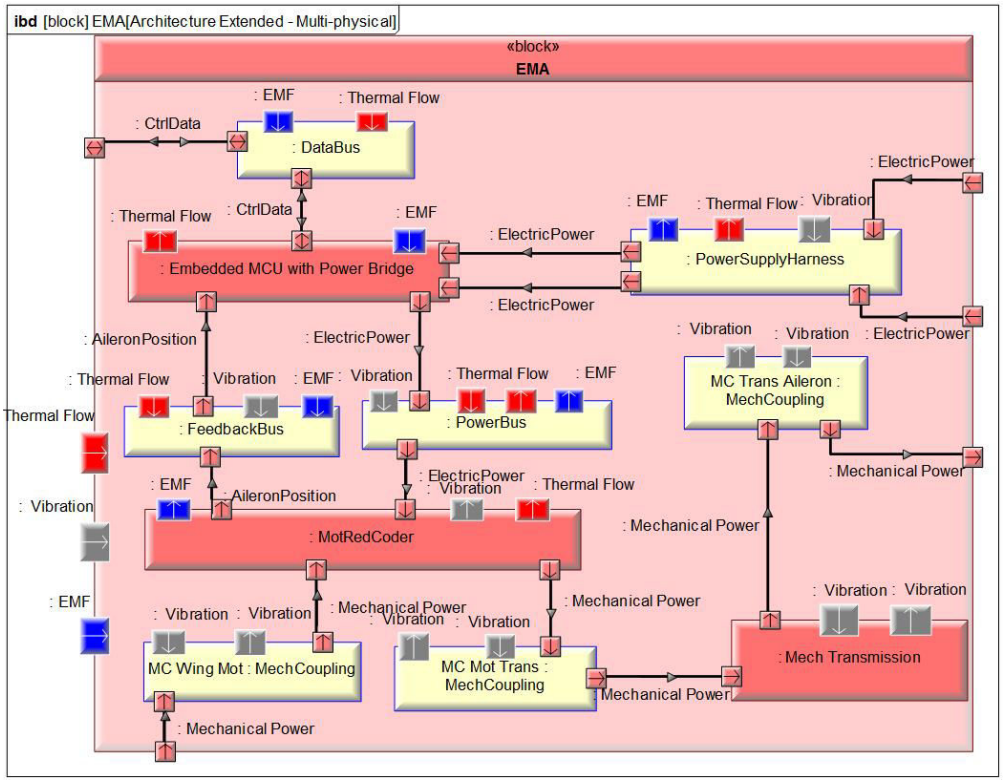


FIGURE 3.14: EMA Architecture Extended with Multi-physical Flows

Component	Failure mode	Causal factors	Immediate Effect	Method of detection	System effect	Recommended Actions	Severity
Embedded MCU with Power Bridge		Thermal Flow EMF	*in*: PowerSupplyHarness *in*: FeedBackBus *inout*: DataBus *out*: PowerBus				
MotRedCoder			*in*: PowerBus *in*: MC Wing Mot *out*: FeedBackBus *out*: MC Mot Trans				
Mech Transmission		Vibration	*in*: MC Mot Trans *out*: MC Trans Aileron				
Data Bus		Thermal Flow, EMF	*inout*: Embedded MCU with Power Bridge *inout*: external				
Feedback Bus		Thermal Flow, Vibration, EMF	*in*: MotRedCoder *out*: Embedded MCU with Power Bridge				
Power Bus		Thermal					

FIGURE 3.15: FMEA Generated with the Mechatronic Extended Modeling

3.6 Conclusion

This chapter dealt with the Failure Mode and Effects Analysis (FMEA). It proposed a method for automating some steps of the generation of both functional and component FMEAs from the SysML system model. The automated steps consist mainly in the extraction of the list of system elements to be analyzed and their relationships to find out all possible failure causes and consequences. The consistency between the latest design modifications and the FMEA information is ensured by the Safety Profile integrated in SysML.

Using the developed safety profile allows to include safety properties in the system model and they will be integrated in the generated FMEA. The safety profile also allows to update the system model with the results of the final FMEA completed by the safety expert. By doing so, the consistency is maintained between safety analyses and the system design process.

The extended modeling using the developed Mechatronic Extended Modeling Profile allows to integrate both the interconnection components and the multi-physical couplings in the system model. This allows their automated integration in the generated FMEA and thus helps a more exhaustive safety analysis.

The automatic generation reduces both errors and duration of the FMEA process and consequently reduces errors and time of the total development process. This leads to a reduction in the development cost and thus increases the competitiveness of the enterprise without loosing in the efficiency of the safety analyses.

Function	Function failure mode	Causal factors	Immediate Effects	System Effects	Recommended actions	Severity
Measure Incidence	Fails to perform	*input*: AngPosition *output*: A_Incidence	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Translate Pilot Instructions			
	Performs incorrectly (degraded performance)	*input*: AngPosition *output*: A_Incidence	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Translate Pilot Instructions			
	Operates inadvertently	*input*: AngPosition *output*: A_Incidence	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Translate Pilot Instructions			
	Operates at incorrect time (early, late)	*input*: AngPosition *output*: A_Incidence	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Translate Pilot Instructions			
	Unable to stop operation	*input*: AngPosition *output*: A_Incidence	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Translate Pilot Instructions			
	Receives erroneous data	*input*: AngPosition *output*: A_Incidence	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Translate Pilot Instructions			
	Sends erroneous data	*input*: AngPosition *output*: A_Incidence	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Translate Pilot Instructions			
Translate Pilot Instructions	Fails to perform	*input*: PilotInstructions, A_Incidence, ElecPwr	*upstream*: Measure Incidence *downstream*: Regulate Electrical Energy			
	Performs incorrectly (degraded	*input*: PilotInstructions, A_Incidence, ElecPwr	*upstream*: Measure Incidence *downstream*: Regulate Electrical Energy			
	Operates inadvertently	*input*: PilotInstructions, A_Incidence, ElecPwr	*upstream*: Measure Incidence *downstream*: Regulate Electrical Energy			
	Operates at incorrect time (early, late)	*input*: PilotInstructions, A_Incidence, ElecPwr	*upstream*: Measure Incidence *downstream*: Regulate Electrical Energy			
	Unable to stop operation	*input*: PilotInstructions, A_Incidence, ElecPwr	*upstream*: Measure Incidence *downstream*: Regulate Electrical Energy			
	Receives erroneous data	*input*: PilotInstructions, A_Incidence, ElecPwr	*upstream*: Measure Incidence *downstream*: Regulate Electrical Energy			
	Sends erroneous data	*input*: PilotInstructions, A_Incidence, ElecPwr	*upstream*: Measure Incidence *downstream*: Regulate Electrical Energy			
Adapt Mechanical Energy	Fails to perform	*input*: MechPwr *output*: AdaptedMechPwr	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Transmit Mechanical Energy			
	Performs incorrectly (degraded performance)	*input*: MechPwr *output*: AdaptedMechPwr	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Transmit Mechanical Energy			
	Operates inadvertently	*input*: MechPwr *output*: AdaptedMechPwr	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Transmit Mechanical Energy			
	Operates at incorrect time (early, late)	*input*: MechPwr *output*: AdaptedMechPwr	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Transmit Mechanical Energy			
	Unable to stop operation	*input*: MechPwr *output*: AdaptedMechPwr	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Transmit Mechanical Energy			
	Receives erroneous data	*input*: MechPwr *output*: AdaptedMechPwr	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Transmit Mechanical Energy			
	Sends erroneous data	*input*: MechPwr *output*: AdaptedMechPwr	*upstream*: Transform Electrical/Mechanical Energy *downstream*: Transmit Mechanical Energy			
Regulate Electrical Energy	Fails to perform	*input*: SupplyPwr *output*: MotorSupplyPower	*upstream*: Translate Pilot Instructions *downstream*: Transform Electrical/Mechanical Energy			
	Performs incorrectly (degraded performance)	*input*: SupplyPwr *output*: MotorSupplyPower	*upstream*: Translate Pilot Instructions *downstream*: Transform Electrical/Mechanical Energy			
	Operates inadvertently	*input*: SupplyPwr *output*: MotorSupplyPower	*upstream*: Translate Pilot Instructions *downstream*: Transform Electrical/Mechanical Energy			
	Operates at incorrect time (early, late)	*input*: SupplyPwr *output*: MotorSupplyPower	*upstream*: Translate Pilot Instructions *downstream*: Transform Electrical/Mechanical Energy			
	Unable to stop operation	*input*: SupplyPwr *output*: MotorSupplyPower	*upstream*: Translate Pilot Instructions *downstream*: Transform Electrical/Mechanical Energy			
	Receives erroneous data	*input*: SupplyPwr *output*: MotorSupplyPower	*upstream*: Translate Pilot Instructions *downstream*: Transform Electrical/Mechanical Energy			
	Sends erroneous data	*input*: SupplyPwr *output*: MotorSupplyPower	*upstream*: Translate Pilot Instructions *downstream*: Transform Electrical/Mechanical Energy			
Transmit Mechanical Energy	Fails to perform	*input*: AdaptedMechPwr *output*: MechPwrAlieron	*upstream*: Adapt Mechanical Energy			
	Performs incorrectly (degraded performance)	*input*: AdaptedMechPwr *output*: MechPwrAlieron	*upstream*: Adapt Mechanical Energy			
	Operates inadvertently	*input*: AdaptedMechPwr *output*: MechPwrAlieron	*upstream*: Adapt Mechanical Energy			
	Operates at incorrect time (early, late)	*input*: AdaptedMechPwr *output*: MechPwrAlieron	*upstream*: Adapt Mechanical Energy			
	Unable to stop operation	*input*: AdaptedMechPwr *output*: MechPwrAlieron	*upstream*: Adapt Mechanical Energy			
	Receives erroneous data	*input*: AdaptedMechPwr *output*: MechPwrAlieron	*upstream*: Adapt Mechanical Energy			
	Sends erroneous data	*input*: AdaptedMechPwr *output*: MechPwrAlieron	*upstream*: Adapt Mechanical Energy			
Transform Electrical/Mechanical Energy	Fails to perform	*input*: ElecSupplyPwr *output*: AngPosition, MechPwr	*upstream*: Regulate Electrical Energy *downstream*: Adapt Mechanical Energy, Measure Incidence			
	Performs incorrectly (degraded	*input*: ElecSupplyPwr *output*: AngPosition, MechPwr	*upstream*: Regulate Electrical Energy *downstream*: Adapt Mechanical Energy, Measure Incidence			
	Operates inadvertently	*input*: ElecSupplyPwr *output*: AngPosition, MechPwr	*upstream*: Regulate Electrical Energy *downstream*: Adapt Mechanical Energy, Measure Incidence			
	Operates at incorrect time (early, late)	*input*: ElecSupplyPwr *output*: AngPosition, MechPwr	*upstream*: Regulate Electrical Energy *downstream*: Adapt Mechanical Energy, Measure Incidence			
	Unable to stop operation	*input*: ElecSupplyPwr *output*: AngPosition, MechPwr	*upstream*: Regulate Electrical Energy *downstream*: Adapt Mechanical Energy, Measure Incidence			
	Receives erroneous data	*input*: ElecSupplyPwr *output*: AngPosition, MechPwr	*upstream*: Regulate Electrical Energy *downstream*: Adapt Mechanical Energy, Measure Incidence			
	Sends erroneous data	*input*: ElecSupplyPwr *output*: AngPosition, MechPwr	*upstream*: Regulate Electrical Energy *downstream*: Adapt Mechanical Energy, Measure Incidence			

FIGURE 3.16: Automatically Generated Functional FMEA

Function	Function failure mode	*Causal factors (predefined)*	Causal factors (Expert)	*Immediate Effects (predefined)*	Immediate Effects (Expert)	System Effects	Recommended actions	Severity
Measure Incidence	Fails to perform	*input:* AngPosition, ElecPwr *output:* A_Incidence	Internal faillure; faulty power supply; faulty measurement	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Translate Pilot Instructions	Faulty measurement provided to "Translate Pilot Instructions"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Performs incorrectly (degraded performance)	*input:* AngPosition, ElecPwr *output:* A_Incidence	Internal faillure; faulty power supply; faulty measurement	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Translate Pilot Instructions	Faulty measurement provided to "Translate Pilot Instructions"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Operates inadvertently	*input:* AngPosition, ElecPwr *output:* A_Incidence	Internal faillure; faulty power supply; faulty measurement	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Translate Pilot Instructions	Faulty measurement provided to "Translate Pilot Instructions"	Random inappropriate response of aileron	Appropriate reliabilty; preventive diagnostics	catastrophic
	Operates at incorrect time (early, late)	*input:* AngPosition, ElecPwr *output:* A_Incidence	Internal faillure; faulty power supply; faulty measurement	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Translate Pilot Instructions	Faulty measurement provided to "Translate Pilot Instructions"	Response delay of the aileron	Appropriate reliabilty; preventive diagnostics	critical
	Unable to stop operation	*input:* AngPosition, ElecPwr *output:* A_Incidence	Non-relevant	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Translate Pilot Instructions	Faulty measurement provided to "Translate Pilot Instructions"	Non-relevant	*	*
	Receives erroneous data	*input:* AngPosition, ElecPwr *output:* A_Incidence	Faulty measurement	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Translate Pilot Instructions	Faulty measurement provided to "Translate Pilot Instructions"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Sends erroneous data	*input:* AngPosition, ElecPwr *output:* A_Incidence	Internal faillure; faulty power supply	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Translate Pilot Instructions	Faulty measurement provided to "Translate Pilot Instructions"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
Translate Pilot Instructions	Fails to perform	*input:* PilotInstructions, A_Incidence, ElecPwr *output:* ModulationRatio, F_Back_to_CtrlU	Internal faillure; faulty power supply; inappropriate input data	*upstream:* Measure Incidence *downstream:* Regulate Electrical Energy	Inappropriate data provided to "Regulate Electrical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Performs incorrectly (degraded performance)	*input:* PilotInstructions, A_Incidence, ElecPwr *output:* ModulationRatio, F_Back_to_CtrlU	Internal faillure; faulty power supply; inappropriate input data	*upstream:* Measure Incidence *downstream:* Regulate Electrical Energy	Inappropriate data provided to "Regulate Electrical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Operates inadvertently	*input:* PilotInstructions, A_Incidence, ElecPwr *output:* ModulationRatio, F_Back_to_CtrlU	Internal faillure; faulty power supply; inappropriate input data	*upstream:* Measure Incidence *downstream:* Regulate Electrical Energy	Inappropriate data provided to "Regulate Electrical Energy"	Random inappropriate response of aileron	Appropriate reliabilty; preventive diagnostics	catastrophic
	Operates at incorrect time (early, late)	*input:* PilotInstructions, A_Incidence, ElecPwr *output:* ModulationRatio, F_Back_to_CtrlU	Internal faillure; faulty power supply; inappropriate input data	*upstream:* Measure Incidence *downstream:* Regulate Electrical Energy	Inappropriate data provided to "Regulate Electrical Energy"	Response delay of the aileron	Appropriate reliabilty; preventive diagnostics	critical
	Unable to stop operation	*input:* PilotInstructions, A_Incidence, ElecPwr *output:* ModulationRatio, F_Back_to_CtrlU	Internal failure	*upstream:* Measure Incidence *downstream:* Regulate Electrical Energy	Inappropriate data provided to "Regulate Electrical Energy"	Unable to stop operation	Appropriate reliabilty; preventive diagnostics	marginal
	Receives erroneous data	*input:* PilotInstructions, A_Incidence, ElecPwr *output:* ModulationRatio, F_Back_to_CtrlU	inappropriate incidence data; inappropriate instructions	*upstream:* Measure Incidence *downstream:* Regulate Electrical Energy	Inappropriate data provided to "Regulate Electrical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Sends erroneous data	*input:* PilotInstructions, A_Incidence, ElecPwr *output:* ModulationRatio, F_Back_to_CtrlU	Internal faillure; faulty power supply; inappropriate input data	*upstream:* Measure Incidence *downstream:* Regulate Electrical Energy	Inappropriate data provided to "Regulate Electrical Energy"	Aileron out of control; inappropriate report to pilot	Appropriate reliabilty; preventive diagnostics	catastrophic

Function	Function failure mode	*Causal factors (predefined)*	Causal factors (Expert)	*Immediate Effects (predefined)*	Immediate Effects (Expert)	System Effects	Recommended actions	Severity
Adapt Mechanical Energy	Fails to perform	*input:* MechPwr *output:* AdaptedMechPwr	Internal failure; faulty mechanical couplings	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Transmit Mechanical Energy	Inappropriate mechanical energy provided to "Transmit Mechanical Energy"; Potential source of faulty behavior of "Transform Electrical/Mechanical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Performs incorrectly (degraded performance)	*input:* MechPwr *output:* AdaptedMechPwr	Internal failure; faulty mechanical couplings	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Transmit Mechanical Energy	Inappropriate mechanical energy provided to "Transmit Mechanical Energy"; Potential source of faulty behavior of "Transform Electrical/Mechanical Energy"	degraded motion of aileron (speed, acceleration, angular amplitude)	Appropriate reliabilty; preventive diagnostics	critical
	Operates inadvertently	*input:* MechPwr *output:* AdaptedMechPwr	Non-relevant	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Transmit Mechanical Energy	*	Non-relevant	*	*
	Operates at incorrect time (early, late)	*input:* MechPwr *output:* AdaptedMechPwr	Non-relevant	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Transmit Mechanical Energy	*	Non-relevant	*	*
	Unable to stop operation	*input:* MechPwr *output:* AdaptedMechPwr	Non-relevant	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Transmit Mechanical Energy	*	Non-relevant	*	*
	Receives erroneous data	*input:* MechPwr *output:* AdaptedMechPwr	faulty mechanical couplings	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Transmit Mechanical Energy	Inappropriate mechanical energy provided to "Transmit Mechanical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Sends erroneous data	*input:* MechPwr *output:* AdaptedMechPwr	faulty mechanical couplings	*upstream:* Transform Electrical/Mechanical Energy *downstream:* Transmit Mechanical Energy	Inappropriate mechanical energy provided to "Transmit Mechanical Energy"; Potential source of faulty behavior of "Transform Electrical/Mechanical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
Regulate Electrical Energy	Fails to perform	*input:* ModulationRatio, ElecPwr *output:* MotorSupplyPower	Faulty power supply; inappropriate modulation input; internal failure	*upstream:* Translate Pilot Instructions *downstream:* Transform Electrical/Mechanical Energy	Inappropriate mechanical energy provided to "Transform Electrical/Mechanical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Performs incorrectly (degraded performance)	*input:* ModulationRatio, ElecPwr *output:* MotorSupplyPower	Faulty power supply; inappropriate modulation input; internal failure	*upstream:* Translate Pilot Instructions *downstream:* Transform Electrical/Mechanical Energy	Inappropriate mechanical energy provided to "Transform Electrical/Mechanical Energy"	degraded motion of aileron (speed, acceleration, angular	Appropriate reliabilty; preventive diagnostics	critical
	Operates inadvertently	*input:* ModulationRatio, ElecPwr *output:* MotorSupplyPower	Faulty power supply; inappropriate modulation input; internal failure	*upstream:* Translate Pilot Instructions *downstream:* Transform Electrical/Mechanical Energy	Inappropriate mechanical energy provided to "Transform Electrical/Mechanical Energy"	Random inappropriate response of aileron	Appropriate reliabilty; preventive diagnostics	catastrophic
	Operates at incorrect time (early, late)	*input:* ModulationRatio, ElecPwr *output:* MotorSupplyPower	Faulty power supply; inappropriate modulation input; internal failure	*upstream:* Translate Pilot Instructions *downstream:* Transform Electrical/Mechanical Energy	Inappropriate mechanical energy provided to "Transform Electrical/Mechanical Energy"	Response delay of the aileron	Appropriate reliabilty; preventive diagnostics	critical
	Unable to stop operation	*input:* ModulationRatio, ElecPwr *output:* MotorSupplyPower	internal faiilure	*upstream:* Translate Pilot Instructions *downstream:* Transform Electrical/Mechanical Energy	Inappropriate mechanical energy provided to "Transform Electrical/Mechanical Energy"	Unable to stop operation	Appropriate reliabilty; preventive diagnostics	marginal
	Receives erroneous data	*input:* ModulationRatio, ElecPwr *output:* MotorSupplyPower	Inappropriate modulation input	*upstream:* Translate Pilot Instructions *downstream:* Transform Electrical/Mechanical Energy	Inappropriate mechanical energy provided to "Transform Electrical/Mechanical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Sends erroneous data	*input:* ModulationRatio, ElecPwr *output:* MotorSupplyPower	Faulty power supply; inappropriate modulation input; internal failure	*upstream:* Translate Pilot Instructions *downstream:* Transform Electrical/Mechanical Energy	Inappropriate mechanical energy provided to "Transform Electrical/Mechanical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic

