SafeSysE: A Safety Analysis Integration in Systems Engineering Approach

Faïda Mhenni, Nga Nguyen, and Jean-Yves Choley

***Abstract*—The main objective of this paper is the integration of safety analysis in a SysML-based systems engineering approach in order to make it more effective and efficient. It helps to ensure the consistency between safety analyses and system design and then to avoid late errors and to reduce system development time. To achieve this purpose, we tackled the following axes: 1) formalizing a SysML-based design methodology that will be the support for safety analyses; 2) providing an extension of SysML to enable the integration of specific needs for safety concepts in the system model; and 3) performing an automated exploration of the SysML models to generate necessary information to elaborate safety artifacts such as failure mode and effects analysis (FMEA) and fault tree analysis (FTA). The proposed methodology named safety integration in systems engineering (SafeSysE) is applied to a real case study from the aeronautics domain: electromechanical actuator (EMA).**

***Index Terms*—Failure mode and effects analysis (FMEA), fault tree analysis (FTA), model-based safety analysis (MBSA), model-based systems engineering (MBSE), model checking, safety analysis, systems engineering.**

I. Introduction

MODERN systems are getting more complex due to the integration of several interacting components with different technologies to offer more functionality to the final user. Accidents can result from unsafe interactions between nonfailed components or software-related errors [1]. The complexity in these systems requires new appropriate processes, tools, and methodologies for their design, analysis, and validation while remaining competitive with regard to cost and time-to-market constraints. Model-based systems engineering (MBSE) [2] is a systems engineering approach that explores the use of models, which are more expressive and less ambiguous than documents. Model simulation also offers an easier way to perform tradeoffs and comparisons 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. In the MBSE approach, different languages and modeling tools can be used according to the domains involved in the system, the level of detail, the system aspects to be modeled, etc. SysML [3] is a systems modeling language that aims at providing a unified standard for "specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities." It also allows a multiview model and building traceability links. This relatively recent language is already widely used in both industrial and academic worlds like in [4]–[11] to cite only few works.

Safety analysis has the objective to assess system safety during design phase and ensure that the designed systems have satisfactory safety level. Different techniques and methods exist and are used for different purposes and at different design levels [12]. The two most traditionally used techniques are failure mode and effects' analysis (FMEA) and fault tree analysis (FTA) [12], [13]. 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 that identifies the failure modes of system functions or components and then determines their effects on the system level. 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. 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. These safety 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. To be efficient and correctly explored by system designers, safety analyses must be performed in the early phase of design. That will help 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, errorlessly. To respond to these requirements, safety analyses should be integrated into the design process. To reduce error proneness and development time, the generation of safety artifacts via model-to-model transformation approach (automating the building and modification of models) is needed.

In this paper, we propose a methodology of safety integration in systems engineering approach named SafeSysE. This methodology, we believe, contributes to solving the problems mentioned above. Based on a SysML system model, the methodology provides to all the contributors, a common reference system model that takes into account the constraints of

F. Mhenni and J.-Y. Choley are with Quartz, SUPMECA, Paris 93400, France (e-mail: faida.mhenni@supmeca.fr; jean-yves.choley@supmeca.fr).

N. Nguyen is with Quartz, EISTI, Cergy 95000, France (e-mail: nga.nguyen@eisti.eu).

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. Based on this model, the methodology automates some steps of the safety artifacts generation via model exploration and model-to-model transformation. One important point of our approach with respect to other methods in the literature is that the design data and safety data are kept in the same SysML model, via an integrated safety profile. Since safety artifacts can be either constructed from structural models as well as from the dynamic behavioral models of the system, a safety analysis can be qualified as compositional or behavioral, respectively [14]. Our approach studies the two well-known compositional safety techniques, namely FMEA and FTA, recommended by safety standards such as IEC 61508 [15], and SAE ARP 4754 and 4761 [16], [17]. The general IEC 61508 concerns all systems based on electric, electronics, and programmable electronics, and SAE ARP 4754 and 4761 are civil aircraft and systems standards. SafeSysE addresses also a behavioral safety analysis (BSA) based on model checking [18], a formal verification method that allows validating if the system dynamic behavior satisfies some safety requirements written in temporal logic. These interrelated analysis techniques are used by SafeSysE to perform safety analyses in a very complementary way, during the whole system design process. However, as the BSA is not in the scope of the paper, the readers can reference to our other publications [19], [20] for more details.

This paper is organized as follows. Section II represents related work about the integration of systems engineering and safety analysis. Section III details the SafeSysE methodology. It also introduces the safety profile, a SysML extension which allows integrating safety-relevant properties in the system model to facilitate the automatic generation of safety artifacts. The implemented tool supporting SafeSysE is also described in this section. Section IV explains the automated generation of the functional and the component FMEA from SysML models. The fault tree generation algorithm, including the pattern identification phase as well as the graph traversal phase, is given in Section V. The same case study, i.e., electromechanical actuator (EMA), is used throughout Sections IV and V to illustrate the different steps in SafeSysE. Finally, this paper is concluded in Section VI.

II. Related work

This section discusses related work concerning the integration of safety analysis into MBSE, called model-based safety analysis (MBSA). In order to compare them with SafeSysE, we will focus on publications using SysML as systems modeling language as well as literature on automating FMEA and fault tree generation.

Laleau *et al.* [21] tried to combine SysML requirement diagrams and the B formal specification language. Since requirements in SysML are textual, the SysML requirement models are first 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, Albinet *et al.* [22] proposed to directly include system requirements in the design process but the separation with the proposed solutions as required by safety standards such as ISO 26262 [23] is achieved by isolating the following triplet: requirement models, solution models, and validation and verification models.[1] A SysML profile called requirement profile for MeMVaTEX (RPM) has been developed in [22]. 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 V&V models using these properties. These V&V models have also been explored in the work of Guillerm *et al.* [25].

Another approach to integrate SysML and safety analysis is the use of the common modeling Eclipse framework [26]. In this work, an independent tool called Obeo designer safety viewpoint that implements classical risk analyses is developed. Then, 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. So, in this approach, SysML is not extended with a safety profile to tune the SysML models. In a more recent work [27], a translation from Obeo designer's domain-specific language for FMEA and preliminary hazard analysis (PHA) into AltaRica [28] is added to enable formal verification. However, no real system has been studied yet to prove the scalability of the method.

Helle [11] presented an integration process of MBSA in a SysML-based MBSE. In this work, an extension of 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 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 reliability block diagrams [29], [30] representing this cut set.

[1] ISO 26262, an adaptation of the Functional Safety Standard IEC 61508 for automotive electric/electronic systems imposes a clear distinction between the concepts: the solution has to be developed independently with respect to the requirements as well as to the verification and validation (V&V) part. The separation is important because from the given requirements, various solutions can be defined. Also, as cited in [24], the developed solutions must be evaluated by actors independently of the design process, which will promote a diversity of analysis while increasing the coverage and confidence levels of the safety conclusions. Of course, this is not in contradiction with an integrated framework where the traceability between the solutions and the requirements as well as the safety analysis will be respected.

Garro and Tundis [31] 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 though simulation. A formal verification method was not used in this research for safety assessment.

Tajarrod and Latif-Shabgahi [32] described fault trees' construction from MATLAB Simulink models. In this work, the nominal model is built in Simulink and then manually extended with failure behavioral information of the system. Based on this extended model and the classification of components, the fault tree for a specific top event is automatically constructed.

An automatic generation of fault trees from architecture analysis and design language (AADL) models is proposed in [33]. In this work, the system architectural model is built with the AADL language and then annotated with fault and failure information using the error annex, a sublanguage of AADL. Based on the annotated model, fault trees are automatically generated in the commercial tool CAFTA.

FSAP/NuSMV-SA [34] 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, i.e., in the presence of faults. The safety analysis engine based on the NuSMV model checker can be used to produce FTA. 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 system and consequently, exploring the fault tree to understand the fault propagation through the system components is not very intuitive for systems engineers.

In her thesis [14], Sharvia dealt with the integration of the compositional safety analysis (CSA) and the BSA. The first part is carried out with hierarchically performed hazards' origin and propagation studies (HiP-HOPS), a safety analysis technique presented in [35]. In this part, system failure models such as FTA and FMEA 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 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.

David *et al.* [8] worked on the generation of an FMEA report from system functional behaviors written in SysML models and on the construction of dysfunctional models using the AltaRica language 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 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.

Yakymets *et al.* [36] presented a safety modeling framework for fault tree generation SMF-FTA. This framework includes metamodels, profiles, model transformation, verification, and FTA tools. 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. 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 fault tree. The resulting fault tree can be represented either with open-probabilistic safety assessment (PSA) or a SysML dedicated profile.

In our work, the safety artifacts including FMEA, FTA, and NuSMV programs are extracted directly from SysML system models, not using an external language such as AltaRica. In this way, we reduce the model-to-model transformations and thus the possibility of data losses during successive transformations. We also update the system model directly with the results of different safety analyses via the integrated safety profile: safety data and design data are stored in the same model. The work of David *et al.* [8] requires accessing an external database to import dysfunctional behavior. In addition, SafeSysE takes into account both compositional and behavioral aspects of safety analyses. The result of one analysis is used in the next step to refine the other, assuring the consistency of the methodology. A proof-of-concept tool has also been implemented to illustrate SafeSysE.