Function	Function failure mode	*Causal factors (predefined)*	Causal factors (Expert)	*Immediate Effects (predefined)*	Immediate Effects (Expert)	System Effects	Recommended actions	Severity
Transmit Mechanical Energy	Fails to perform	*input:* AdaptedMechPwr *output:* MechPwrAileron	Internal failure; faulty mechanical couplings	*upstream:* Adapt Mechanical Energy	Inappropriate mechanical energy provided to aileron; Potential source of faulty behavior of "Adapt Mechanical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Performs incorrectly (degraded performance)	*input:* AdaptedMechPwr *output:* MechPwrAileron	Internal failure; faulty mechanical couplings	*upstream:* Adapt Mechanical Energy	Inappropriate mechanical energy provided to aileron; Potential source of faulty behavior of "Adapt Mechanical Energy"	degraded motion of aileron	Appropriate reliabilty; preventive diagnostics	critical
	Operates inadvertently	*input:* AdaptedMechPwr *output:* MechPwrAileron	Non-relevant	*upstream:* Adapt Mechanical Energy	*	Non-relevant	*	*
	Operates at incorrect time (early, late)	*input:* AdaptedMechPwr *output:* MechPwrAileron	Non-relevant	*upstream:* Adapt Mechanical Energy	*	Non-relevant	*	*
	Unable to stop operation	*input:* AdaptedMechPwr *output:* MechPwrAileron	Non-relevant	*upstream:* Adapt Mechanical Energy	*	Non-relevant	*	*
	Receives erroneous data	*input:* AdaptedMechPwr *output:* MechPwrAileron	faulty mechanical couplings	*upstream:* Adapt Mechanical Energy	Inappropriate mechanical energy provided to aileron	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Sends erroneous data	*input:* AdaptedMechPwr *output:* MechPwrAileron	faulty mechanical couplings	*upstream:* Adapt Mechanical Energy	Inappropriate mechanical energy provided to aileron; Potential source of faulty behavior of "Adapt Mechanical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
Transform Electrical/Mechanical Energy	Fails to perform	*input:* MotorSupplyPower *output:* AngPosition, MechPwr	Faulty power supply, internal failure	*upstream:* Regulate Electrical Energy *downstream:* Adapt Mechanical Energy, Measure Incidence	Inappropriate mechanical energy provided to "Adapt Mechanical Energy"; Inappropriate angular position provided to "Measure Incidence"; Potential source of faulty behavior of "Regulate Electrical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Performs incorrectly (degraded performance)	*input:* MotorSupplyPower *output:* AngPosition, MechPwr	Faulty power supply, internal failure	*upstream:* Regulate Electrical Energy *downstream:* Adapt Mechanical Energy, Measure Incidence	Inappropriate mechanical energy provided to "Adapt Mechanical Energy"; Inappropriate angular position provided to "Measure Incidence"; Potential source of faulty behavior of "Regulate Electrical Energy"	degraded motion of aileron (speed, acceleration, angular amplitude)	Appropriate reliabilty; preventive diagnostics	critical
	Operates inadvertently	*input:* MotorSupplyPower *output:* AngPosition, MechPwr	Faulty power supply, internal failure	*upstream:* Regulate Electrical Energy *downstream:* Adapt Mechanical Energy, Measure Incidence	Inappropriate mechanical energy provided to "Adapt Mechanical Energy"; Inappropriate angular position provided to "Measure Incidence"; Potential source of faulty behavior of "Regulate Electrical Energy"	Random inappropriate response of aileron	Appropriate reliabilty; preventive diagnostics	catastrophic
	Operates at incorrect time (early, late)	*input:* MotorSupplyPower *output:* AngPosition, MechPwr	Faulty power supply, internal failure	*upstream:* Regulate Electrical Energy *downstream:* Adapt Mechanical Energy, Measure Incidence	Inappropriate mechanical energy provided to "Adapt Mechanical Energy"; Inappropriate angular position provided to "Measure Incidence"; Potential source of faulty behavior of "Regulate Electrical Energy"	Response delay of the aileron	Appropriate reliabilty; preventive diagnostics	critical
	Unable to stop operation	*input:* MotorSupplyPower *output:* AngPosition, MechPwr	Faulty power supply	*upstream:* Regulate Electrical Energy *downstream:* Adapt Mechanical Energy, Measure Incidence	Inappropriate mechanical energy provided to "Adapt Mechanical Energy"; Inappropriate angular position provided to "Measure Incidence"; Potential source of faulty behavior of "Regulate Electrical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Receives erroneous data	*input:* MotorSupplyPower *output:* AngPosition, MechPwr	Faulty power supply	*upstream:* Regulate Electrical Energy *downstream:* Adapt Mechanical Energy, Measure Incidence	Inappropriate mechanical energy provided to "Adapt Mechanical Energy"; Inappropriate angular position provided to "Measure Incidence"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic
	Sends erroneous data	*input:* MotorSupplyPower *output:* AngPosition, MechPwr	inappropriate angular position; inappropriate mechanical power	*upstream:* Regulate Electrical Energy *downstream:* Adapt Mechanical Energy, Measure Incidence	Inappropriate mechanical energy provided to "Adapt Mechanical Energy"; Inappropriate angular position provided to "Measure Incidence"; Potential source of faulty behavior of "Regulate Electrical Energy"	Aileron out of control	Appropriate reliabilty; preventive diagnostics	catastrophic

FIGURE 3.17: Functional FMEA Completed by the Safety Expert

Chapter 4

Automatic Fault Tree Generation from SysML Models

4.1 Introduction

Fault Tree Analysis (FTA) is a popular deductive safety analysis technique that, starting from an undesired state (usually a failure state), aims at identifying all the possible paths leading to this state. The paths are constituted of one or more basic faults contributing to the undesired event [23, 85]. The result of the analysis is a tree-like graphical representation where the basic faults (constituting the leafs of the tree) are linked together by logical relationships. FTA allows to model the fault propagation through the system. It is then very useful for the identification of combinations of faults leading to hazardous undesired system states. It can be also used in a quantitative way for reliability computation to ensure that the system meets regulatory safety requirements.

However, for large systems, the manual construction of fault trees is laborious and error-prone [85]. This led to a growing interest for the automated generation of fault trees.

In this chapter we propose a method for the automated fault tree generation that is performed in two steps. The first step provides a generic fault tree based on the system structural model. It consists in an automated exploration of the system model to identify specific patterns, then for each pattern, an algorithm is built to generate the corresponding partial fault tree. The resulting generic

fault tree is then obtained by assembling all partial fault trees. The second step builds a specific fault tree for a special undesired event. This step is based on the generic fault tree as well as the component FMEA already built in previous steps. Fault tree is generated in two formats: an image for a better understanding of fault propagation and an Open-PSA format for further exploration with existing dedicated tools. The methodology is described in more detail in this chapter after a reminder of the FTA basic concepts as well as a review of the related work about the automated generation of fault trees.

This chapter is organized as follows. First the basic theoretical concepts about fault trees are given in section 4.2. Then it gives an overview of related work in section 4.3. Our pattern-based approach for automated generation of fault trees from SysML models is presented in section 4.4 and illustrated with a case study in section 4.5. Finally, the chapter is concluded in section 4.6.

4.2 Fault Tree Analysis (FTA)

In this section, the concepts and uses of Fault Tree Analysis (FTA) are presented. Fault trees are widely used for safety assessment and reliability of systems for over 40 years [86]. FTA is a popular deductive top-down technique for reliability and safety analysis. Starting from an undesired state (usually a failure state of the system), FTA aims at identifying all the possible causes of this failure state by a top-down traversal of the fault tree until reaching the root causes. The causes are usually given as paths constituted of one or more basic events (usually faulty events but can also include normal events) contributing to the undesired event [23, 26, 85]. The analysis output is a graphical tree diagram, called fault tree. A fault tree describes the different paths within a system that lead to a potential undesired loss event (or a failure). The pathways interconnect events and conditions that contribute to the undesired system event or state using standard logical operators or gates. The two most commonly used gates are the AND and OR gates. The AND gate is used when both events need to occur to cause the top event occurrence. If the occurrence of either event results in occurrence of the top event, then the OR gate is used to connect the events.

A fault tree can be translated into a mathematical model to compute failure probabilities. This model expresses the relationship between the probability of the

top level event and the probabilities of the cause events. For each logical gate, a mathematical formula gives the relationship between the probability of the output event and the respective probabilities of the input events of the gate. The top level event probability can then be deduced from this model.

FTA is also used in a qualitative way to identify weak points in the design. This is obtained by identifying the necessary and sufficient combinations of basic events that cause the top level events. These necessary and sufficient combinations are called the *minimal cut sets*.

4.2.1 Fault Tree Event Types

In a fault tree, three kinds of events can be found: the top event, the intermediary events and the primary events. Different kinds of primary events can be used in a fault tree. The definition and symbol of each of these primary events are given hereafter [26]. A specific geometrical shape is given to each kind of event. Two different ways can be found in the literature to note the name of event: either in a rectangle stuck just above the geometric form or directly inside it. Both symbols are given for each event below.

- **Basic event**

  A basic event is a basic initiating fault requiring no further development. The basic event symbol is given in Figure 4.1.

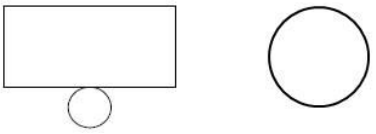


FIGURE 4.1: Basic Event Symbol

- **Conditioning event**

  A conditioning event is a specific condition or restriction that applies to any logic gate (used primarily with PRIORITY AND and INHIBIT gates). The conditioning event symbol is given in Figure 4.2.

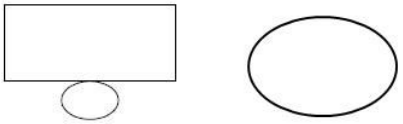


FIGURE 4.2: Conditioning Event Symbol

- **Undeveloped event**

  An undeveloped event is an event which is not further developed either because it is of insufficient consequence or because information is unavailable. The undeveloped event symbol is given in Figure 4.3.

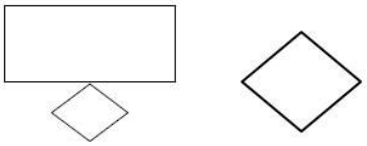


FIGURE 4.3: Undeveloped Event Symbol

- **House event**

  A house event is an event which is normally expected to occur as part of the nominal system operating. The house event symbol is given in Figure 4.4.

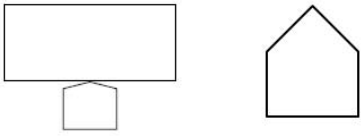
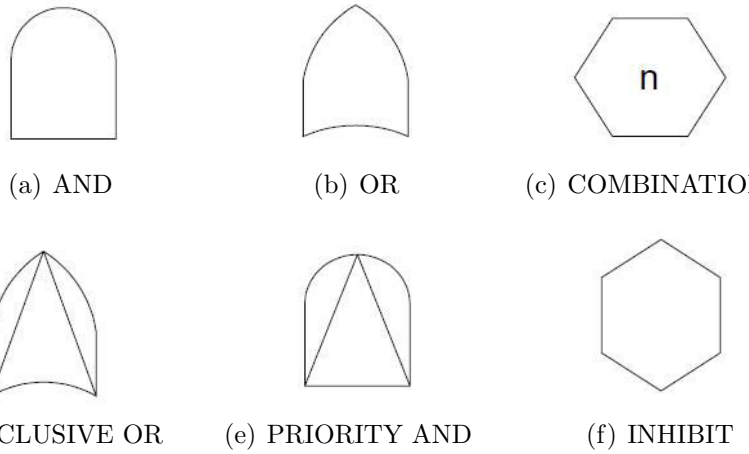


FIGURE 4.4: House Event Symbol

4.2.2 Fault Tree Logical Gates

Different logical gates can be used in a fault tree to link the events (gate inputs) leading to the upper level event (gate output). The gates describe the logical combination of input events that lead to the output event. The main gates are described hereafter by using the definitions given in [26]. The gates symbols are given in Figure 4.5.

- **AND gate**: "Output fault occurs if all of the input faults occur";



(a) AND (b) OR (c) COMBINATION

(d) EXCLUSIVE OR (e) PRIORITY AND (f) INHIBIT

FIGURE 4.5: Logical Gates Symbols

- **OR gate**: "Output fault occurs if a least one of the input faults occurs";
- **COMBINATION**: "Output fault occurs if n of the input faults occur";
- **EXCLUSIVE OR**: "Output fault occurs if exactly one of the input faults occurs".
- **PRIORITY AND**: "Output fault occurs if all of the input faults occur in a specific sequence (the sequence is represented by a CONDITIONING EVENT drawn to the right of the gate)";
- **INHIBIT**: "Output fault occurs if the (single) input fault occurs in the presence of an enabling condition (the enabling condition is represented by a CONDITIONING EVENT drawn to the right of the gate)".

4.3 Automated Generation of Fault Trees: Related Work

Manual construction of fault trees is time consuming and error prone, especially for complex systems. To cope with this complexity, automatic generation of fault tree has been subject of many research works. Several approaches are used for the automatic generation of fault trees. The main difference concerns the type of the starting model based on which the generation is performed. In the following, a brief overview of some recent works is given.

Yakymets et.al in [87] combine the analytic approach with formal verification methods for the automatic generation of fault trees from SysML models. In this approach several steps are needed. First, the system to be analyzed is designed and its structural models are built using the SysML BDD and IBD diagrams. These models are then annotated with failure behavior. Then the entire model is converted into AltaRica language [38, 42, 43]. An algorithm already existing in the ARC tool analyzes the AltaRica model and derives the different minimal cut-sets from the model. These cut-sets are assembled to form the final FT. The resulting FT can be represented either with Open-PSA or a SysML dedicated profile.

Tajarrod and Latif-Shabgahi in [86], describe fault trees construction from MATLAB Simulink models. In this work, the nominal model is built in Simulink and then is manually extended with failure behavioral information of the system. Based on this extended model and the classification of components, fault tree for a specific top event is automatically constructed.

An automatic generation of fault trees from AADL (Architecture Analysis and Design Language) models is proposed in [88]. In this work, the system architectural model is built with the AADL language and then is annotated with fault and failure information using the Error Annex, a sub-language of AADL. Based on the annotated model, fault trees are automatically generated in a commercial tool: CAFTA.

The Formal Safety Analysis Platform FSAP toghether with the NuSMV-SA (the NuSMV model checker for safety analysis) engine presented in [39, 89] provide a uniform platform for modeling both nominal and degraded behavior of complex system with the objective to formally verify this behavior. In addition to the formal verification, the tool can be also used as a powerful fault tree analysis tool. However, one limitation of this tool is that the fault trees automatically generated with minimal cut sets have a flat structure with only two levels deep. This representation does not reflect the structure of the system and consequently, exploring the fault tree to understand the fault propagation through the system components is not very intuitive for systems engineers.

However, as far as we know, there is no related work concerning the fault tree generation from SysML models as in our study.

4.4 Automated Fault Tree Generation from SysML IBD

In this section, we will describe our method to generate fault trees automatically from structural diagrams, i.e SysML Internal Block Diagrams (IBD). The IBD gives the internal structure of the system and the interactions among components. The interfaces through which the components interact are represented via standard and flow ports and the interactions are represented via paths between the corresponding ports called "connectors". If a failure occurs in one component, it will be propagated throughout the system via these paths. The idea of this work is to automatically generate fault trees based on the system IBD, using two concepts: *directed graph traversal* and *block design patterns*. Each concept is detailed hereafter.

4.4.1 Directed Graph Traversal

An IBD can be represented as a directed graph $G = (V, E)$ where V is the set of vertices and E is the set of directed edges. The set of vertices is composed of system components and external interfaces respectively represented by the parts of an IBD and ports that are situated on the border of the system. The external interfaces can be either input ports through which the system receives flows from its environment (users or contributing systems) or output ports through which the system provides required output flows to its environment. The internal ports through which the components interact do not need to be represented since they can be abstracted by edges directly connecting parts. It is also noted that the graph G accepts multi-edges between two parts that symbolize different kinds of items flowing between these two parts. So, to build a fault tree for a given undesired top event, a graph traversal algorithm can be used to find out components relating to each other by using the directed edges. This algorithm follows the principle of backtracking from the hazard to the leaf events. The traversal starts at an external output port, traces back to nodes that are his predecessors and continues to visit the other nodes. Since a node can have several predecessors, a branch is finished when we reach an external input port, or when we arrive back to a node that has already been visited.

4.4.2 Block Design Pattern

To facilitate the fault tree generation, we also use the "divide and conquer" principle by partitioning the IBD and treating each partition separately. Indeed, during the graph traversal, the algorithm also identifies some interesting patterns in an IBD. Each pattern gives rise to a sub-fault tree and the whole fault tree will be assembled automatically by using the mentioned graph traversal algorithm. The fault tree generated in this way is a generic one transcribing the system topology, i.e. the different paths within a system through which faults can propagate to reach the mentioned output port. If the system has several outputs, then a generic fault tree is built for each output port to describe all the paths that could lead to an error on this output.

In this work we have identified different patterns each of which has a specific role in the system. These patterns are *Entry*, *Exit*, and *Feedback*. Another kind of pattern, named *Redundant* pattern, related to safety design criteria where a block part can have input ports coming from components assuring redundancy for higher reliability is also studied.

The following subsections describe the recognized patterns as well as their generated partial fault trees. All these patterns are grouped into an illustrating IBD in Figure 4.6. Each pattern is surrounded with a dashed rectangle annotated with the corresponding name in an attached note.

4.4.2.1 Entry Pattern

An "Entry pattern" is composed of an entry part and its ports. An entry part in an IBD is a block part that has at least one input port receiving item flow from outside the actual system/subsystem (block B1 in Figure 4.6 is an entry part). In the generated sub-fault tree (Figure 4.7), this special input port will be transformed into a basic event representing a failure or error of a system component that is outside the actual block. We will have an OR logic gate whose operands are: the internal failure of the part and the basic events representing the external failures (from input ports on the boundary) and failure of all eventual input ports of the part coming from other components.

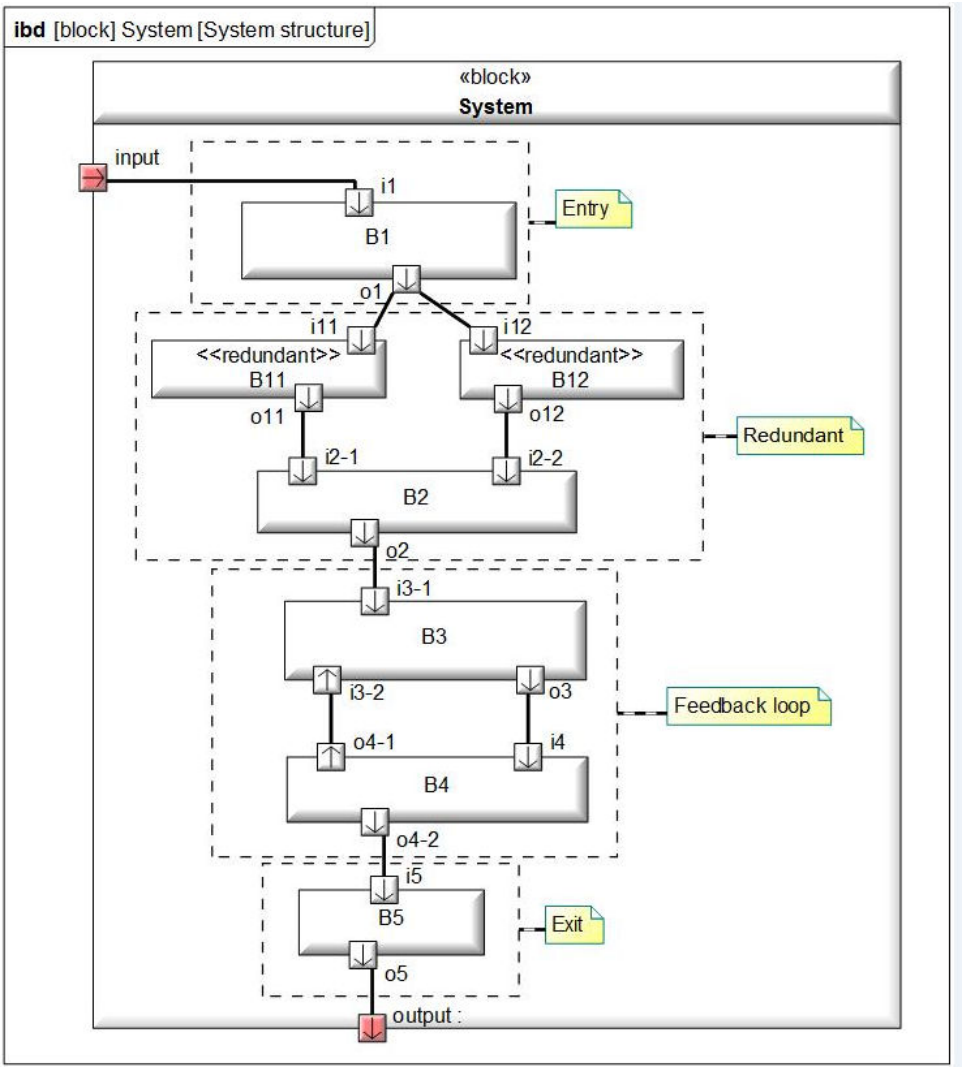


FIGURE 4.6: IBD Block Design Patterns

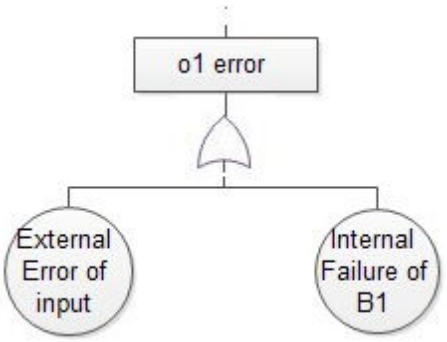


FIGURE 4.7: Fault Tree for Entry Pattern in Figure 4.6

4.4.2.2 Exit Pattern

An "Exit pattern" is composed of an exit part and its ports. An exit part in an IBD is a block part that has at least one output port sending item flow out of the actual system/subsystem (bloc B5 in Figure 4.6). In the corresponding fault tree (Figure 4.8), this special output port gives rise to a top event undesired state of the actual system modeled by the IBD. The corresponding partial fault tree is given by an OR gate whose operands are: the internal failure of the exit part and all other eventual intermediate events that characterize failures coming from other input ports of the part.