III. SafeSysE Methodology

This section begins with a description of the SafeSysE process. The proposed safety profile, which is a part of the methodology content, is discussed in Section III-B. The implementation details of the SafeSysE tool are presented in Section III-C.

A. SafeSysE Integrated Process

In this section, the integrated process of systems engineering and safety analysis is presented through a set of steps. In Fig. 1, an AD is used to describe the process. Swim lanes are used to make a distinction between systems engineering and safety analysis activities (or processes). SafeSysE starts with a

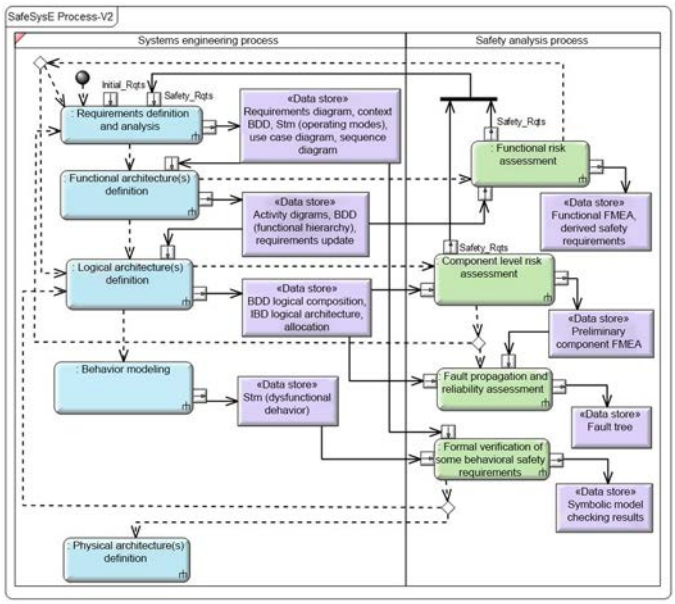

Fig. 1. SafeSysE integrated process.

requirements' definition and analysis process with, as a starting point, a set of initial requirements describing the need. Data stores are used to model the storage of the different artifacts issued from each activity.

1) Step 1: Requirements Definition and Analysis: In this step, system functionalities as well as its external interfaces are described by a set of requirements. Several SysML diagrams such as use case diagrams and BDDs for the system context can be used to help in the identification of these requirements. Since the requirement definition and analysis are not in the main scope of this paper, refer to our paper about SysML-based systems engineering methodology in [37] for more detail.

2) Step 2: Functional Architecture Definition: Based on the functional requirements identified in Step 1), one or more functional architectures are proposed during this step. The final result is a hierarchical model of the breakdown of the system main function(s) into subfunctions. In SysML, functions are represented by activities, and the functional breakdown is modeled through a set of ADs, each AD representing the breakdown of a given function (activity) into subfunctions. ADs also show the progressive transformation of input flows into output flows.

3) Step 3: Functional Risk Assessment: In this step, a functional FMEA is used to identify potential hazards caused by failures and their effects. The automatically generated FMEA data sheet contains the list of functions and a list of generic failure modes. The safety expert then performs the analysis and completes the FMEA with the relevant data. All this new safety information is then updated into the SysML model via the safety profile extension (Section III-B). The gap between safety analysis and design modification is shortened, 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 such as diagnosis and reconfiguration functions. 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.

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 built by allocating components to functions. A BDD describes the components of the system and an 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).

5) Step 5: Component Risk Assessment: When the structure of the system is defined, the safety analysis results are updated and a component-level risk assessment is performed. 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 at a nonacceptable 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. This step, as well as Step 3), is explained in more detail in Section IV.

6) Step 6: Fault Propagation and Reliability Assessment: 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 FTA tools. For more details about fault tree generation refer to Section V.

7) Step 7: Behavioral Safety Analysis: This final step is carried out to complete the safety analyses. 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 explored to generate automatically a NuSMV 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. Since this step is not in the scope of this paper, the readers can reference to our other publication [20] for more details.

B. 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. And in the opposite direction, it is

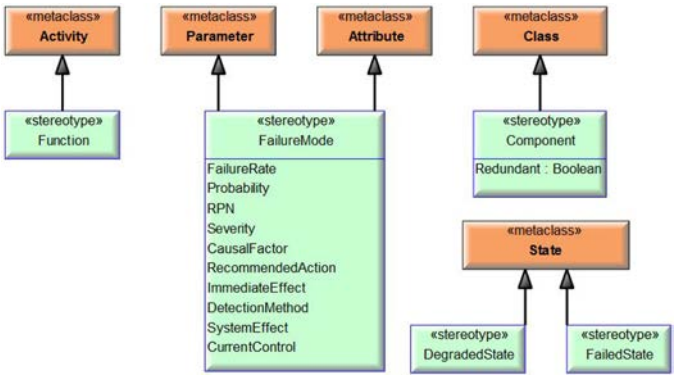

Fig. 2. Safety profile diagram.