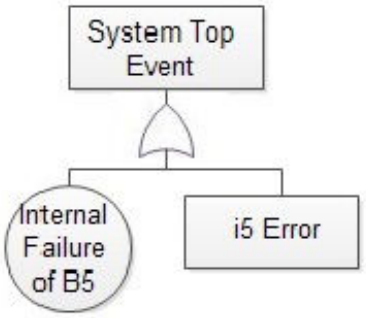


FIGURE 4.8: Fault Tree for Exit Pattern in Figure 4.6

4.4.2.3 Feedback Pattern

By traversing the directed graph representing an IBD, if we encounter a node that has already been visited, then we have a loop or a "Feedback pattern" in the current graph. In Figure 4.6, when generating the logic diagram for the output port o4-2 of the part B4, we need to take into account the input port i4 which comes from the output port o3 of the part B3. In its turn, the logic diagram of o3 must consider errors that may come from i3-2, propagated from B4. A cut can be realized here in order not to take into consideration the input ports such as i4 as an operand of the OR gate. The corresponding fault tree of the feedback pattern of Figure 4.6 is illustrated in Figure 4.9.

4.4.2.4 Redundant Pattern

When a part in an IBD receives item flows coming from redundant blocks that carry out the same system function, then we have a "Redundant pattern" (B2,

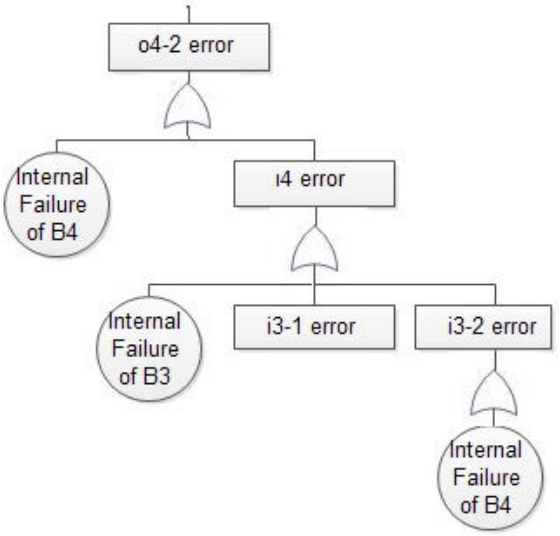


FIGURE 4.9: Fault Tree for Feedback Pattern in Figure 4.6

B11 and B12 in Figure 4.6). By using the safety profile described in section 2.3, the blocks B11 and B12 are stereotyped "redundant" and, in order to ensure consistency, the two blocks must be allocated to the same system function. In this case, an AND gate is used for different faults coming from different inputs to model the fact that if there is no internal failure in the component B2, the component will not work only if all the redundant item flows fail. The fault tree for our example of redundant pattern is given in Figure 4.10.

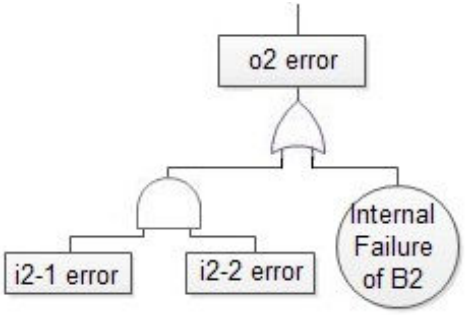


FIGURE 4.10: Fault Tree for Redundant Pattern in Figure 4.6

When the whole fault tree is generated automatically from an IBD by using the identified patterns and a graph traversal algorithm, we will have a generic fault tree for the corresponding system. In order to have a specific fault tree for an undesired top event failure, information from previous safety analysis results, i.e. component FMEA can be used to refine this generic fault tree. Knowledge of safety experts is also very important in order to detail some branches with different failure modes

or to cut out some unreachable branches, regarding the undesired top event failure. This proposal will be explained via the case study given in Section 4.5.

4.4.3 Implementation

As in the FMEA generation, the user can choose to generate a fault tree from an Internal Block Diagram found in a given package. There are two possible output formats: a .svg image file or a .psa file following the Open PSA Model Exchange Format (http://www.open-psa.org/joomla1.5/index.php). The FTA generation has three steps: internal block diagram extraction, fault tree generation and fault tree prettyprint. The first two steps are common for the different formats, and only the back-end step is specific.

1) Internal Block Diagram Extraction:

The graph for internal block diagram data is built of vertices representing blocks and ports while maintaining only id, type and name information. The edges of the graph showing blocks and ports that are connected to each other in the original internal block diagram via connectors.

2) Fault Tree Generation:

This step follows the algorithm described in Section 4.4.1. The result of the step is a tree whose root is a port with no successor (output port) and at least one ancestor. We have different Python classes for different kinds of node in a fault tree : LogicalGate (AND and OR ports for the moment), Leaf (Internal and External Failure) and FTANode for internal nodes. A simple depth-first search on the graph built from the previous step is implemented, by keeping the list of visited vertices to avoid cycles.

3) Fault Tree Prettyprint

- SVG File

  Tree layout is an NP-complete problem, so we have tried to implement different techniques (http://billmill.org/pymag-trees/) to make an acceptable representation. First, the graphical layout of the tree computed from the previous step is stored in the Python TreeLayout class, and then its size (vertical and horizontal dimensions) is defined. The Python svgwrite library

is used to implement Scalable Vector Graphics drawing objects. The choice of SVG, an XML-based vector image format for interactivity and animation, is justified by modern web browsers support.

- Open PSA Model Exchange Format

  The algorithm to transform a fault tree into Open PSA format performs a breadth-first search on the tree generated from the previous step. It is straightforward to carry out the transformation by adding necessary tags to define or to reference an element in the output file.

4.5 Case Study

The case study considered in this chapter is the Electro-Mechanical Actuator (EMA) presented in Chapter 3. In this step, we consider that all the previous steps, i.e. the functional architecture as well as the functional FMEA and the logical architecture with the component FMEA of the EMA are already established. As a reminder, the obtained logical architecture for the EMA is given in Figure 3.11. The three parts constituting the EMA are named respectively *Geared Motor with Encoder* for the electric motor, *Mechanical Transmission* for the mechanical transmission and *Embedded MCU with Power Bridge* for the electronic components and the code that control the system.

In the previous step, the component FMEA containing the list of components was automatically generated from the IBD in Figure 3.11 and then completed. To ensure the consistency of the component FMEA with respect to the functional FMEA, the functions allocated to each component and their failure modes are also automatically added. The safety expert then associates the functional failure modes into the corresponding component failures and also checks and completes the other columns in consistency with the functional FMEA. For more detail about this step, please refer to the Section 3.5 of the Chapter 3.

Figure **??** shows an extract of the final component FMEA for the EMA containing the failure modes leading to the "Aileron locked" undesired event, which will be used for the fault tree generation. The pre-filled data is written in italic font and red color, and the information added by safety expert is written in normal text.

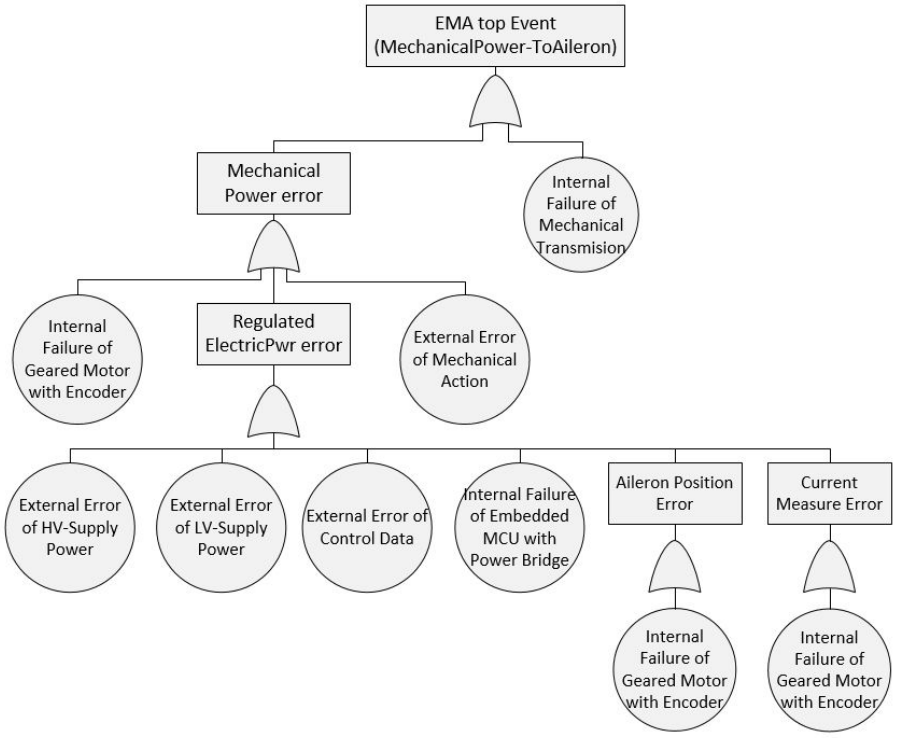


FIGURE 4.11: EMA Generic Fault Tree

Based on the system model given in Figure 3.11, a generic fault tree for a top event at a specific output of the system is automatically generated. This fault tree is built from partial fault trees generated according to the patterns and the graph traversal algorithm given in section 4.4. The generic fault tree for the "Mechanical Power" output is given in Figure 4.12.

Several undesired events can occur at each output. For each specific undesired top event, we can extract from the FMEA results the corresponding failure modes of each component that lead to the top event in question. The specific fault tree for the "Aileron locked" top event is given in Figure 4.13. In this fault tree, a branch is eliminated because no failure mode of the Geared Motor with Encoder leads to the "Aileron locked" event. Internal failure of some components such as Geared Motor with Encoder and Embedded MCU with Power Bridge is completed by their specific failure modes.

The FMEA and FTA generation has already been implemented by using programs written in Python language. The FMEA program parses an XMI file representing the SysML models and generates an Excel file corresponding to the functional or the component FMEA worksheet. This Excel file can be then completed by safety analysts, and parsed again by another Python script in order to insert new

Component	Function	Function failure mode	Failure mode		System effect
Embedded MCU with Power Bridge	Adapt Supply Energy	Fails to perform	hardware defect	...	Aileron locked in a position or out of control
	Translate Pilot Instructions	Fails to perform	software fault/error	...	Aileron locked in a position or out of control
			hardware defect	...	Aileron locked in a position or out of control
			synchronization error	...	Aileron locked in a position or out of control
			Memory defect/loss	...	Aileron locked in a position or out of control
Geared Motor with Encooder	Adapt Mechanical Energy	Fails to perform	Loss of structural integrity	...	Aileron locked or free mouvement of the aileron
			jamming	...	Aileron locked
	Transform Electrical - Mechanical Energy	Fails to perform	Jamming (stall)	...	Aileron locked
			Loss of structural integrity	...	Aileron locked or free motion
			short-circuit between two windings	...	Aileron locked or free motion
Mechanical Transmission	Transmit Mechanical Energy	Fails to perform	Loss of structural integrity	...	Aileron locked or free mouvement of the aileron
			jamming	...	Aileron locked

FIGURE 4.12: Extract of component FMEA

information into the safety stereotypes integrated in the XMI file, via the safety profile extension. The FTA program analyzes the same XMI file and can generate fault trees in different formats. Actually, two formats are proposed: an image format useful for visualizing the fault tree and an Open-PSA (Probabilistic Safety Assessment) format to make the results exploitable by the XFTA engine [27]. In case of need, implementation can be performed to include other formats to make the generated fault trees exploitable by other existing fault tree analysis tools.

4.6 Conclusion

In this chapter, the automated fault tree generation is presented. This is a step of our methodology to integrate safety analysis within a SysML-based systems engineering approach. The automatic generation of safety analysis artifacts (for instance fault trees) will help reducing time and errors. Consistency between

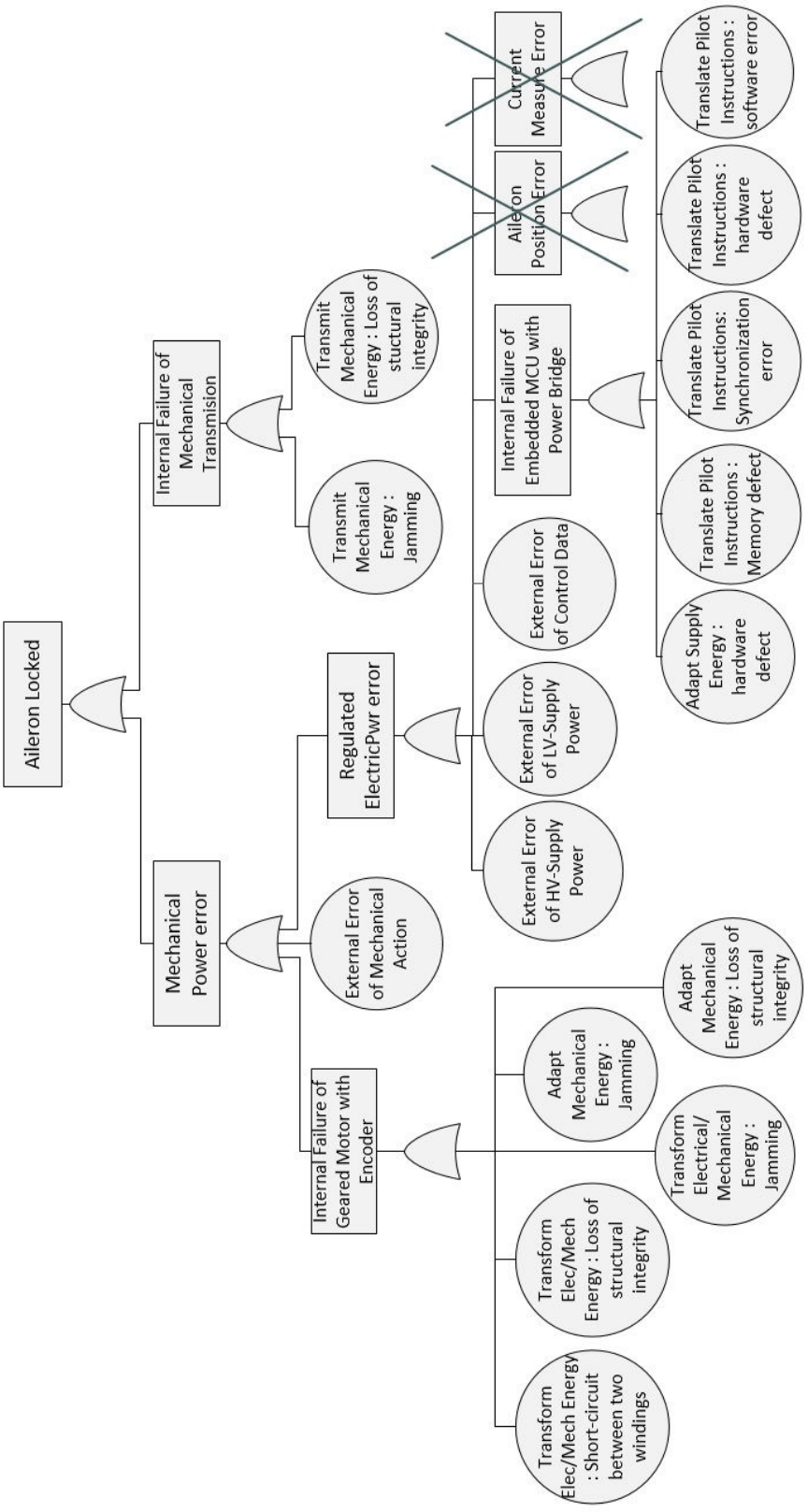


FIGURE 4.13: EMA Specific Fault Tree for " Aileron locked" Top Event

safety analyses and design is maintained throughout this integrated process since FMEAs and fault trees are directly generated from the latest system model versions. Consistency between different safety analyses is also maintained since each step is based on the previous analysis results. Indeed, the fault tree generation also relies on the component FMEA results.

In this work, only basic gates (i.e. AND and OR) are considered. In future work, we will try to deal with more advanced feature by detailing the different kinds of redundancy and including more gates in the generated fault trees.

Chapter 5

Behavioral Safety Analysis

5.1 Introduction

The previous chapters dealt with compositional safety analysis, i.e. safety analyses based on system composition (or hierarchical models) [22] and particularly focused on FMEA and FTA. These artifacts analyze the relationships between failures and their effects and causes based on the system topology. They are valuable as they can be started early in the design process and early identify weak design points such as single points of failures. Consequently, they influence design decisions and help designers to improve their work by choosing appropriate components and adding redundancies when needed, and help also in deriving and refining requirements [22].

The key limitation of these techniques however, is the fact that they are mainly static analyses and they do not take into account the dynamic behavior of the system. A comprehensive safety analysis must take into account both compositional and behavioral aspects. Behavioral Safety Analysis (BSA) [22] aims at assessing that the system really performs the specified behavior. In this approach, system-level effects of failures are established by the injection of faults into the system formal specification and observing the effects of these faults on the system behavior. The analysis is usually carried out by using mathematical tools such as guarded state graphs, Petri Nets, theorem proving or model checking. The use of formal methods to assess system behavior is recommended by some standards for safety critical systems thanks to the rigor of these methods [90].

This chapter deals with the behavioral safety analysis by giving an overview of its purposes and techniques. It also represents our approach using model checking for safety requirements assessment. The SysML state machines of different components representing the nominal and error states as well as the IBD of the system modeling error propagation are exploited to generate a NuSMV [39] program. This program, which is an abstraction model of the system, will be used to verify if some safety requirements (written in temporal logic formulas) are satisfied. A simplified case study, the Wheel Brake System (WBS) is used to illustrate the approach.

The chapter is organized as follows. First, the principle of model checking and some related work concerning the application of model checking in safety analysis are given in section 5.2. Our proposal to use SysML state-machines for state transitions and IBD for error propagation in order to generate automatically a NuSMV program to verify safety requirements is presented in section 5.3. Section 5.4 describes in detail the Wheel Brake System case study, from preliminary analysis to safety requirements formal verification. Conclusions are given in section 5.5.

5.2 Formal Methods for Safety Analysis

5.2.1 Model Checking Principle

Model checking is a formal verification method used to verify a set of desired behavioral properties (usually related to safety requirements) of a system through exhaustive enumeration of all the states reachable by the system and the behavior that goes through them. This automated process receives a model of the system and one or more temporal logic formulas [91] representing the properties to be verified, and then determines whether the system satisfies these properties or not. If the properties are not satisfied, the model checker provides a counter example (a path or a sequence of steps leading to undesired state) to help the designer perform corrective design changes. This process is described in Figure 5.1.

Because of automation and counter-example generation facility, model checkers have become popular debugging tools and have been used in reactive systems

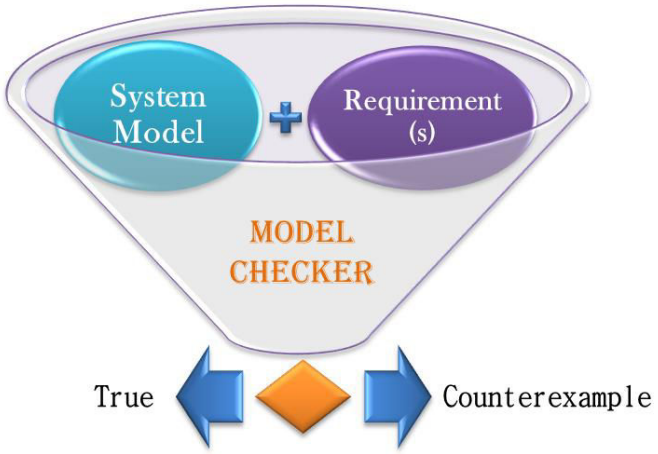


FIGURE 5.1: Model Checking Principle

such as hardware and circuit designs, communication protocols, telecommunication switches and even user interfaces. The application domain of model checking is usually finite-state concurrent system. As described by Clarke et al. in [92], the process of model checking consists of the following tasks.

5.2.1.1 Modeling

In this phase, the formal model of the system is built by converting a system into a formal model accepted by the model checking tool that will be used. Abstraction may be used to eliminate irrelevant or unimportant details of the system with respect to the specified properties that we check for the correctness.

5.2.1.2 Specification

The properties that the system must satisfy are stated by e specification process. The specification often uses a *temporal logic* formalism, which can assert how the behavior of the system evolves over time. This formalism is used for describing sequences of transitions among states in a reactive system. A temporal logic formula might specify that *eventually* some selected state is reached, or that an error state is *never* entered, by using special *temporal operators*. However, a problem in property specification is the *completeness*. It is impossible to determine whether a given specification covers all the properties that the system should satisfy or not.

5.2.1.3 Verification

Given a model with a set of states that represents a finite-state concurrent system and a temporal formula describing the specification, the model checking problem is to find the set of all states that satisfy this formula. The verification is done automatically by a model checker but the analysis of the verification results often involves human assistance. In case of negative results, a counter example of the checked property is used to help the designer to find out where the error occurred. However, an error trace can also result from incorrect modeling of the system or from an incorrect specification. Another possibility is that the verification fails to terminate normally, due to the size of the model, which is too large with respect to the memory capacity. In either case, the modeling or the specification or the verification phase must be adjusted and then the process be repeated.

The major problem with model checking is the state explosion problem. The size of the systems under analysis grows exponentially with the number of variables in the system. This makes model checking impractical for all but small systems. Some existing advanced techniques can deal with the state explosion problem using symbolic representations based on ordered binary decision diagrams, partial order reduction, modular structure with the assume-guarantee reasoning strategy, abstraction, symmetry, induction.

Another disadvantage of model checking is the restriction to finite state systems. One possible solution is the integration with the theorem proving method, which has so far been the only viable alternative.

5.2.2 Related Work

The application of model checking in safety analysis has been studied in many researches [22, 38–41] for safety requirements verification and/or automatic fault trees generation.

FSAP/NuSMV-SA [39, 89] is an automated safety analysis tool that aims at providing a uniform environment for design and safety assessment of complex systems. It provides a library of predefined failure modes that can be injected to the initial system model to augment it with failure behavior and thus create a so-called extended system model. By having both nominal and extended modeling, the tool

allows to assess the system safety both in nominal conditions and in user-specified degraded situations, that is, in the presence of faults. This platform is made up of two components: FSAP (Formal Safety Analysis Platform), which provides a graphical user interface, and NuSMV-SA, which is a safety analysis engine based on the NuSMV model checker. The FSAP/NuSMV-SA takes as input a system model in NuSMV format and automatically produces the analysis results. As a result, the tool produces property verification results as well as counter examples if the verified properties do not hold, minimal cut-sets and fault trees. The safety requirements (i.e., the properties to verify) are expressed in temporal logic and can be subsequently verified using the NuSMV model checking verification engine. The model checker is used as a validation tool.

AltaRica is another formal specification language that was designed to specify the behavior of complex systems [38, 42, 43]. An AltaRica model is composed of nodes that are characterized by their reachable states, in and out flows, events, transitions and assertions. The main phases of safety assessment with AltaRica include system modeling, formal safety requirements, graphical interactive simulation and safety assessment [38]. Once a system model is specified in the AltaRica language, it can be compiled into a lower level formalism such as finite-state machines, fault trees and stochastic Petri Nets. The safety requirements are formalized with the use of linear temporal logic operators and the formal verification technique can be performed by the AltaRica's MEC 5 model checker. To perform safety analysis with AltaRica, a new model of the system shall be built in AltaRica language which does not guarantee consistency between the two models. The model checker is also limited by the size of systems it can handle.

Joshi et al described in their report [40] the so called Model-Based Safety Analysis in which the nominal (non-failure) system behavior captured in model-based development is augmented with the fault behavior of the system. Like in [22], temporal logic is used to formalize informal safety requirements, and the model checker NuSMV is used to validate these requirements. To illustrate the process, a case study about the wheel brake system is well detailed in the report, with a fault model consisting of different component failures, i.e. digital and mechanical failure modes. Fault tolerance verification is carried out by using additional variables and real-time temporal logic operators to investigate if the system can handle some fixed number of faults. Nonetheless, the model-based development studied in this paper addresses principally Simulink. In this thesis however, we

are interested in integrating safety analysis into a more general framework, namely systems engineering via SysML.

In her thesis [22], Sharvia dealt with the integration of the Compositional Safety Analysis (CSA) and the Behavioral Safety Analysis (BSA). In her work, BSA is based on the CSA results, i.e., system failure models and preliminary information about state-automata that represent the transition between normal and failure states of the system. Next, in the BSA, model checking can be carried out on these behavioral models in order to verify automatically the satisfaction of safety properties. So the CSA and BSA could be effectively combined to benefit from the advantages of both approaches. Even so, behavioral information captured from CSA is rather limited because its main purpose is the failure propagation and hierarchy, not the dynamic behavior.

Yakymets et al. in [87] present a Safety Modeling Framework for Fault Tree generation SMF-FTA. This framework includes meta-models, profiles, model transformation, verification and FTA tools. It enables the use of formal verification and FTA algorithms during MBSE process. The model transformations tools in this framework allow the transformation of SysML models into several formats like AltaRica.

5.3 Our Proposal: Model Checking with SysML

5.3.1 Principle

There are many works about model-based safety analysis but few researches have addressed the direct connection between SysML and model checking. As our integrated methodology SafeSysE is based on the use of SysML to model the system, we will also integrate formal verification with model checking directly with SysML models in our approach. In this process, the dysfunctional behavior of the system and its components is captured in SysML via state machine diagrams describing the system/component states (nominal, degraded and failed) and the transitions between them. Combining the information provided by these state machine diagrams with the information provided by internal block diagrams, we can generate automatically the NuSMV model that will serve for verifying the given properties we want to assess the system for. The mapping between SysML

models and the NuSMV model is shown in Figures 5.2 and 5.3 for the component modules and main module respectively.

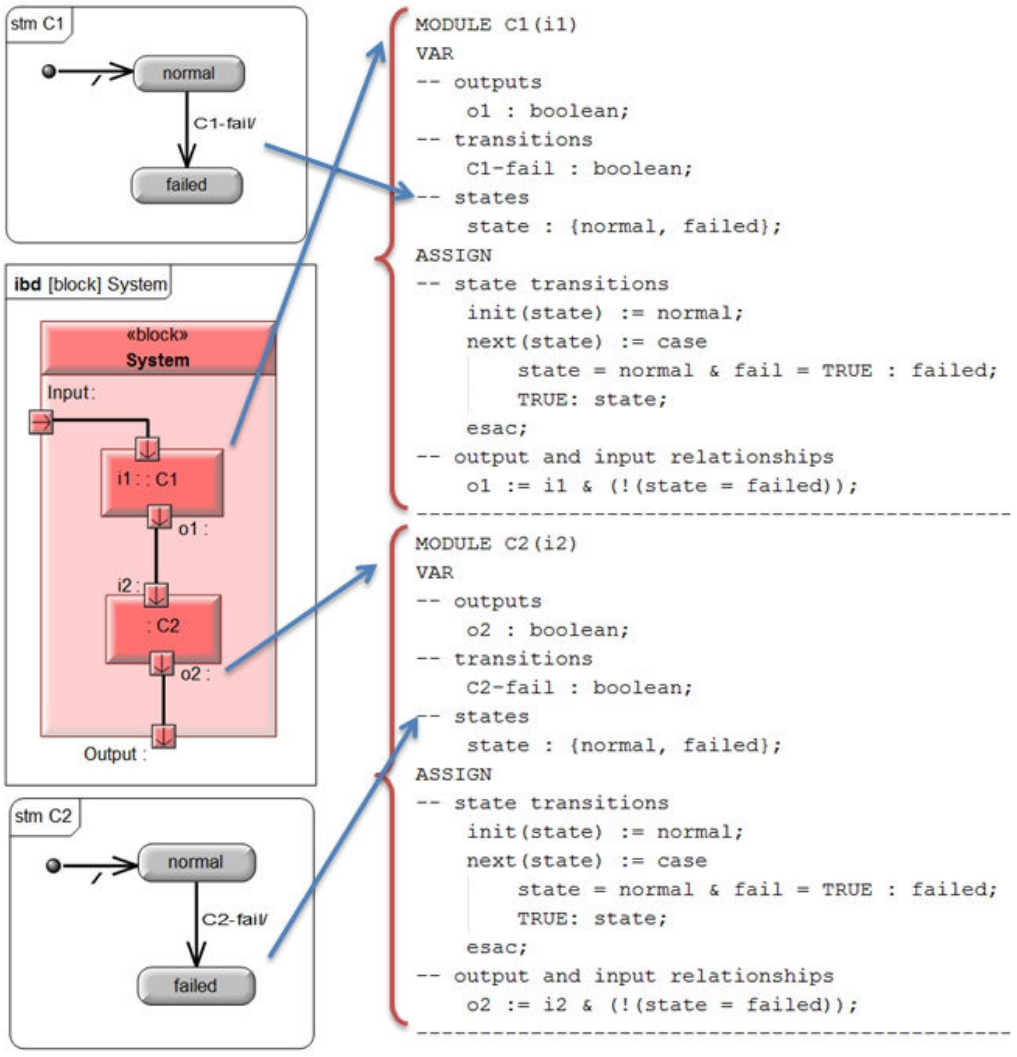


FIGURE 5.2: NuSMV Component Modules Mapping with SysML diagrams

Once the NuSMV program is generated, manual modifications can be added to precise the relationships between parameters and to define the requirement specifications.

5.3.2 Implementation

We have implemented a prototype for the NuSMV program generation in our SafeSysE tool. For this purpose, the parsed tree corresponding to the initial XMI file is used to build a graph containing information about the state machine of

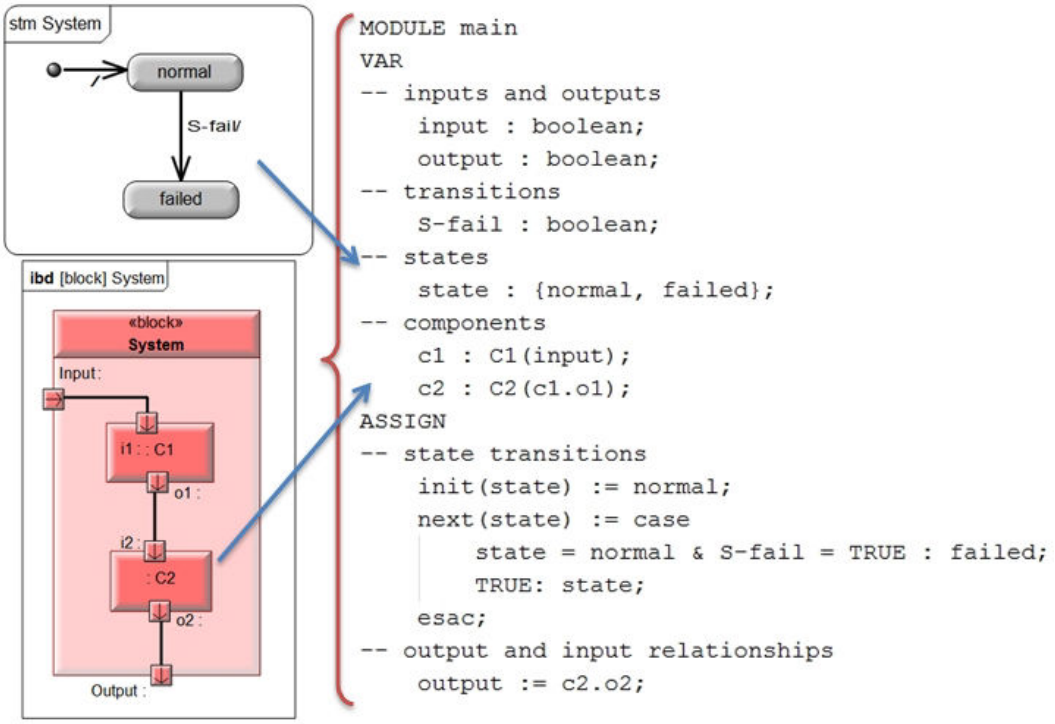


FIGURE 5.3: Main NuSMV Module Structure

the system and of the components: the nodes of this graph are the states and the edges are the transitions. Connections between blocks in the Internal Block Diagram are also taken into account to model failure propagation. By using the integrated safety profile, we are able to identify a nominal, a degraded or a failed state of the component or the system.

A NuSMV module is generated for each component. The list of formal arguments of the module contains all input ports of the corresponding block in the IBD. The module variable declarations are added by taking all output ports of the block as well as transitions in the state machine of the component. The initial state is easily identified by using the state machine information. Equations establishing the relationships between input and output values, as well as nominal and failure states are also given. The main module corresponding to the system is then built from its internal states and variables representing different components. State transitions will take into account the relationships between input and output ports of different components. Once a NuSMV code is generated for the whole system, a safety requirement written in temporal logic syntax can be tested with a model checker to verify if the system satisfies this requirement.

The main problem encountered while developing this prototype, is the difficulty

to translate the properties to verify from natural language to the correct formula in temporal logic format. Indeed, translating safety requirements into temporal logic formula requires a knowledge in this domain and a systems engineer may be not very familiar with it. An idea to tackle this problem would be to propose a set of templates for expressing the properties in the natural language and provide an automatic translation for these well-known patterns.

5.4 Case study

The case study in this chapter is the Wheel Brake System (WBS) of an aircraft described in the Appendix L of the SAE-ARP-4761 standard [12], and also studied several works such as [16, 22, 40]. This WBS with other aircraft sub-systems contribute to achieving the system function: "Decelerate Aircraft on Ground". This function is safety critical since its failure could lead to catastrophic effects.

As the behavioral safety analysis is performed once the system structure and behavior are established, we will first give an overview of the previous steps including compositional safety analysis and system structure definition according to this analysis. Next, we will run the BSA in a very simplified way to illustrate the different steps of performing it.

5.4.1 Preliminary Analysis

In this section, the steps assumed to be already performed before the behavioral safety analysis are presented for the case study. These steps are those described in Figure 2.16. This top-down design process usually begins with requirements definition and analysis. Then a breakdown of system functions is established based on the defined system functional requirements. A functional decomposition of the aircraft functions is given in the SAE-ARP-4761 standard [12] but is outside the scope of our interest. In this work, we will focus on the sub-function "Decelerate Aircraft on Ground" of the higher level function "Control Aircraft on Ground".

A preliminary safety analysis of the "Decelerate Aircraft on Ground" shows that the considered function is critical to the functioning of the system since its failure could lead to catastrophic consequences like the aircraft leaving the runway or

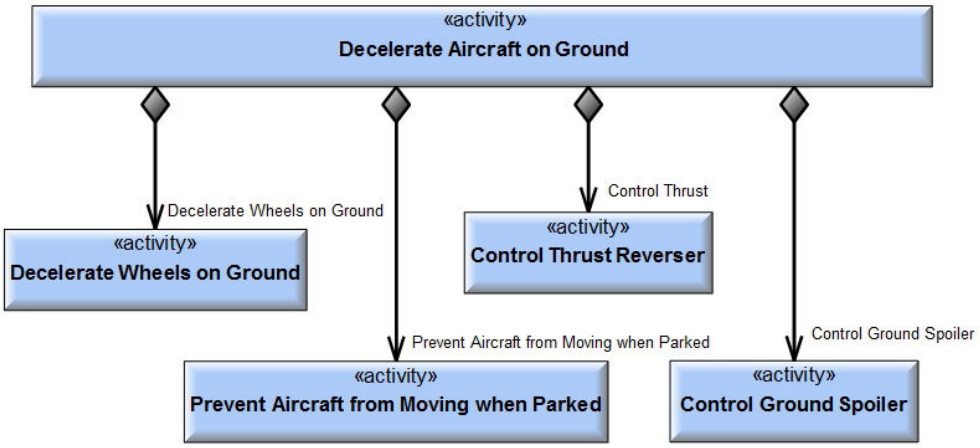


FIGURE 5.4: Breakdown of the "Decelerate Aircraft on Ground"

crashing the buildings or equipment in the airport. A safety requirement is derived from this analysis: "Loss of wheel braking during landing or rejected take-off shall have a frequency less than 5 E-7 per flight" [40]. This requirement is noted as SR-WBS.

A further breakdown of the "Decelerate Aircraft on Ground" shows that it has four sub-functions that are: "Decelerate Wheels on Ground", "Prevent Aircraft from Moving when Parked", "Control Thrust Reverser" and "Control Ground Spoiler" (Figure 5.4). Since the wheels deceleration has a stronger effect than other braking systems, our safety analysis will be hold on the function "Decelerate Wheels on Ground". The wheel braking system is allocated from this function. It is made of a physical part that actually brakes the wheels and a control part to monitor the functioning.

Based on safety analyses results, and given that the WBS performs a safety critical function, it must be fault tolerant. This property is obtained by using redundant components for both the control and the physical parts. The physical part contains two redundant hydraulic lines, a *Normal* line that is first activated and an *Alternate* one that is activated when the normal chain is inoperative. Each of the two systems has an independent power source the *NormalHydraulicResource* and the *AlternateHydraulicResource* respectively. A supplementary power source, called emergency power source, is also mandatory for the wheel-brake system in aircraft [6]. In our case, an *Accumulator* is added and provides the braking system with hydraulic power when all the other power sources are inoperative. The

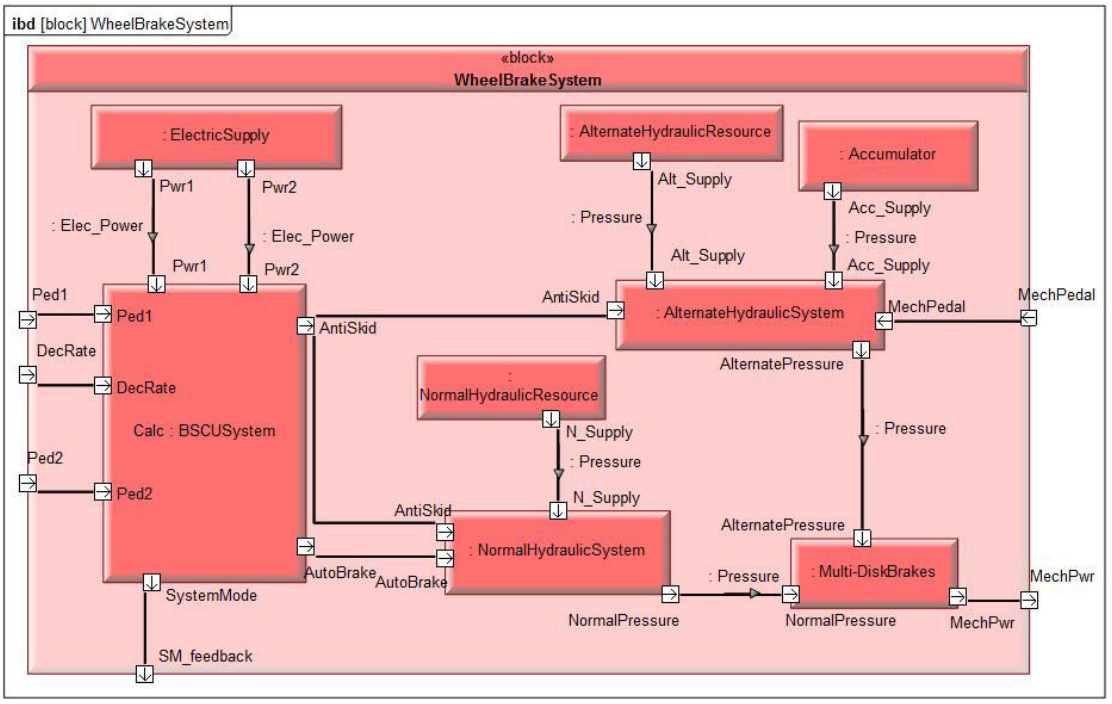


FIGURE 5.5: Wheel Brake System Structure

SysML Internal Block Diagram in Figure 5.5 describes the wheel brake system and the interactions between components. The interactions help analyzing the fault propagation through the system.

Once the system structure is defined, we can model the dysfunctional behavior and verify safety properties using formal methods. This will be the subject of the next section.

5.4.2 Safety Requirements Formal Verification

In this section, the behavioral safety analysis of the case study is described. The three steps of the model checking process are detailed respectively.

5.4.2.1 Modeling

The first step in performing model checking is building the formal model of the system. In our case, this model is obtained by converting the relevant parts of the SysML model of the system. To model the dysfunctional behavior of the WBS, we consider four modes or states. These states are:

- normal: when the Normal line is operated;
- alternate: activated when the Normal line fails and this failure is noted as **failNormal**;
- emergency: activated when Alternate mode also fails (**failAlternate**). The accumulator provides power source in this mode;
- fail: when all components are in failure state. The failure of the accumulator is noted as **failEmergency**.

These modes and the corresponding transitions are modeled in an automata describing the dynamic behavior of the system including fault effects given by a SysML State Machine Diagram (see Figure 5.6).

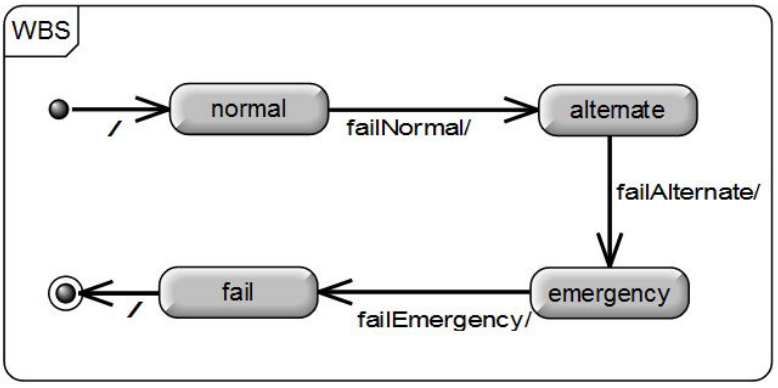


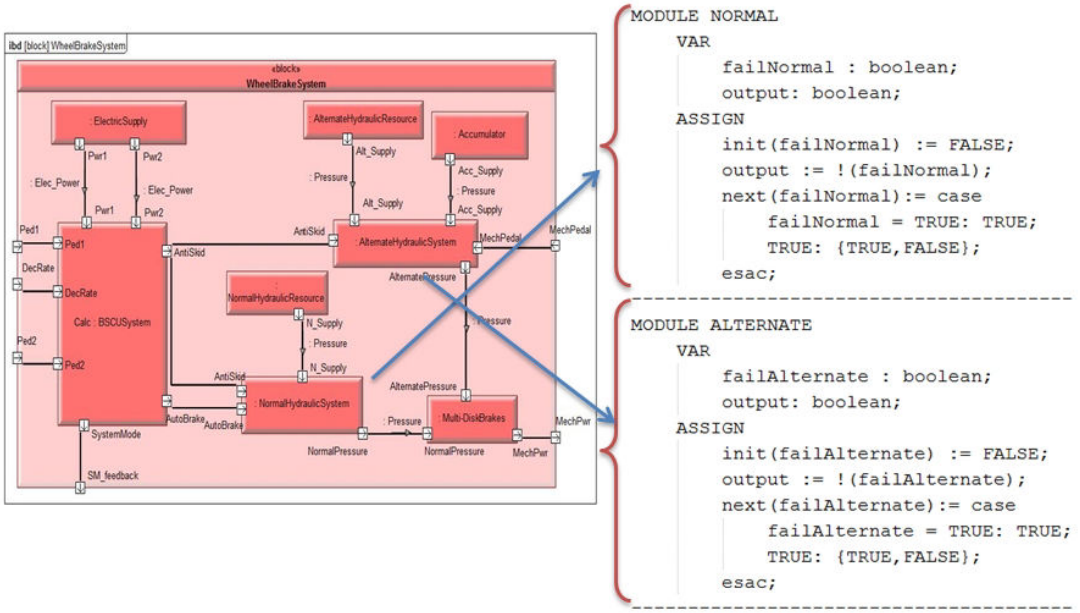
FIGURE 5.6: Dynamic Behavior Including Faults Automata

The NuSMV model is then generated from the SysML IBD (Figure 5.5) and this state machine diagram. The mapping principle between the two models is described in section 5.3.2 and is illustrated in Figure 5.7.