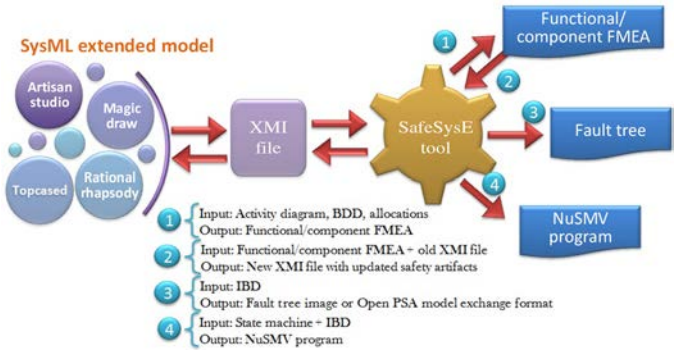

Fig. 3. SafeSysE tool.

important for a safety expert to feedback safety analysis results to systems engineers to take them into account in the system design. In order to integrate safety information directly into SysML models, we have explored the extension mechanism of UML to create a so-called safety profile. A profile allows adaptation or customization of UML metamodels to a specific platform, domain, or method through stereotype and tag definition concepts. 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 BSA.

Since in our methodology, a system function is represented by an activity, it is straightforward to consider function as a stereotype extending the activity metaclass. A system component is a SysML block, so the component stereotype will extend the class metaclass of UML. Because each activity may have several parameters and each class may have several attributes, Parameter and attribute are chosen as extended metaclasses for FailureMode stereotype. By doing so, we could 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, and detection methods can be simply considered as the tag definitions of the FailureMode stereotype. Fig. 2 gives the profile diagram of our safety profile. This diagram models also a simple redundancy mechanism as well as dysfunctional behavior information such as degraded and failed states. It is also noted that there is no unique solution for the safety profile. A simple and efficient solution that allows us to represent all needed information while not overloading the XML metadata interchange (XMI) file generated from the SysML model is preferred.

C. SafeSysE Tool Implementation

To support and validate the methodology for real-scale applications, we have developed the SafeSysE tool to automatically generate different safety artifacts. This proof-of-concept tool is written in the Python language. For the automatic generation, first the SysML model is exported into an XMI file [38]. 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 modeling tool, so our program will be 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. Data are organized in a tree structure. Our SafeSysE tool parses the XMI file using *Beautiful Soup*, a Python library that provides methods to navigate, search, and modify a parse tree. For each function, the tool builds a graph with nodes and edges containing all information needed for the generation of the corresponding safety artifacts. Fig. 3 shows the program's main functionalities with the corresponding input and output data. Further algorithmic details will be given in the following sections for each functionality.

IV. FMEA Generation From SysML Models

FMEA is a reliability tool widely used in safety analysis. To take full benefit of FMEA, it is critical to conduct it at early design stages and continue concurrently with the design evolution, in order to reduce development time and error proneness. In our work, the preliminary FMEA is automatically generated from the system model, and the consistency with the system model is then ensured. Information added by safety experts in the generated FMEA then will be updated in SysML models via the safety profile. Concretely, our SafeSysE tool uses the updated FMEA worksheet and the current XMI file corresponding to the system model as input, and outputs a new XMI file containing updated safety information in the appropriate stereotypes and tag definitions.

A. Functional FMEA

As described in Step 2) of SafeSysE, the functional breakdown of the system mission into subfunctions is modeled with ADs. 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. 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 the failure modes of the higher level functions are the result of the failure modes of the lower level ones. In an AD, 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

not only is very important to understand the system functioning but it is also critical for safety analysis.

When using SafeSysE tool, the user can choose to generate a functional FMEA for a particular package that represents a specific functional architecture solution. The functional FMEA generation algorithm realizes the following steps.