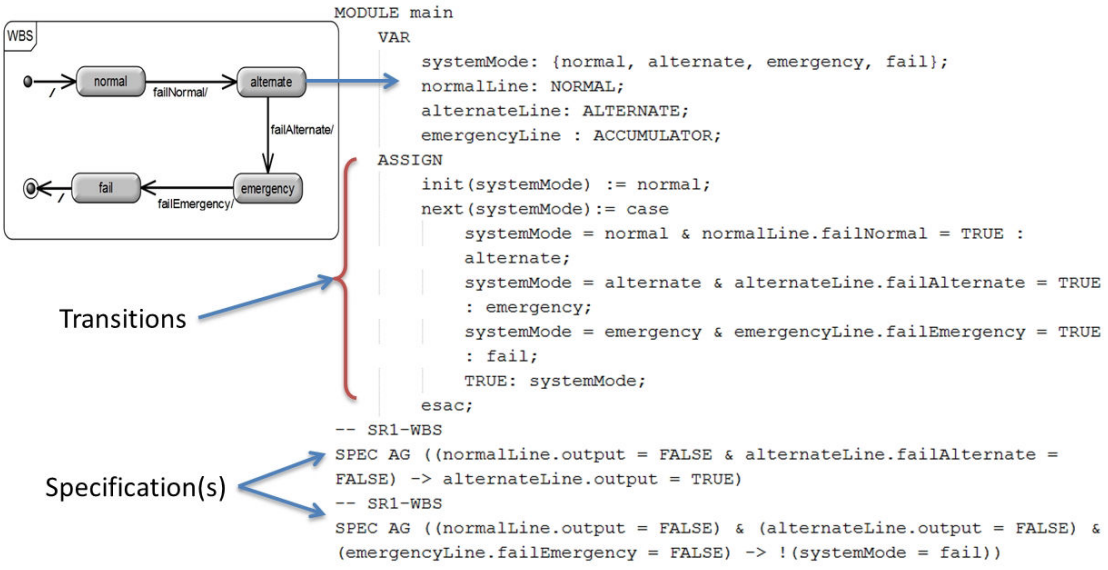
Once the formal model is built, the next step is the safety requirements specification.

5.4.2.2 Specification

The specification describes the properties that the system shall satisfy, in our case safety requirements. For the case study, a further refinement of the safety requirement SR-WBS results in the safety requirements SR1-WBS and SR2-WBS.



(a) Example of NuSMV Component Modules



(b) NuSMV Main Module Structure

FIGURE 5.7: Mapping Between SysML Model and NuSMV Model

These requirements specify the fault tolerance specifications required from the wheel brake system behavior. The WBS shall be able to brake the aircraft even in presence of a certain number of failures detailed in SR1-WBS and SR2-WBS.

- SR1-WBS: When output is not supplied by Normal Line, and there is no failure accounted in Alternate line, pressure shall be supplied from Alternate line.

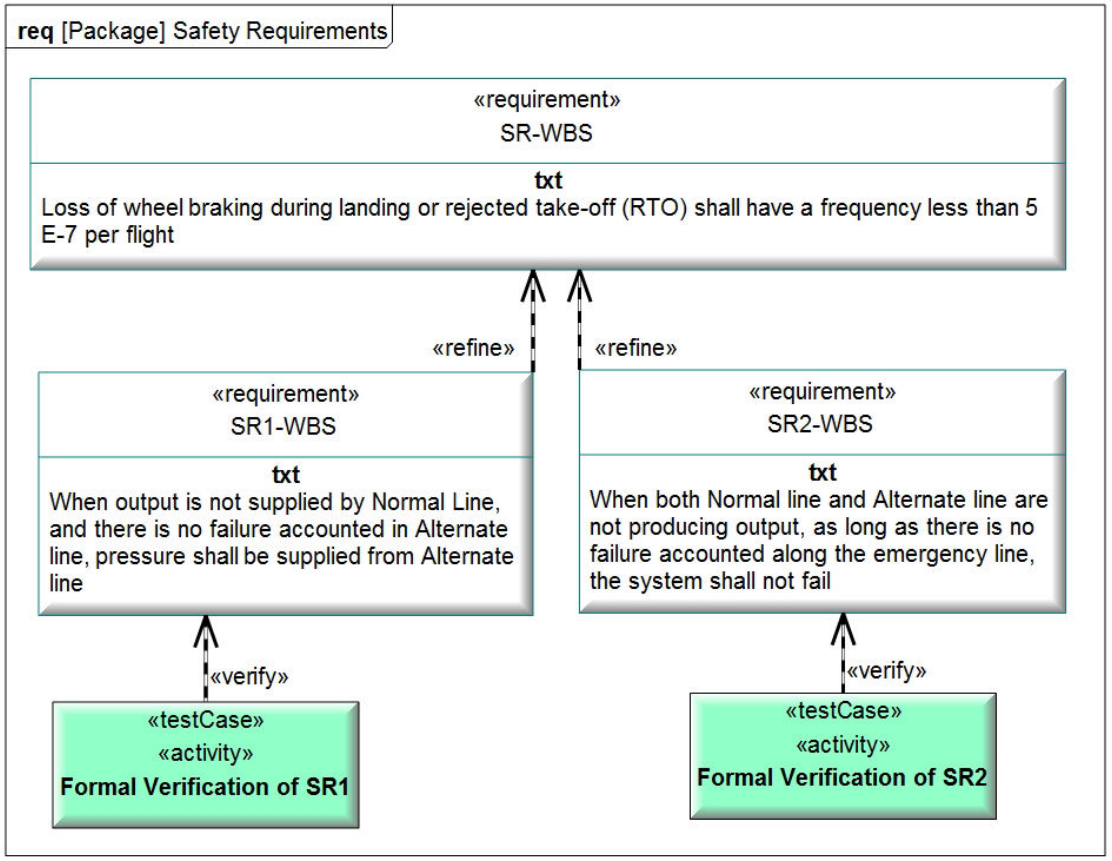


FIGURE 5.8: Safety Requirements in SysML

- SR2-WBS: When both Normal line and Alternate line are not producing output, as long as there is no failure accounted along the emergency line, the system shall not fail.

These requirements are captured in the SysML model and linked to corresponding test cases as shown in the SysML Requirements Diagram given in Figure 5.8.

In addition to the automata and IBD, the NuSMV program shall also contain the specification, i.e., the requirements to be verified (Figure 5.7 b).

The safety requirements SR1-WBS and SR2-WBS are respectively specified by temporal formulas F1-WBS and F2-WBS:

- F1-WBS: SPEC AG ((*normalLine.output* = False AND *alternateLine.failAlternate*= False) → *alternateLine.output*=True).
- F2-WBS: SPEC AG ((*normalLine.output* = False) AND (*alternateLine.output* = False) AND (*emergencyLine.failEmergency* = False) → !(*systemMode* = fail))

```
NuSMV >
NuSMV > check_ctlspec
-- specification AG ((normalLine.output = FALSE & alternateLine.failAlternate =
FALSE) -> alternateLine.output = TRUE)  is true
-- specification AG (((normalLine.output = FALSE & alternateLine.output = FALSE)
 & emergencyLine.failEmergency = FALSE) -> !(systemMode = fail))  is true
NuSMV >
```

FIGURE 5.9: The NuSMV Verification Result

In these formulas, the following operators are used:

- Logical operators:
  - !: logical NOT
  - AND: logical AND
- Temporal operators:
  - A that means All: A p means that the property p has to hold on all paths starting from the current state;
  - G that means Globally: G p means that p has to hold on the entire subsequent path.

More detail about temporal logic can be found in [91].

5.4.2.3 Verification

Once the whole model and the specifications are entered in the NuSMV model, the next step is the automated verification. Running the NuSMV program, these two properties are shown to hold. The solver returns true for both properties (see Figure 5.9).

5.5 Conclusion

In the previous chapters, we have shown how some safety analysis results such as FMEA and FTA can be generated from SysML structural and behavioral diagrams. In this chapter, we extend the work by integrating in this unique framework the automated verification of SysML models with regard to safety properties. However, there are still some limitations with our approach. The abstract model

of the example system is rather simple and can be refined to model more exactly the behavior of the system in case of various failure modes and different redundancy mechanism. In addition, the translation of informal safety requirements into temporal logic formulas is not always straightforward. A counter-example of the checked property can in fact result from incorrect modeling of the system or from an incorrect specification.

Our future work is to investigate in more detail the structure of the system to propose a more precise behavioral formal model with different failure modes propagation. Complex system properties for verifying system fault tolerance by using special temporal logic operators must be supported. At the same time, the scalability of the model checker must be tested in accordance to the size and the complexity of the models.

Conclusion

The increasing complexity that characterizes new manufactured systems is a real challenge for designers, mainly for systems engineers and safety experts that deal with the whole systems. The first ones, as system architects, have to manage this complexity generated by the interesting perspectives given by merging several engineering domains (mechanics, electronics, software, hardware, etc.,) while being constrained by the physical behavior and the interference of several physics implying taking into account constraints like Electro-Magnetic Compatibility (EMC), dynamics and vibrations, thermal couplings, etc.

SysML, the relatively new but widely used systems modeling language provides systems engineers with the ability to generate a consistent multi-view modeling of the system. This language supports model-based systems engineering approach and facilitates harmonizing the overall design process by paving the way for simulation and dimensioning of alternative solutions in order to manage trade-offs and choose the more relevant design for a given set of requirements.

The design of safety critical systems implies a particular focus on safety requirements to comply with the constraints imposed by regulations and safety standards. But nowadays, it seems obvious that system engineers are not fully equipped and trained to tackle such essential requirements. Safety experts, from their standpoint, have to deal with safety relevant aspects in order to verify, a posteriori, that all the relevant safety requirements are satisfied by the system. In the case of complex systems, their task becomes challenging. Traditional safety assessment methods and models such as FMEA and FTA, are of very great help for this purpose as they provide system architects with reports and recommendations that are of great interest in order to adjust or redesign the system. However, they are usually very long to perform and error-prone. If they occur late in the design process, designers will miss the opportunity of taking full advantage of their results and

directives to improve the system design. Here relies one main issue with regards to complex systems design: how to perform safety analyses efficiently (mainly by reducing development time and error proneness) to have full benefit of their results and thus avoid costly and time consuming redesign iterations.

The main contribution of this work is to deal with this issue by efficiently merging systems engineering process and safety assessment methods in a unique framework that we named Safety Integration in Systems Engineering approach: SafeSysE. This work is based on a SysML-based MBSE approach.

Our first goal is to establish a thorough SysML-based system engineering process for complex systems, such as mechatronic systems. Starting with a rigorous requirements engineering phase that delivers a consistent and traceable set of requirements, this process provides consistent functional, logical and physical views of the system, all of them being traceable up to the requirements. To be more adapted to mechatronic systems and taking the advantage of the extension mechanisms provided by SysML, we have developed a mechatronic extended modeling profile.

Our second goal is to efficiently integrate safety analysis into a SysML-based systems engineering process. For this, we proposed our integrated process SafeSysE in which we automated the generation of some well known safety assessment artifacts such as FMEA and FTA as well as the generation of NuSMV formal program for model checking. Again, we have added some extensions to SysML to integrate some safety relevant concepts in order to better support safety analyses. A SysML safety profile has been developed for this purpose.

The first safety analysis artifact considered in this work is the FMEA. Both functional and components FMEAs are generated by the automatic exploration of SysML model and extracting the relevant information to be added into the FMEAs. This facilitates the task of safety expert and reduces error-proneness and shortens the time dedicated to safety analyses. FMEAs identify the system failure modes and their effects on the system behavior. They provide system designers with recommended actions to improve their design. System architecture is then defined taking into account the results of these safety analyses.

Once a detailed system architecture is defined, as well as the undesired system states (provided by the system level effects of potential failure modes in the FMEA), the next step is to analyze the fault propagation along the system. This

is obtained by the generation of fault trees. By means of a thorough analysis of the system architecture defined in SysML IBDs, we have proposed an automated generation of fault trees. The generation of safety artifacts directly from the system models makes them fully consistent with the design models. A generic fault tree describing the fault propagation among the system is automatically generated and then detailed into a specific fault tree based on the component FMEA results.

During this integrated design process, some behavioral analysis may be conducted in order to make sure that a new system architecture is relevant for a new set of safety requirements revealed by the combination of systems engineering and safety assessment. Thus, we have proposed the integration of model-checking with SysML by providing a model-to-model transformation tool to automatically generate formal models from SysML models. A case study is treated during this work to illustrate how SafeSysE deals with complex system design.

This work, we believe, contributes to tackling the current issues of designing complex safety critical systems. The automated generation of safety artifacts reduces the time spent in performing safety analyses and thus reduces the whole development time and increases the competitiveness. It also reduces error proneness since it automatically extracts the relevant information from system models. The consistency between the different safety analysis artifacts is also enhanced since, in each step of safety analysis, we use the results of previous analyses.

The work carried out during this thesis made the subject of a journal paper [77] and several conference papers [93–97].

Previously mainly working on systems engineering, this work constituted our first step towards the very interesting domain of safety analysis. As a continuity to this thesis, we would like to carry out more in-depth work in this domain. We are particularly interested in dealing with the following points.

First, we will try to improve our work by applying SafeSysE to other examples of systems to identify potential lacks and bugs. This will lead to improve both the process and the developed prototyping tool. We will also try to resolve some encountered technical problems, most of them being inherent to the tools implementing SysML used during this work.

In addition, we will try to improve the consistency between the system model and the safety analysis artifacts by linking the safety properties of the system

to modeling elements like requirements, etc. We will also examine the different possible ways to model dysfunctional behavior in SysML in order to improve the verification and validation of the system. Finally, scalability will be addressed in order to prove the adequacy of SafeSysE for larger and more complex systems.

APPENDIX

Appendix A

Case study: The Electro-Mechanical Actuator (EMA)

A.1 Introduction

In this thesis an example of electro-mechanical actuator (EMA) is considered as a case study. Since we were unable to obtain any industrial information about the EMA because of confidentiality, we tried to study it with our limited knowledge of the real system, and without any return of experience. The models and analyses provided here express our point of view and may be criticized for lack of accuracy. The EMA aims at actuating the ailerons in an aircraft. The use of EMAs in flight control is increasing since they have many advantages [98]:

- Better environmental respect with suppression of hydraulic power and oil leak risks;
- Weight saving on aircraft;
- Maintenance cost reduction;
- Performance increase and speed accuracy due to electric actuators.

Their competitiveness in performance is due to the use of rare earth magnetic materials in electric motors, high-power solid-state switching devices, and microprocessors for lightweight control of the actuator motor [6].

If the feasibility and interest of electro-mechanical architecture are demonstrated in terms of performance and functionalities, there is still a difficulty to prove the safety adequacy of their use for safety critical purposes such as actuating flight control surfaces (Fig. A.1). The EMA is mainly made of three parts: an electric part constituted by an electric motor, a mechanical part formed by the transmission and an electronic and software part composed of a calculator that controls the system. Failure modes and their causes are very different for each of these component types.

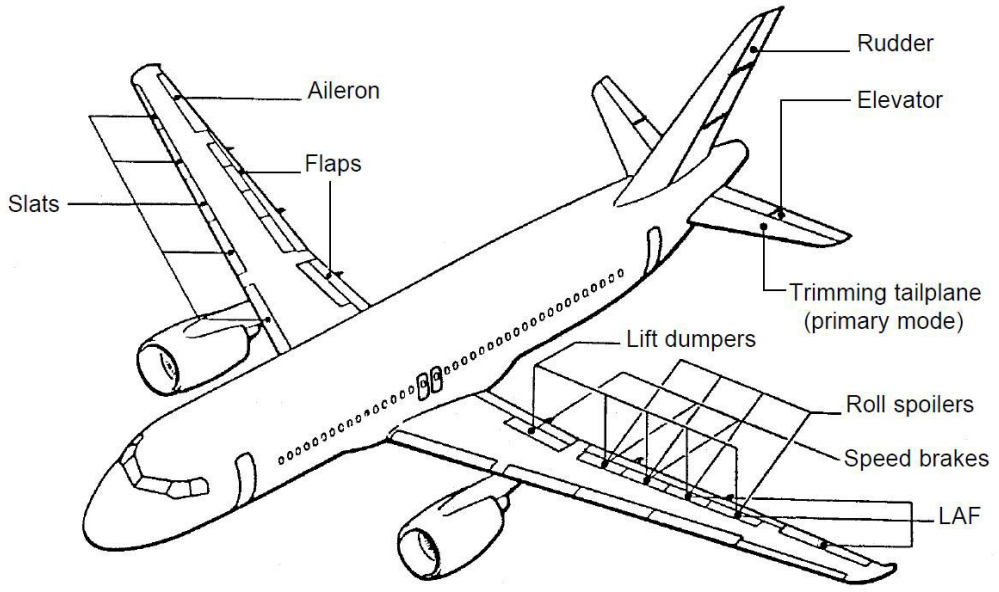


FIGURE A.1: Example of Flight Control Surfaces of a Commercial Airliner (A320) [6]

A.2 EMA Modeling with SysML

In this section, SysML models of the EMA are built using a top-down approach. This model only presents the parts that are not in the text of this manuscript and that are necessary to understand the case study. The different diagrams are presented within the corresponding design steps mentioned in Chapter 2.

A.2.1 Requirement Definition and Analysis

The first step in the EMA modeling is identifying the boundary of the system, its context of use, the external elements interacting with it and the missions it must fulfill. A SysML Block Definition Diagram, labelled *Context diagram* is used to describe the context of the system and its interactions with external elements (other systems or humans). Fig. A.2 details the context of the EMA, it indicates for instance that the EMA should be attached to the airplane wing and actuate the aileron. It should also deal with the airplane control unit and must be supplied with the electric power source available on the plane. It should finally stand the environmental conditions of the atmosphere during flight.

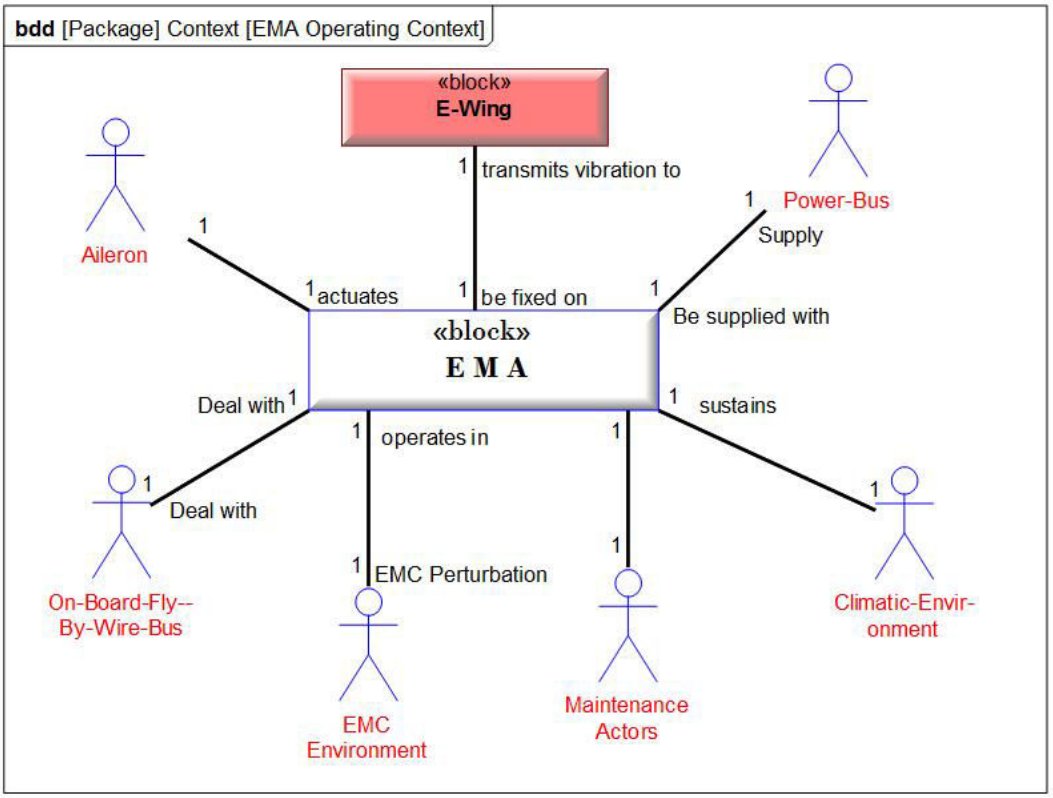


FIGURE A.2: Operating Context of the EMA

Modeling the context helps a better identification of the constraints that the system will face during its operation and also provides a set of requirements on the system like the interface requirements for instance.

The next step is modeling the operating modes in the nominal behavior of the system. These operating modes are modeled with a SysML state machine diagram given in Figure A.3. We consider that the initial state of the EMA is *Not powered*.

This corresponds to the case where the aircraft is on ground and the EMA is not powered. When the EMA is powered, the first step is an on-ground *CheckList*. If the check list is not ok (*CheckListKO*), then the EMA is in a maintenance phase. If the check list is successfully performed, then the EMA is idle *powered-idle* and waiting for the pilot instructions. When it receives the pilot instructions, the EMA alternates between *moving* and *stopped* states according to the pilot instructions and in conformance with the transitions in the state diagram.

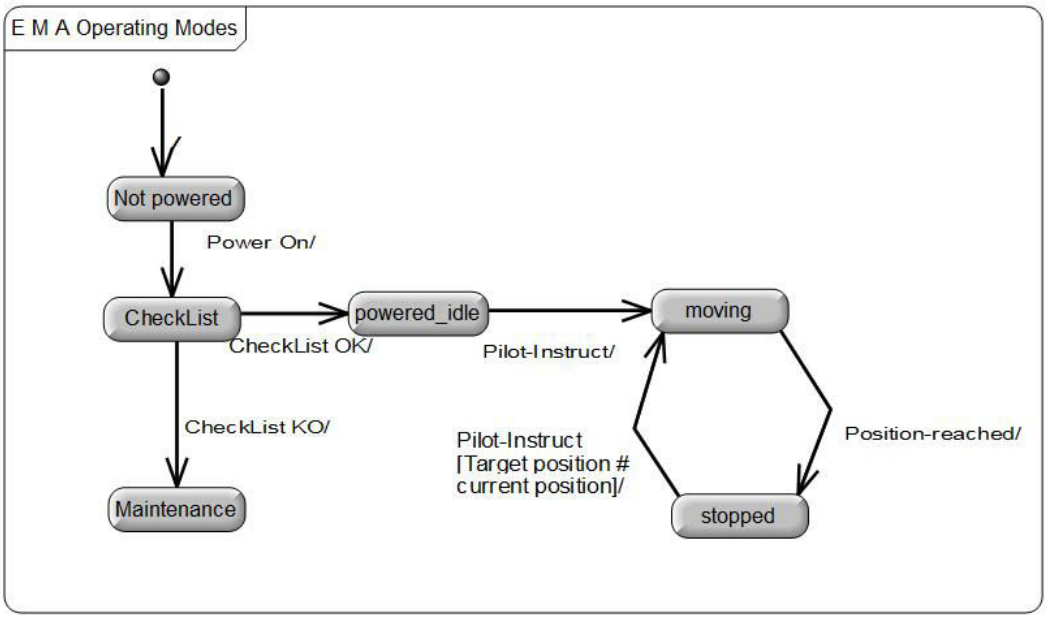


FIGURE A.3: Operating Modes of the EMA (state machine diagram)

An extract of the requirements issued from the Requirements Definition and Analysis process is given in Figure A.4. In this diagram, a focus is given on the decomposition of the main function while the constraints resulting from the context of the system are only mentioned.

A requirement table can also be extracted from SysML model. Requirement tables offer the possibility to display the requirements in a tabular form that is more usual and may be easier to explore in the case of a big number of requirements. The designer can chose the packages to analyze (i.e. t include in the table only the requirements in the selected packages) as well as the properties and/or relationships that are of interest. A requirement table containing only the textual description of the initial requirements is given in Figure A.5.

The use cases of the EMA are represented in Figure A.6. It shows that to control the aileron incidence, the EMA must *Deal with On Board Control Unit* and *Actuate*

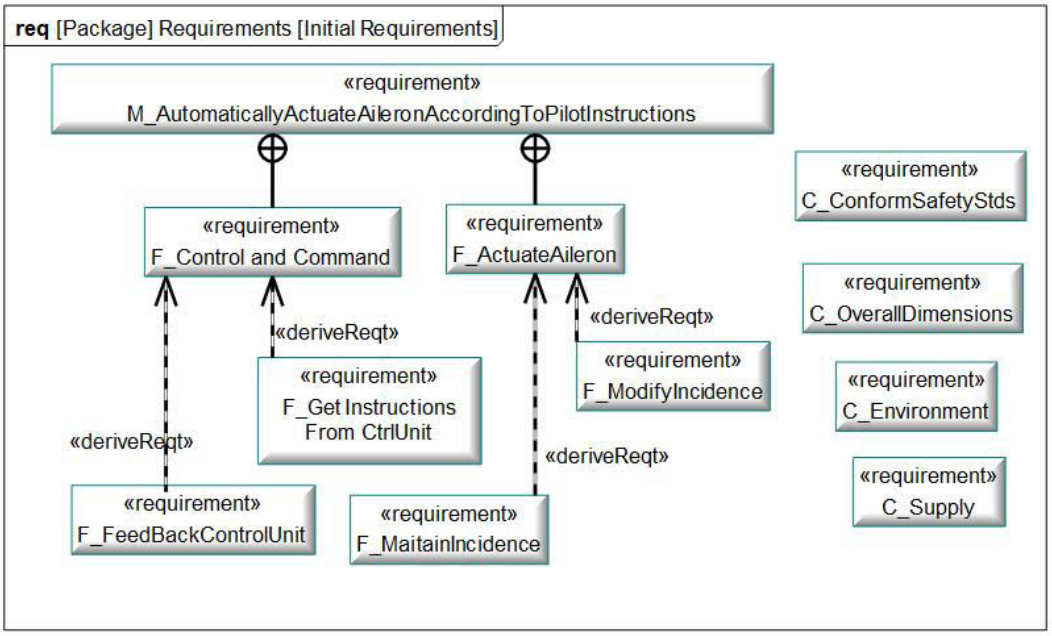


FIGURE A.4: Extract of the EMA Initial Requirements

Name	Txt
C_ConformSafetyStds	The System shall conform to safety standards
C_Environment	The System shall be adapted to the environment (temperture, humidity...).
C_OverallDimensions	The System shall be inserted on the wing of the aircraft. The overall dimensions shall fit into the wing.
C_Supply	The System shall be supplied with the available voltage on the aircraft.
F_FeedBackControlUnit	The System shall feed back the control unit
F_Get Instructions From CtrlUnit	The System shall recieve instructions from control unit.
F_MaitainIncidence	The System shall maintain aileron incidence
F_ModifyIncidence	The System shall modify the aileron incidence to conform withe pilot/autopilot instructions
M_AutomaticallyActuateAileronAccordingToPilotInstructions	The System shall automatically actuate aileron according to the pilot/autopilot instructions.
F_ActuateAileron	The System shall move the aileron according to pilot instructions (transmitted by the control unit)
F_Control and Command	The System shall be able to control and command the aileron.

FIGURE A.5: Extract of the EMA Initial Requirements Table

Aileron with regard to provided instructions. In this use case, the system interacts with the *Aileron*, the aircraft *Wing*, the *Power-Bus* and with the control unit through the *On-Board-Fly-By-Wire-Bus*.

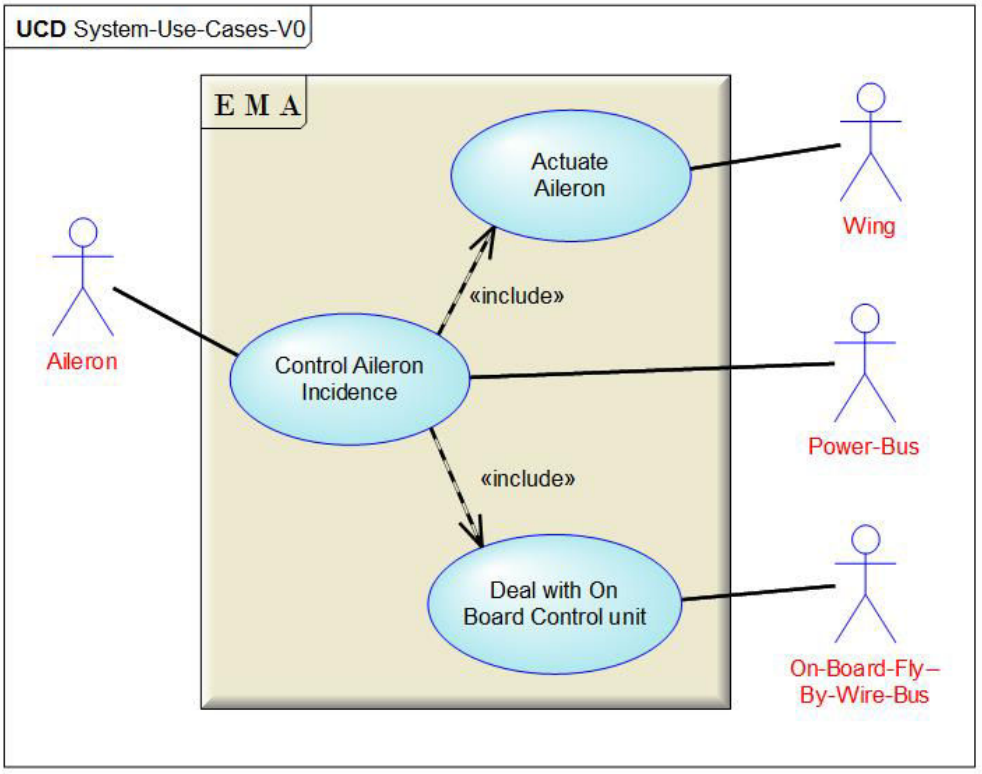


FIGURE A.6: EMA Use Cases

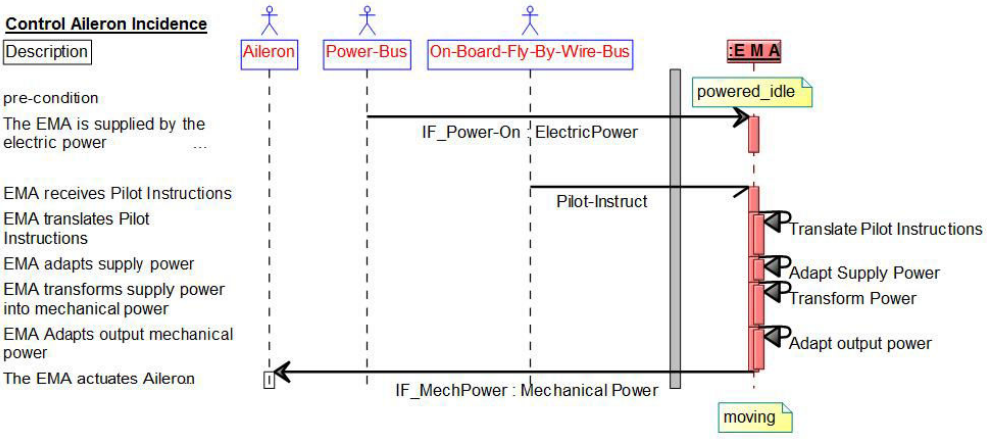


FIGURE A.7: EMA Sequence Diagram

A.3 Functional Architecture Definition

The same decomposition is represented by the high level activity diagram in Figure A.8. Then a further decomposition is realized; the sub-functions *c* and *Actuate Aileron* are detailed in their own activity diagrams in Figure A.9 and Figure A.10 respectively.

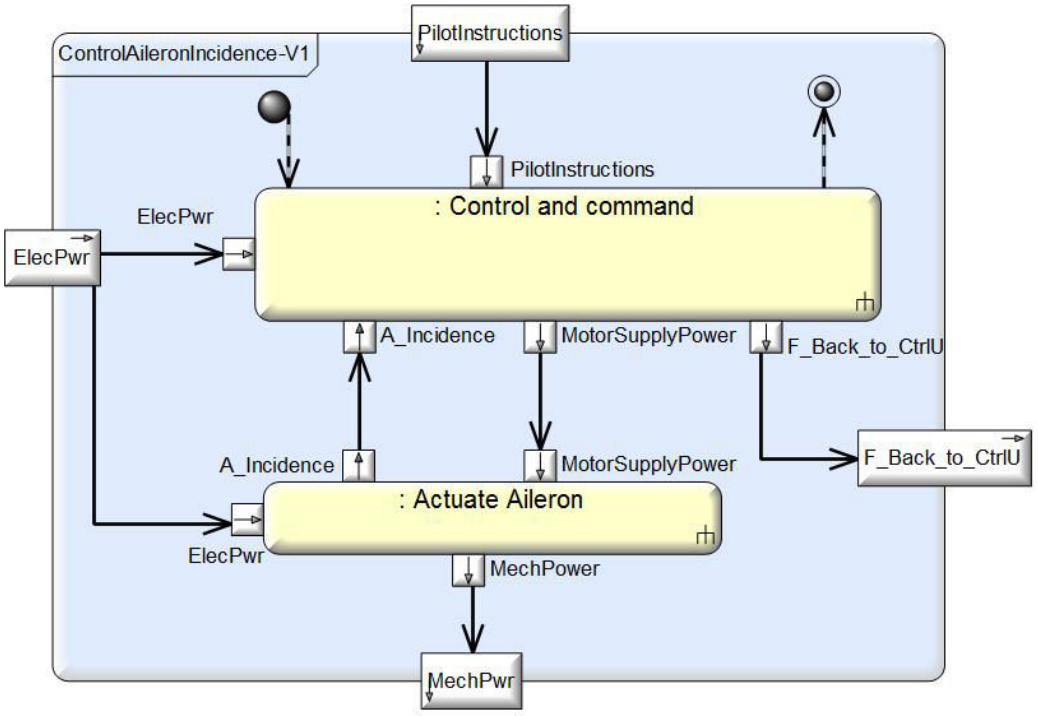


FIGURE A.8: Activity Diagram

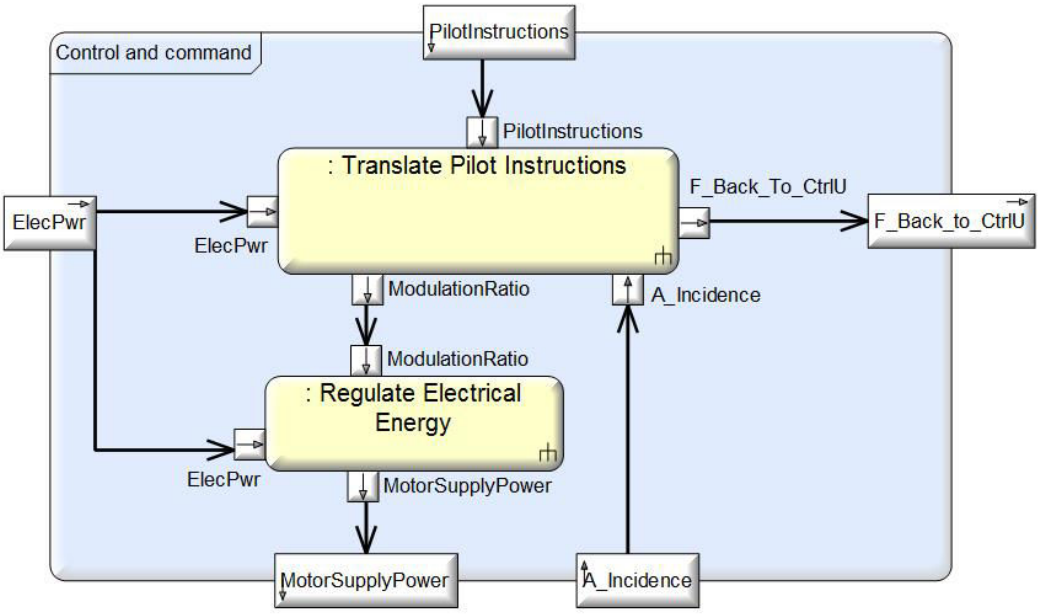


FIGURE A.9: Functional Decomposition of the 'Control and command' function

This leads to an updated list of requirements including functional requirements corresponding to the identified functions (represented by activities). The new requirement diagram is given in Figure A.11

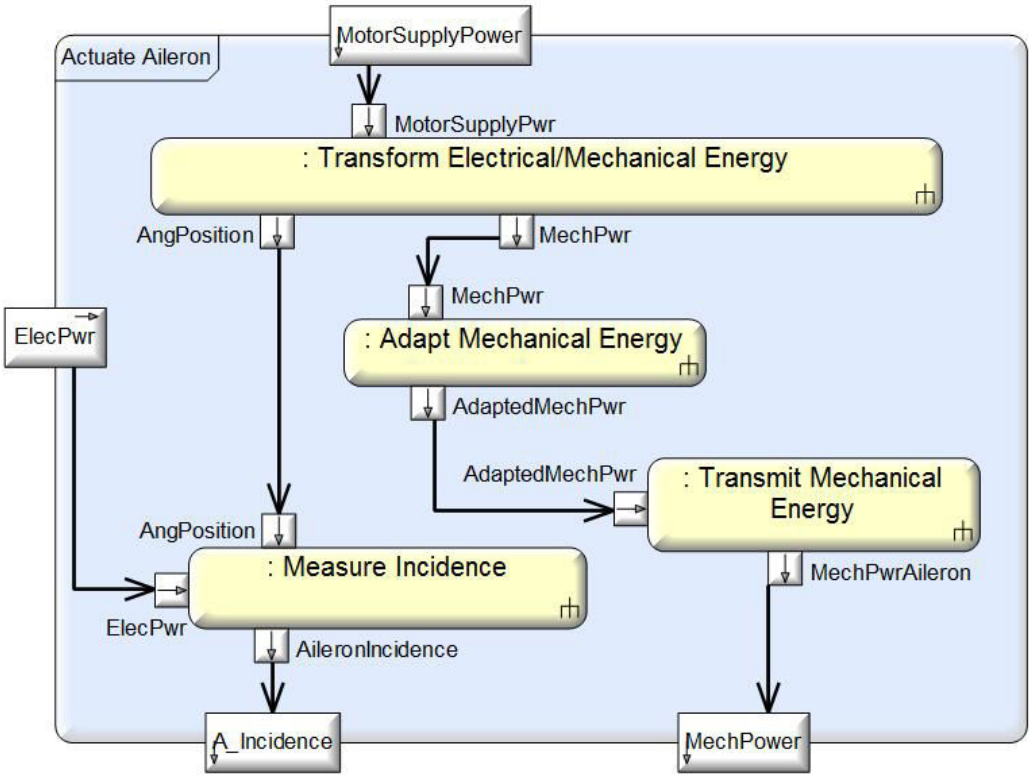


FIGURE A.10: Functional Decomposition of the 'Actuate Aileron' Function

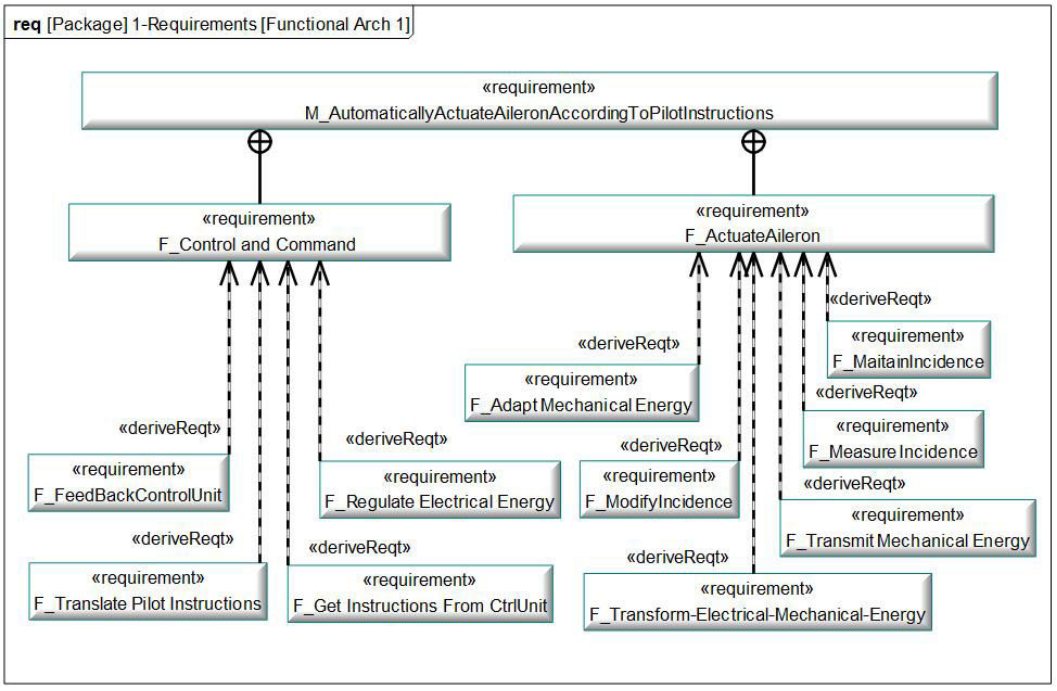


FIGURE A.11: Functional Requirements

A.4 Logical Architecture Definition

In this section, the logical architecture of the system is defined. Based on the functional architecture, components will be chosen to accomplish the different functions identified in the functional architecture. Each component will be allocated to the functions to achieve. The rule for the allocation is that a function can be allocated to only one component at once while a components may have more than one function to perform. The reason behind this is that if one function is allocated to several components it is hard to define the responsibility of each component towards the function (i.e. which parts of the function the component shall achieve) and thus it is hard to check if the function is fully satisfied. This rule does not apply in the case of redundant components that achieve the same function. When a function is performed by more than one component, then it should be decomposed further until its sub-functions are allocated unambiguously to components respecting the rule above. Note here that in all cases, only the leaf functions (the lowest level of the functional breakdown) of the functional hierarchy are allocated to components.