1) AD Extraction: From the node corresponding to the chosen package in the parsed tree, the tool extracts all the information related to ADs 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 prefilled columns are described hereafter.

a) *Function*: Functions corresponding to activities are found from the graph built from the previous step. 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.
b) *Function failure mode*: The list of generic failure modes available in a functional FMEA is saved in a configuration file containing: "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 is used to fill in automatically the failure mode cells of each function. Other specific failure modes of the system functions will be added later by safety experts directly in the FMEA worksheet.
c) *Causal factors*: They are the input and output parameters of the current activity/function. These data are not real information in an FMEA analysis, but they help to be exhaustive in finding all possible causes of failures.
d) *Immediate effects*: As the causal factors, we prefill 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. He can add new failure modes that have not been considered if any, or remove irrelevant failure modes. 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 a 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

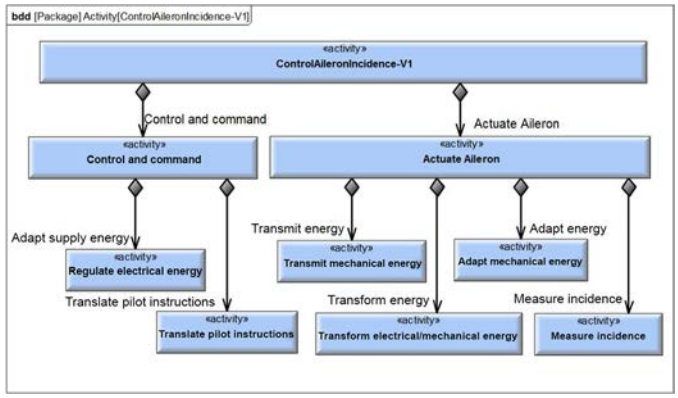

Fig. 4. Functional architecture of the EMA.

can also fail because of the failure or degraded operating of the downstream functions. The safety expert has to analyze each failure mode and determine 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 to identify critical functions and prioritize the list of corrective actions.

B. Component FMEA

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, 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 of component FMEA, 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, thanks to the safety profile. The elaboration of a component FMEA is quite similar to the functional FMEA. The only difference is that instead of activity nodes, the graph is built from block data extracted from the parsed tree. In the XLS file generation step, one more column for the component name is added. If n functions are allocated for this component, there will be in different lines which have the same format as the functional FMEA corresponding to this component.

C. Case Study

In this paper, the EMA example [39] is used to illustrate SafeSysE. The use of EMAs in flight control is increasing

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			

Fig. 5. 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 failure; 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 reliability; preventive diagnostics	catastrophic
	Performs incorrectly (degraded performance)	*input: AngPosition, ElecPwr* *output: A_Incidence*	Internal failure; 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 reliability; preventive diagnostics	catastrophic
	Operates inadvertently	*input: AngPosition, ElecPwr* *output: A_Incidence*	Internal failure; 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 reliability; preventive diagnostics	catastrophic
	Operates at incorrect time (early, late)	*input: AngPosition, ElecPwr* *output: A_Incidence*	Internal failure; 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 reliability; 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 reliability; preventive diagnostics	catastrophic
	Sends erroneous data	*input: AngPosition, ElecPwr* *output: A_Incidence*	Internal failure; faulty power supply	*upstream: Transform Electrical/Mechanical Energy* *downstream: Translate Pilot Instructions*	Faulty measurement provided to "Translate Pilot Instructions"	Aileron out of control	Appropriate reliability; preventive diagnostics	catastrophic

Fig. 6. Functional FMEA completed by the safety expert.

since they have many advantages, such as better environmental respect, weight saving, maintenance cost reduction, performance increase, and speed accuracy. An EMA is mainly made of three parts: an electric motor, a mechanical transmission, and an electronic and software part composed of a calculator that controls the system. After the requirements' definition and analysis process, the functional analysis of the EMA is performed. A functional breakdown is represented with nested ADs in SysML. The resulting functional architecture of the EMA is given in a BDD in Fig. 4.

By using SafeSysE tool and the flows exchanged between functions in corresponding ADs (to avoid redundancy, these diagrams will be given later, only for the updated functional architecture), the preliminary FMEA automatically generated for leaf functions in Fig. 4 is given in Fig. 5. The automatic generation will ensure exhaustiveness of failure modes, causal factors, and effects of failures. To save space, we only show an extract of FMEA, i.e., for function "Measure Incidence." Based on this preliminary FMEA and on a good understanding of the system functioning, the safety expert performs the analysis to generate the complete FMEA (Fig. 6). Then, our SafeSysE tool will update the system model with the new FMEA information. This allows to regenerate new FMEA when needed (when design changes occurred) without losing the safety expert's work. As seen in the FMEA in Fig. 6, 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. In this case, we decided to add a new function "Internal Diagnosis" to the system. This function collects measures of some critical parameters of the other