Several ways to allocate functions to components are possible and consequently, several logical architecture can be identified and compared. However, for seek of simplicity, we selected only one logical architecture for the EMA. To achieve the functions identified in the functional architecture definition (presented in section A.3), three components are chosen. The EMA is then composed of an "Embedded MCU with Power Bridge", a "Geared Motor with Encoder" and a "Mech Transmission". The Block Definition Diagram (BDD) in Figure A.12 shows the composition if the system as well as the functions allocated to each component. We can see that all the sub-functions of the *Control and Command* function are allocated to the component *Embedded MCU with PowerBridge* and it could be argued that the decomposition of this function is not useful, but, as mentioned above, several allocations are possible and several logical architectures can be determined and compared. In this work, the choice was to minimize the number of components for a seek of simplicity and because we already had in mind the extended modeling with interconnection components and multi-physical ports presented in Chapter 2.

The BDD (Figure A.12) shows the components of the system but does not show how these components communicate and interact together. This is detailed in the

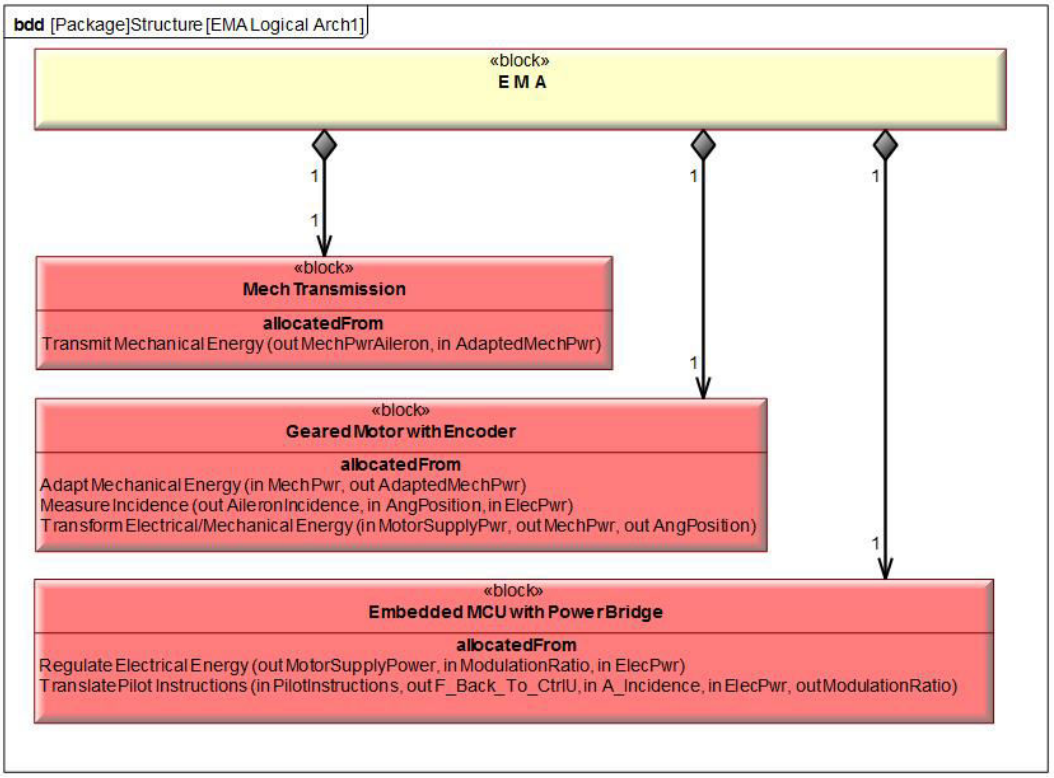


FIGURE A.12: EMA Logical Structure

next step of the methodology and given by an Internal Block Diagram (IBD). The internal structure of the EMA is given in Figure A.13.

Next to the safety analyses presented in this work, some redundant architectures of the EMA are presented in Figures A.14 and A.15.

These solution proposal can be compared.

A.5 Conclusion

This Appendix introduces one of the two case studies of this work. It illustrates the design methodology with SysML language without considering safety aspects. The Appendix details the different steps of the methodology on the example of EMA showing the ability of SysML to model different aspects of the system (requirements, structure and behavior) and also to maintain consistency between these different views of the system model.

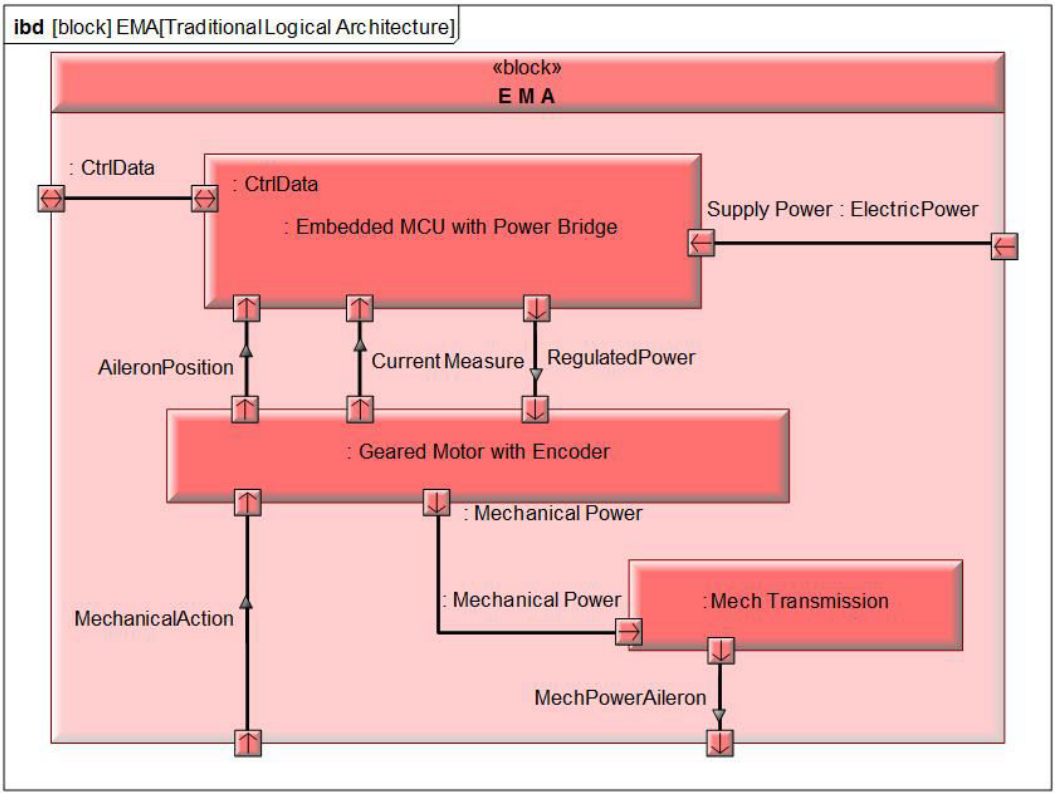


FIGURE A.13: EMA Logical Architecture

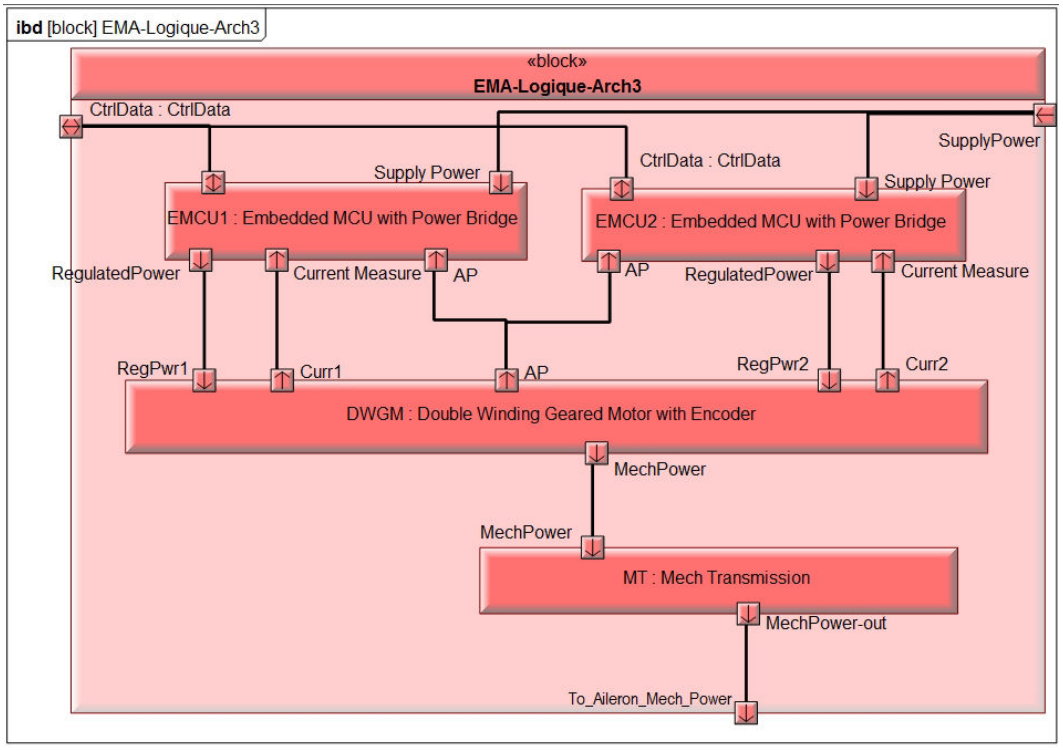


FIGURE A.14: EMA Logical Architecture with Redundancy - Proposal 1

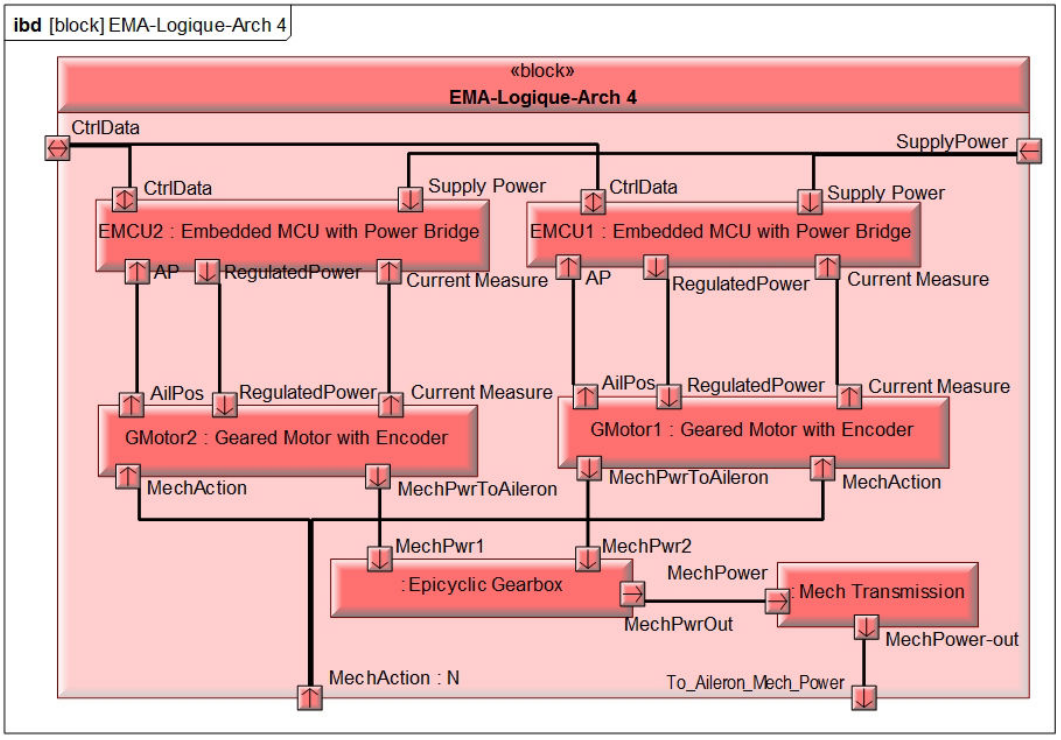


FIGURE A.15: EMA Logical Architecture with Redundancy - Proposal 2

Bibliography

[1] Nancy G. Leveson. *Engineering a Safer World, Systems Thinking Applied to Safety*. MIT PRESS, 2011.

[2] OMG UML Diagrams. URL http://www.uml-diagrams.org.

[3] Object Management Group (OMG) - Systems Modeling Language (SysML), . URL www.omgsysml.org.

[4] John Holt and Simon Perry. *SysML for Systems Engineering*. The Institution of Engineering and Technology, London, United Kingdom, 2008.

[5] Object Management Group (OMG), . URL http://www.omg.org/.

[6] Ian Moir and Allan Seabridge. *Aircraft Systems - Mechanical Electrical and Avionics Subsystems Integration*. Professionnal Engineering Publishing, second edition, 2001.

[7] Pierre David, Vincent Idasiak, and Frederic Kratz. Reliability study of complex physical systems using SysML. *Reliability Engineering and System Safety*, 95(4):431 – 450, 2010. ISSN 0951-8320.

[8] Verein Deutscher Ingenieure. Design methodology for mechatronic systems, June 2004. VDI 2206.

[9] Marco Bozzano and Adolfo Villafiorita. Integrating fault tree analysis with event ordering information. In *Safety and Reliability: Proceedings of the ESREL 2003 Conference*, Maastricht, the Netherlands, 15-18 June 2003.

[10] Nancy G. Leveson. Complexity and safety. In Daniel Krob Omar Hammami and Jean-Luc Voirin Editors, editors, *Complex Systems Design and Management, Proceeding of the Second International Conferance on Complex Systems Design and Management CSDM 2011*, pages 27–39, 2011.

[11] Society of Automotive Engineers SAE International. Guidelines for development of civil aircraft and systems, 2010. SAE-ARP-4754A.

[12] Society of Automotive Engineers SAE International. Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, 1996. SAE-ARP-4761.

[13] Yue Cao, Yusheng Liua, Hongri Fana, and Fanb. SysML-based uniform behavior modeling and automated mapping of design and simulation model for complex mechatronics. *Computer Aided Design*, 45(3):764–776, March 2013.

[14] US Department of Defense (DoD). Engineering management, May 1974. Mil-Std-499A.

[15] Howard Eisner. *Essentials of Project and Systems Engineering Management*. Wiley Publisher, 2002.

[16] Romaric Guillerm. *Intégration de la sûreté de fonctionnement dans les processus de l Ingénierie Système*. PhD thesis, Université de Toulouse, 2011.

[17] William B. Rouse. Engineering complex systems: Implications for research in systems engineering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 33(2):154 – 156, 2003.

[18] IEEE Computer Society. Standard for application and management of the systems engineering process, 1999. IEEE-1220.

[19] Electronic Industries Alliance (EIA). Processes for engineering a system, 1998. EIA-632.

[20] Florian Schneider and Brian Berenbach. A literature survey on international standards for systems requirements engineering. *Procedia Computer Science*, 16(0):796 – 805, 2013. ISSN 1877-0509. 2013 Conference on Systems Engineering Research.

[21] Sarah A. Sheard and Jerome G. Lake. Systems engineering standards and models compared. In *Proceedings of the Eighth International Symposium on Systems Engineering, Vancouver, Canada*, pages 589–605, 1998.

[22] Septavera Sharvia. *Integrated Application of Compositional and Behavioural Safety Analysis*. PhD thesis, University of Hull, February 2011.

[23] Clifton A. Ericson. *Hazard Analysis Techniques for System Safety.* John Wiley & sons, 2005.

[24] Ephraim Balz and Joachim Goll. Use case-based fault tree analysis of safety-related embedded systems. In *Proceedings Software Engineering and Applications*, 2005.

[25] Ningcong Xiao, Hong-Zhong Huang, Yanfeng Liand Liping He, and Tongdan Jin. Multiple failure modes analysis and weighted risk priority number evaluation in FMEA. *Engineering Failure Analysis*, 18:1162–1170, 2011.

[26] Nasa. *Fault Tree Handbook with Aerospace Applications.*, version 1.1 edition, 2002.

[27] Antoine Rauzy. *XFTA, an Open-PSA Fault Tree Engine*, 2012.

[28] Haiping Xu, Liudong Xing, and Ryan Robidoux. DRBD: Dynamic reliability block diagrams for system reliability modeling. *International Journal of Computers and Applications*, 31(2):202–, 2009.

[29] Marvin Rausand and Arnljot Hoyland. *SYSTEM RELIABILITY THEORY - Models, Statistical Methods, and Applications.* John Wiley & sons, 2008.

[30] Salvatore Distefano and Antonio Puliafito. Dynamic reliability block diagrams: Overview of a methodology. In *The 18th European Safety and Reliability Conference, ESREL*, volume 7, pages 1059–1068, 2007.

[31] The International Electrotechnical Commission (IEC). Functional safety of electrical/electronic/programmable electronic safety-related systems. parts 1 to 7, 1998. IEC 61508.

[32] *An Introduction to Functional Safety and IEC 61508.* MTL Instruments Group plc, March 2002. AN9025-3.

[33] INCOSE. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities.* Internationl Council of Systems Engineering, Cecilia Haskins edition, June 2006.

[34] Dennis M. Buede. *The Engineering Design of Systems: Models and Methods.* John Wiley & sons, 2009.

[35] Ian Sommerville. *Software Engineering.* Addison Wesley, 7 edition, 2004.

[36] Ian Sommerville. *Software Engineeing*. Addison Wesley, ninth edition edition, 2014.

[37] Philipp Helle. Automatic SysML-based safety analysis. In *Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems*, 2012.

[38] P. Bieber, C. Castel, and C. Seguin. Combination of fault tree analysis and model checking for safety assessment of complex system. In *Proceedings Fourth European Dependable Computing Conference (EDCC4). Toulouse (France)*, pages pp 19–31, October 2002.

[39] Marco Bozzano and Adolfo Villafiorita. Improving system reliability via model checking: The FSAP/NuSMV-SA safety analysis platform. In Stuart Anderson, Massimo Felici, and Bev Littlewood, editors, *Computer Safety, Reliability, and Security*, volume 2788 of *Lecture Notes in Computer Science*, pages 49–62. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20126-7.

[40] Anjali Joshi, Mats P. E. Heimdahl, Steven P. Miller, and Mike W. Whalen. Model-based safety analysis. Contractor report Cecilia Haskins, Nasa Langley Research Center, February 2006.

[41] Charlotte Seidner. *Vérification des EFFBDs : Model checking en Ingénierie Système*. PhD thesis, Université de Nantes, 2009.

[42] Gérald Point. *Alta-Rica: Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement*. PhD thesis, Université de Bordeaux I, 2000.

[43] A. Arnold, A. Griffault, G. Point, and A. Rauzy. The AltaRica language and its semantics. *Fundamenta Informaticae*, 34:109–124, 2000.

[44] Yiannis Papadopoulos and John A. McDermid. Hierarchically performed hazard origin and propagation studies. In *SAFECOMP 99, 18th International Conference on Computer Safety, Reliability and Security, Toulouse, France*, volume 1698 of *Lecture Notes in Computer Science*, pages 139–152. Springer Verlag, 1999.

[45] Régine Laleau, Farida Semmak, Abderrahman Matoussi, Dorian Petit, Ahmed Hammad, and Bruno Tatibouet. A first attempt to combine SysML

requirements diagrams and B. *Innovations in Systems and Software Engineering*, 6:47–54, 2010. ISSN 1614-5046.

[46] Hubert Dubois. Gestion des exigences de sûreté de fonctionnement dans une approche IDM. In *Journées Neptune N 5, Paris, France*, 08 avril 2008.

[47] Frédéric Thomas and Fabien Belmonte. Performing safety analyses and SysML designs conjointly : a viewpoint matter. In *Complex Systems Design & Management*, 2011.

[48] Fabien Belmonte and Elie Soubiran. A model based approach for safety analysis. In Frank Ortmeier and Peter Daniel, editors, *Computer Safety, Reliability, and Security*, volume 7613 of *Lecture Notes in Computer Science*, pages 50–63. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33674-4. doi: 10.1007/978-3-642-33675-1_5.

[49] Robin Cressent, Vincent Idasiak, and Frederic Kratz. Prise en compte des analyses de la sûreté de fonctionnement dans l'ingénierie de système dirigée par les modèles SysML. *Génie Logiciel*, pages 33–39, 2011.

[50] Pierre David. *Contribution à l'analyse de sûreté de fonctionnement des systèmes complexes en phase de conception: application à l'évaluation des missions d'un réseau de capteurs de présence humaine*. PhD thesis, Université d'Orléans, Novembre 2009.

[51] Alfredo Garro and Andrea Tundis. Enhancing the RAMSAS method for system reliability analysis - an exploitation in the automotive domain. In *SIMULTECH*, pages 328–333, 2012.

[52] Thomas A. Johnson, Jonathan M. Jobe, Christiaan J. J. Paredis, and Roger Burkhart. Modeling continuous system dynamics in SysML. In *ASME Conference Proceedings*, pages 197–205. ASME, 2007.