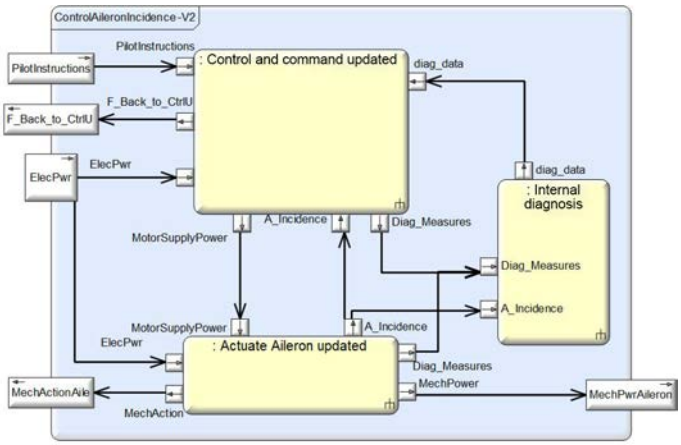

Fig. 7. Updated AD for Control Aileron Incidence.

functions and tries to identify potential faults in the system. 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. The new added function implies modification of the other functions since they have to provide this function with monitoring data. The updated ADs of the top-level function "Control Aileron Incidence" and the function "Actuate Aileron Updated" (as an example) are given in Figs. 7 and 8, respectively. 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.

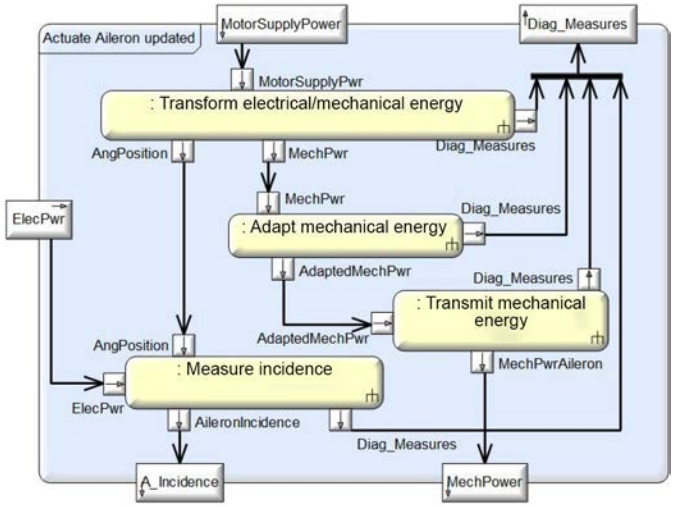

Fig. 8. Updated AD for Actuate Aileron.

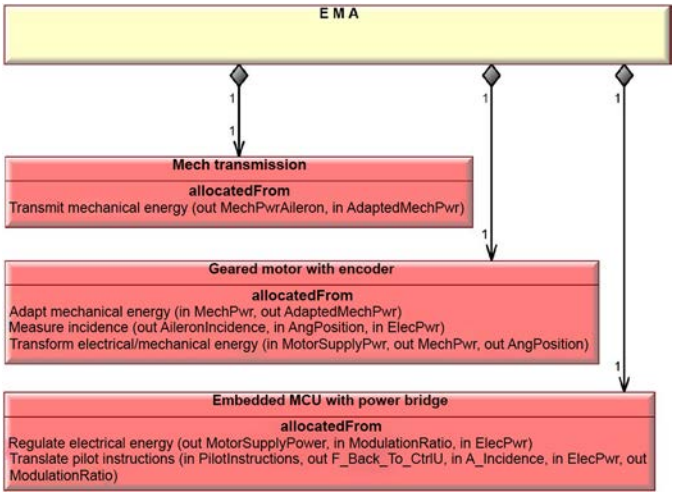

Fig. 9. EMA logical structure with functional allocation.

When no more change needs to be done at the functional level, it is time to move to Step 4) of SafeSysE. The logical architecture is obtained in two steps. The first one is the identification of the components and allocating them to the functions, and the next one then consists in identifying the communication among components. This leads to the BDD representing the system structure in Fig. 9 and the IBD representing the interactions among the components in Fig. 10.

A preliminary FMEA generated from the IBD is given in Fig. 11 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.

V. Automatic Fault Tree Generation From SysML Models

In this section, we will describe our method to generate fault treesautomatically from structural diagrams, i.e., SysML

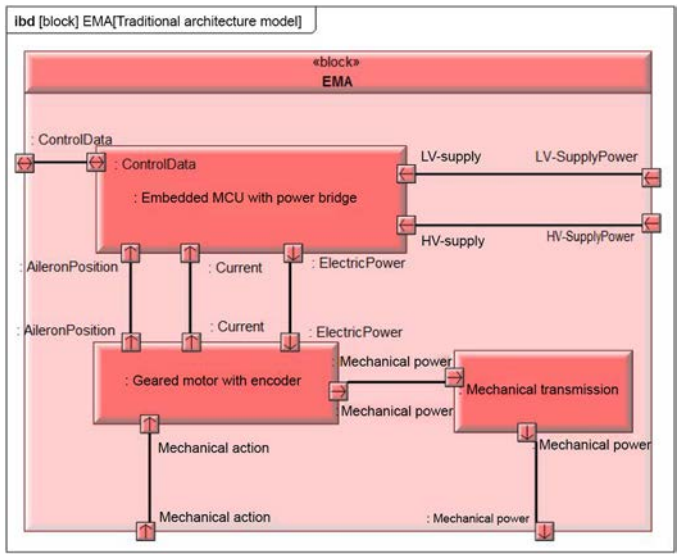

Fig. 10. EMA internal architecture.

IBDs. 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 using two concepts: *directed graph traversal* and *block design patterns*. Each concept is detailed hereafter.

A. 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 multiedges 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 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 and traces back to nodes that are his predecessors and continue 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 visisted.

Component	Function	Failure mode	Local effects	System effects
Embedded MCU with power bridge	*Adapt supply energy*	Electronic hardware defect	Motor stops or rotates randomly	Aileron locked or moving randomly
	Translate pilot instructions	Software error		Aileron locked
		Hardware defect		Aileron locked
		Synchronization error		Aileron locked
		Memory defect or loss		Aileron locked
Geared motor with encoder	*Adapt mechanical energy*	Jam	Locked output	Aileron locked
	Transform energy	Jam		Aileron locked
		Loss of structural integrity		Aileron locked or in free movement
		Short-circuit between two windings		Aileron locked
	Measure current	Hardware defect	Erroneous current measure	Aileron locked
Mechanical transmission	*Transmit energy*	Jam	Motor overloaded & output locked	Aileron locked

Fig. 11. Extract of the component FMEA of the EMA (the text in italics is generated automatically).

B. Block Design Patterns

To facilitate the fault tree generation, we also use the "divide and conquer" principle by partitioning the IBD and treating each partition separately. So, the IBD is splitted into smaller IBDs which are more trivial to solve. Indeed, during the graph traversal, the algorithm also identifies some interesting patterns in an IBD. Each pattern gives rise to a subfault tree and the whole fault tree will be assembled automatically 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*, *Single*, 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 sections describe the recognized patterns as well as their generated partial fault trees. All these patterns are grouped into an illustrating IBD in Fig. 12. Each pattern is surrounded with a dashed rectangle annotated with the corresponding name in an attached note.

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 Fig. 12 is an entry part). In the generated subfault tree [Fig. 13(a)], 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 as follows: 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.

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 (block B6 in Fig. 12). In the corresponding fault tree [Fig. 13(b)], this special output port gives rise to a top event undesired state of the actual block. We will have 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.

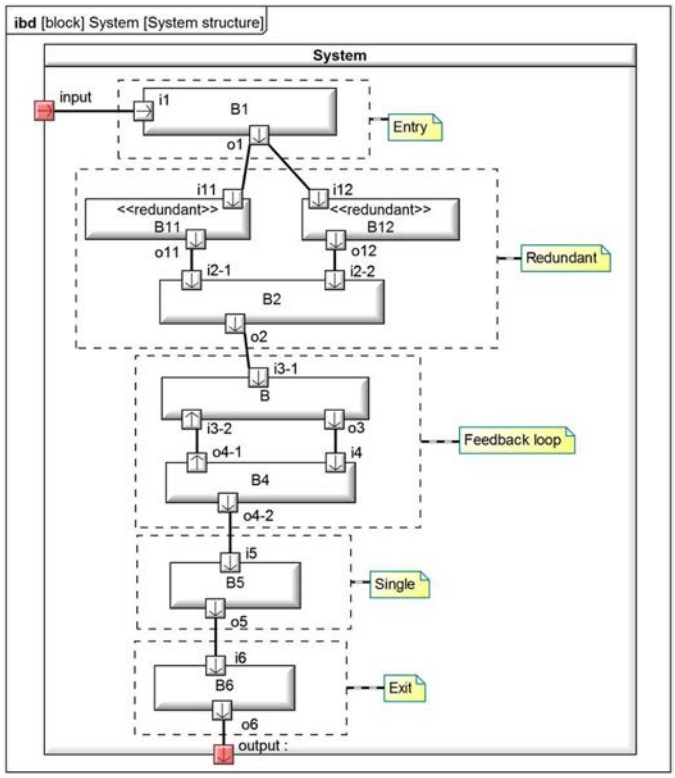

Fig. 12. IBD block design patterns.