[53] Robin Cressent, Pierre David, Vincent Idasiak, and Frédéric Kratz. Apports de SysML à la modélisation des systèmes complexes à fortes contraintes de sûreté de fonctionnement. In *Technological Innovation and Transport Systems ITT09*, 2009.

[54] Eric Adrianarison and Jean-Denis Piques. SysML for embedded automotive systems - a practical approach. In *Embedded Real Time Software and Systems ERTS 2010*, 19th - 21st may 2010, Toulouse, France. 2010.

[55] What is UML, . URL http://www.omg.org/gettingstarted/what_is_uml.htm.

[56] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. ADDISON-WESLEY Longman, 1999.

[57] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. The Addison-Wesley object technology series. Pearson Education, 1999. ISBN 9788177583724. URL http://books.google.fr/books?id=a5J49FoFKq8C.

[58] www.omgmarte.org/, . URL www.omgmarte.org/.

[59] Madeleine Faugere. *MARTE: Also a UML profile for AADL*. Thalès Research & Technology, SAE AADL meeting Seattle edition, 2009.

[60] Muhammad Zohaib Iqbal, Shaukat Ali, Tao Yue, and Lionel Briand. Experiences of applying UML/MARTE on three industrial projects. In RobertB. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson, editors, *Model Driven Engineering Languages and Systems*, volume 7590 of *Lecture Notes in Computer Science*, pages 642–658. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33665-2. doi: 10.1007/978-3-642-33666-9_41.

[61] Shaukat Ali, LionelC. Briand, and Hadi Hemmati. Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. *Software & Systems Modeling*, 11(4):633–670, 2012. ISSN 1619-1366. doi: 10.1007/s10270-011-0206-z.

[62] Wilhelm Schäfer and Heike Wehrheim. Model-driven development with mechatronic UML. In Gregor Engels, Claus Lewerentz, Wilhelm Schäfer, Andy Schürr, and Bernhard Westfechtel, editors, *Graph Transformations and Model-driven Engineering*, pages 533–554. Springer-Verlag, Berlin, Heidelberg, 2010. ISBN 3-642-17321-7, 978-3-642-17321-9.

[63] UDPM-Group and OMG. Advancing MBSE via unified profile for DoDAF/-MODAF (UPDM).

[64] ISO/IEC/IEEE. Systems and software engineering - architecture description, 2011. ISO-IEC-IEEE-42010.

[65] OMG systems modeling language (OMG SysML ™), June 2012. URL http://www.omg.org/spec/SysML/1.3/.

[66] James E. Long. Relationships between common graphical representations used in system engineering. In *Proceedings of the SETE 2000, Systems Engineering and Test & Evaluation"in the Changing Environment of the 21st Century*, Brisbane, Queensland, 15-17 November 2000 2000.

[67] Michel Héon, Josianne Basque, and Gilbert Paquette. Validation de la sémantique d'un modèle semi-formel de connaissances avec OntoCASE. In *21èmes Journées Francophones d'Ingénierie des Connaissances*, Nîmes, France, 2010.

[68] Jean-François Pétin, Dominique Evrot, Gérard Morel, and Pascal Lamy. Combining SysML and formal models for safety requirements verification. In *22nd International Conference on Software & Systems Engineering and their Applications, Paris, France*, 2010.

[69] Nicolas Belloir, Jean-Michel Bruel, Natacha Hoang, and Cong-Duc Pham. Utilisation de SysML pour la modélisation des réseaux de capteurs. In *Actes de la conférence Langages et Modèles à Objets (LMO 08) Montreal, Canada*, 2008.

[70] Jean-Denis Piques and Eric Adrianarison. SysML for embedded automotive systems: lessons learned. In *Embedded real time Software and Systems ERTS*, 2012.

[71] Damien Chapon and Guillaume Bouchez. On the link between architectural description models and Modelica analyses models. In *Proceedings 7th Modelica Conference, Como, Italy, Sep. 20-22, 2009*, 2009.

[72] Tim Weilkiens. *Systems Engineering with SysML/UML Modeling, Analysis, Design.* Morgan Kaufmann Publishers, 2008.

[73] Stanford Friedenthal, Alan Moore, and Rick Steiner. *A practical Guide to SysML, The Systems Modeling Langage.* Morgan Kaufmann Publishers, 2009.

[74] Claudia Priesterjahn, Dominik Steenken, and Matthias Tichy. Component-based timed hazard analysis of self-healing systems. In *Proceedings of the 8th workshop on Assurances for self-adaptive systems*, ASAS '11, pages 34–43, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0853-3. doi: 10.1145/2024436.2024444.

[75] Donald Firesmith. Common requirements problems, their negative consequences, and the industry best practices to help solve them. *Journal of Object Technology*, 6(1), 2007.

[76] Aude Warniez, Olivia Penas, and Thierry Soriano. About metrics for integrated mechatronic system design. In *Mechatronics (MECATRONICS), 2012 9th France-Japan & 7th Europe-Asia Congress on and Research and Education in Mechatronics (REM), 2012 13th Int'l Workshop on Research and Education in Mechatronics, IEEE MECATRONICS REM, Paris, November 2012*, 2012. ISBN 978-1-4673-4772-3.

[77] Faida 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. ISSN 1474-0346. doi: http://dx.doi.org/10.1016/j.aei.2014.03.006.

[78] Object-Management-Group. XML metadata interchange (XMI) specification.

[79] Sheng-Hsein Teng and Shin-Yan Ho. Failure mode and effects analysis, an integrated approach for product design and process control. *International Journal of Quality and Reliability Management*, 13:8–26, 1996.

[80] Duane Kritzinger. *Aircraft System Safety: Military and Civil Aeronautical Applications*. Woodhead Publishing, June 2006.

[81] US Department of Defense (DoD). Procedure for performing a failure mode, effects and criticality analysis, November 1980. MIL-STD-1629A.

[82] US Department of Defense (DoD). Standard practice for system safety, February 2000. MIL-STD-882D.

[83] Pierre David, Vincent Idasiak, and Frédéric Kratz. Toward a better interaction between design and dependability analysis : FMEA derived from UML/SysML models. In *Proceedings of ESREL 2008 and 17th SRA-EUROPE annual conference*, 2008.

[84] Zoe Andrews and Richard Payne. Static fault analysis support - technical manual. Technical Report D33.3b, European Community, Seventh Framework Programme, 2014.

[85] Richard Banach and Marco Bozzano. Retrenchment, and the generation of fault trees for static, dynamic and cyclic systems. In Janusz Gorski, editor, *Computer Safety, Reliability, and Security. Proceeding of the 25th International Conference, SAFECOMP*, volume 4166 of *Lecture Notes in Computer Science*, pages 127–141. Springer Berlin Heidelberg, Gdansk, Poland, September 27-29 2006. ISBN 978-3-540-45762-6. doi: 10.1007/11875567_10. URL http://dx.doi.org/10.1007/11875567_10.

[86] F. Tajarrod and G. Latif-Shabgahi. A novel methodology for synthesis of fault trees from MATLAB-Simulink model. *World Academy of Science, Engineering and Technology*, 17(5):1256–1262, 2008.

[87] Nataliya Yakymets, Hadi Jaber, and Agnes Lanusse. Model-based system engineering for fault tree generation and analysis. In Slimane Hammoudi, Luís Ferreira Pires, Joaquim Filipe, and Rui César das Neves, editors, *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development, Barcelona, Spain, 19 - 21 February*, pages 210–214, 2013.

[88] Anjali Joshi, Pam Binns, and Steve Vestal. Automatic generation of static fault trees from AADL models. In *Proceedings of the IEEE / IFIP Conference on Dependable Systems and Networks Workshop on Dependable Systems, DSN07-WADS, Edinburgh, Scotland, UK, June*, 2007.

[89] Marco Bozzano and Adolfo Villafiorita. The FSAP/NuSMV-SA safety analysis platform. *International Journal on Software Tools for Technology Transfer*, 9(1):5–24, 2007. ISSN 1433-2779. doi: 10.1007/s10009-006-0001-2.

[90] Mykhaylo Nykolaychuk, Michael Lipaczewski, Tino Liebusch, and Frank Ortmeier. On efficiently specifying models for model checking. In Frank Ortmeier and Antoine Rauzy, editors, *Model-Based Safety Assessment, 4th International Symposium, IMBSA*. Springer, 2014.

[91] Michael Huth and Mark Ryan. *Logic in Computer Science : Modelling and Reasonning about Systems*. Cambridge University Press, 2004.

[92] Edmun M. Clarke, Jr. Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, 1999.

[93] Faida Mhenni, Nga Nguyen, Hubert Kadima, and Jean-Yves Choley. Safety Analysis Integration in a SysML-Based Complex System Design. In *IEEE International Systems Conference SysCon, Orlando, USA.*, 2013.

[94] Faida Mhenni, Nga Nguyen, and Jean-Yves Choley. Automatic fault tree generation from SysML system models. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, 2014.

[95] Faida Mhenni, Nga Nguyen, and Jean-Yves Choley. Towards the integration of safety analysis in a model-based system engineering approach with SysML. In *Design and Modeling of Mechanical Systems Proceedings of the Fifth International Conference Design and Modeling of Mechanical Systems, CMSM 2013, Djerba, Tunisia, March 25-27, 2013*, pages 61–68. SpringerVerlag Berlin Heidelberg, 2013.

[96] Faida Mhenni, Jean-Yves Choley, Alain Rivière, Nga Nguyen, and Hubert Kadima. SysML and safety analysis for mechatronic systems. In *Mechatronics (MECATRONICS), 2012 9th France-Japan & 7th Europe-Asia Congress on and Research and Education in Mechatronics (REM), 2012 13th Int'l Workshop on Research and Education in Mechatronics, IEEE MECATRONICS REM, Paris*, 2012. ISBN 978-1-4673-4772-3.

[97] Faida Mhenni, Jean-Yves Choley, and Nga Nguyen. Extended mechatronic systems architecture modeling with SysML for enhanced safety analysis. In *IEEE International Systems Conference SysCon, Ottawa, Canada*, 2014.

[98] Delphine Mami. *Définition, Conception et Expérimentation de structures d'actionneurs électromécaniques innovants incluant par conception des fonctionnalités de sûreté et de sécurité de fonctionnement.* PhD thesis, Université de Toulouse, Institut National de Polytechnique de Toulouse, 2010.

Component	Function	Function failure mode	Failure mode	Causal factors	Immediate effect (upstream, downsteam)		Method of detection	System effect	Recommended Actions	Severity	RPN	Current Controls
Embedded MCU with Power Bridge	Regulate Electrical Energy	Fails to perform	hardware defect	manufacturing fault, overheating, current/voltage overload, EMC, vibration (bondings welding break)		Motor stops or runs randomly	current monitoring	Aileron locked in a position or out of control	disengage aileron	Catastrophic		
		Performs incorrectly (degraded performance)	hardware defect	manufacturing fault, overheat, current/voltage overload, EMC, vibration (bondings welding break)		inappropriate supply of the motor	current monitoring	degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		
	Translate Pilot Instructions	Fails to perform	software fault/error	bad specification, bad implementation		wrong PWM data sent to power electronics	watch dog	Aileron locked in a position or out of control	disengage aileron	Catastrophic		
			hardware defect	manufacturing fault, overheat		wrong PWM data sent to power electronics	current monitoring	Aileron locked in a position or out of control	disengage aileron	Catastrophic		
			synchronization error	clock defect, timer error, EMC…		wrong PWM data sent to power electronics	watch dog	Aileron locked in a position or out of control	disengage aileron	Catastrophic		
			Memory defect/loss	EMC, manufacturing error, aging of electronic components (memory)		wrong PWM data sent to power electronics	watch dog	Aileron locked in a position or out of control	disengage aileron	Catastrophic		
		Performs incorrectly (degraded performance)	software fault/error	bad specification, bad implementation		wrong PWM data sent to power electronics	watch dog	degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		
			synchronization error	clock defect, timer error, EMC, PWM error…		wrong PWM data sent to power electronics	watch dog	degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		

Geared Motor with Encooder	Adapt Mechanical Energy	Fails to perform	Loss of structural integrity	material defect		stall or free output	motor current measure, position and speed measure (current nil and chaotic angle)	Aileron locked or free mouvement of the aileron	disengage aileron	Catastrophic		
				overload								
				manufacturing defect								
				vibration								
			jamming	material defect		stall	motor current measure, position and speed measure (current peak and fixed angle)	Aileron locked	disengage aileron	Catastrophic		
				manufacturing defect								
				thermal dilation								
				lubrication defect								
		Performs incorrectly (degraded performance)	Poor efficiency	backlash		slow motion, vibration,	response delay and motor current monitoring	degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		
				wear, ageing								
				lubrication fault								
				vibration								
	Measure Incidence	Fails to perform	mechanical drive defect	vibration		No available angular data	Monitoring of aileron motion	Aileron out of control	disengage aileron	Catastrophic		
				unscrewed								
			Power loss	wire or welding Ruprure								
			internal component failure	manufacturing fault/ component breakdown								
		Performs incorrectly (degraded performance)	mechanical drive defect	vibration		Non accurate angular data	Monitoring of aileron motion	degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		
				assembly or manufacturing default								
			low or high voltage									
			measure imperfection	accuracy, repeatability, fidelity, sensitivity								
			measure fault	interruption signal émis (permanent ou momentané)								
	Transform Electrical - Mechanical Energy	Fails to perform	Jamming (stall)	material defect		mechanism locked	current and rotation monitoring	Aileron locked	disengage aileron	Catastrophic		
				manufacturing defect								
				thermal dilation								
				lubrication defect								
			Loss of structural integrity	material defect		mechanism locked or free motion	current and rotation monitoring	Aileron locked or free motion	disengage aileron	Catastrophic		
				overload								
				manufacturing defect								
				vibration								
			short-circuit between two windings	insulation degraded due to overheat		mechanism locked or free motion	current and rotation monitoring	Aileron locked or free motion	disengage aileron	catastrophic		
				manufacturing defect								
		Performs incorrectly (degraded performance)	short-circuit in one winding	insulation degraded because of overheat		underpowered mechanism	current and rotation monitoring	degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		
				manufacturing defect		underpowered mechanism	current and rotation monitoring	degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		

Mech Transmission	Transmit Mechanical Energy	Fails to perform	Loss of structural integrity	material defect overload manufacturing defect vibration		Stall or free output	motor current measure, position and speed measure (current nil and chaotic angle)	Aileron locked or free mouvement of the aileron	disengage aileron	Catastrophic		
			jamming	material defect manufacturing defect thermal dilation lubrication defect		stall	motor current measure, position and speed measure (current peak and fixed angle)	Aileron locked	disengage aileron	Catastrophic		
		Performs incorrectly (degraded performance)	Poor efficiency	backlash wear, ageing lubrication fault vibration		slow motion, vibration,	response delay and motor current monitoring	degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		
Data Bus		Performs incorrectly (degraded performance)	corrupted data	EMC (cross-talk)		corrupted data received by MCU	Parity checks	inappropriate reponse of aileron	adjust instruction or disengage aileron	Catastrophic		
		Fails to perform	no data	short-circuit Loose connection (misconnect or disconnec) Broken PCB or wiring or weld		No data received by MCU		inappropriate reponse of aileron	disengage aileron	Catastrophic		
Feedback Bus		Performs incorrectly (degraded performance)	corrupted data	EMC (cross-talk)		corrupted data received by MCU	Parity checks	inappropriate reponse of aileron	adjust instruction or disengage aileron	Catastrophic		
		Fails to perform	no data	short-circuit Loose connection (misconnect or disconnec) Broken PCB or wiring or weld		No data received by MCU		inappropriate reponse of aileron	disengage aileron	Catastrophic		
Power Bus		Performs incorrectly (degraded performance)	inappropriate voltage or current	EMC (cross-talk)		Motor not correctly powered	Current and voltage monitoring	inappropriate reponse of aileron	adjust instruction or disengage aileron	Catastrophic		
		Fails to perform	Sparking Loose connection (misconnect or disconnect) Broken PCB or wiring or weld	short-circuit mechanical defect, vibration manufacturing defect, vibration		Motor not powered		inappropriate reponse of aileron	disengage aileron	Catastrophic		

Power supply harness	Induced interconnection functions	Performs incorrectly (degraded performance)	inappropriate voltage or current	EMC (cross-talk)		EMA not correctly powered	Current and voltage monitoring	inappropriate reponse of aileron	adjust instruction or disengage aileron	Catastrophic		
		Fails to perform	Sparking	short-circuit		EMA not powered		inappropriate reponse of aileron	disengage aileron	Catastrophic		
			Loose connection (misconnect or disconnect)	mechanical defect, vibration								
			Broken PCB or wiring or weld	manufacturing defect, vibration								
Aileron mechanical coupling		Fails to perform	Jamming	material defect		EMA output locked	Monitoring of aileron motion	Aileron locked	disengage aileron	Catastrophic		
				manufacturing defect								
				thermal dilation								
				lubrication defect								
			Loss of structural integrity	material defect		EMA free output or locked		Aileron locked or free mouvement of the aileron	disengage aileron	Catastrophic		
				overload								
				manufacturing defect								
				vibration								
		Performs incorrectly (degraded performance)	Poor efficiency	Lubrication defect		slow motion, vibration		degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		
				ageing								
Wing mechanical coupling		Fails to perform	Loss of structural integrity	material defect		EMA not fixed	Monitoring of aileron motion	Aileron out of control	disengage aileron	Catastrophic		
				overload								
				manufacturing defect								
				vibration								
		Performs incorrectly (degraded performance)	Faulty fixing	vibration		EMA faulty fixing		degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		
				ageing								

Transmission mechanical coupling		Fails to perform	Jamming	material defect		EMA output locked	Monitoring of aileron motion	Aileron locked	disengage aileron	Catastrophic		
				manufacturing defect								
				thermal dilation								
				lubrication defect								
			Loss of structural integrity	material defect		EMA free output or locked		Aileron locked or free mouvement of the aileron	disengage aileron	Catastrophic		
				overload								
				manufacturing defect								
				vibration								
		Performs incorrectly (degraded performance)	Poor efficiency	Lubrication defect		slow motion, vibration		degraded motion of the aileron (performance)	adjust instruction or disengage aileron	Critical		
				ageing								

THESE DE DOCTORAT DE L'ECOLE CENTRALE PARIS PRESENTEE PAR

Faïda MHENNI

Vers une Approche Intégrée d'Analyse de Sûreté de Fonctionnement des Systèmes Mécatroniques

Résumé :
Les systèmes modernes sont caractérisés par l'intégration de plusieurs composants de technologies diverses interagissant dans le but d'offrir de plus en plus de fonctionnalités aux utilisateurs. La complexité croissante dans ces systèmes pluridisciplinaires dits mécatroniques nécessite la mise en place de nouveaux processus, outils et méthodes pour la conception, l'analyse et la validation de ces derniers en respectant les contraintes de coût et de délais imposés par la concurrence. Ces systèmes doivent également satisfaire des contraintes de fiabilité et surtout de sûreté de fonctionnement. Seule une intégration du processus d'analyse de sûreté de fonctionnement tout au long du processus de développement peut assurer la satisfaction de ces contraintes de manière optimale.

Les travaux de cette thèse ont pour objectif de contribuer à l'intégration des analyses de sûreté de fonctionnement dans le processus d'ingénierie système basée sur SysML afin de rendre ces analyses plus rapides et plus efficaces. Pour ce faire, nous avons traité les axes suivants : la formalisation d'une méthodologie de conception basée sur SysML et qui sera le support des analyses de sûreté de fonctionnement ; l'extension du langage SysML afin de pouvoir intégrer des spécificités des systèmes mécatroniques ainsi que des aspects de sûreté de fonctionnement dans le modèle système; l'exploration automatique des modèles SysML afin d'en extraire les données nécessaires pour l'élaboration des artefacts de la SdF et la génération (semi)/automatique de ces derniers (FMEA et FTA). Nous avons également intégré la vérification formelle d'exigences de sûreté de fonctionnement.
Cette méthodologie nommée SafeSysE a été appliquée sur des cas d'étude du domaine de l'aéronautique : EMA (Electro-Mechenical Actuator) et WBS (Wheel Brake System).

Mots clés: Ingénierie système, Sûreté de fonctionnement, Systèmes mécatroniques**,** MBSE, MBSA, SysML, FMEA, FTA

Safety Analysis Integration in a Systems Engineering Approach for Mechatronic Systems Design

Abstract:
Modern systems are getting more complex due to the integration of several interacting components with different technologies in order to offer more functionality to the final user. The increasing complexity in these multi-disciplinary systems, called mechatronic systems, requires new appropriate processes, tools and methodologies for their design, analysis and validation whilst remaining competitive with regards to cost and time-to-market constraints.

The main objective of this thesis is to contribute to the integration of safety analysis in a SysML-based systems engineering approach in order to make it more efficient and faster. To achieve this purpose, we tackled the following axes: formalizing a SysML-based design methodology that will be the support for safety analyses; providing an extension of SysML in order to enable the integration of specific needs for mechatronic systems modeling as well as safety concepts in the system model; allowing the automated exploration of the SysML models in order to extract necessary information to elaborate safety artefacts (such as FMEA and FTA) and the semi-automated generation of the latters. We have also integrated formal verification to verify if the system behaviors satisfy some safety requirements.
The proposed methodology named SafeSysE was applied to case studies from the aeronautics domain: EMA (Electro Mechanical Actuator) and WBS (Wheel Brake System).

Keywords: Systems Engineering, Safety analysis, Mechatronic systems, MBSE, MBSA, SysML, FMEA, FTA