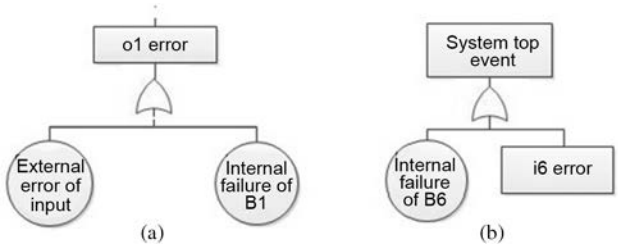

Fig. 13. Fault trees for (a) entry and (b) exit patterns in Fig. 12.

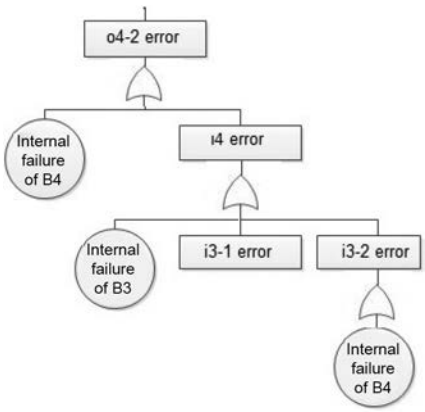

Fig. 14. Fault tree for feedback pattern in Fig. 12.

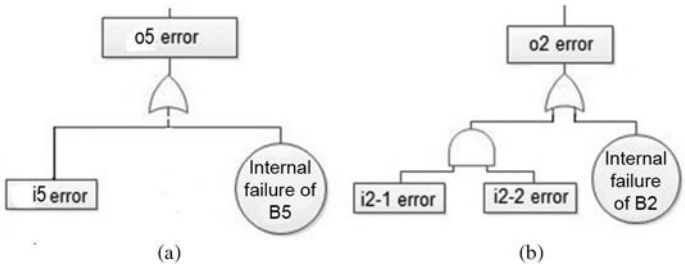

Fig. 15. Fault trees for (a) single and (b) redundant patterns in Fig. 12.

3) Single Pattern: A "Single pattern" is an unique block that is not an entry nor an exit pattern (block B5 in Fig. 12 for example). The construction of the fault tree corresponding to this pattern is straightforward, based on the number of input and output ports of the block as shown in Fig. 15(a).

4) 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 Fig. 12, 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 Fig. 12 is illustrated in Fig. 14.

5) 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, B11, and B12 in Fig. 12). By using the safety profile described in Section III-B, 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 Fig. 15(b).

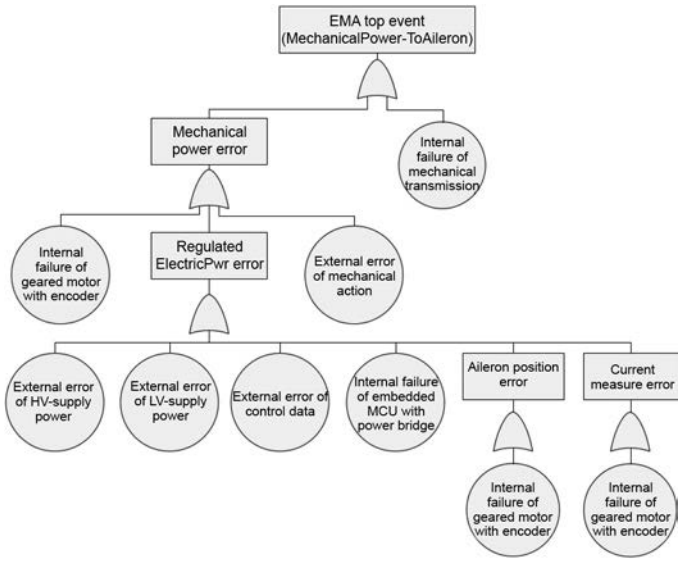

Fig. 16. EMA generic fault tree.

When the whole fault tree is generated automatically from an IBD using the identified patterns and a graph traversal algorithm, we will have a generic fault tree for the corresponding system top event. 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 V-C.

C. Case Study

The starting point of this step is the logical architecture for the EMA given in Fig. 10. Based on this IBD, and according to the patterns and the depth-first search graph traversal algorithm given in the previous sections, the generic fault tree for the "Mechanical Power" output is automatically generated and given in Fig. 16.

Several undesired events can occur at each output. For each specific undesired top event, we can extract from the FMEA results of the previous steps related to 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 Fig. 17. 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.

Our SafeSysE tool can generate fault trees in different formats. Actually, two options are proposed: the SVG image format useful for visualizing the fault tree and the Open-PSA model exchange format to make the results explorable by the XFTA engine [40].

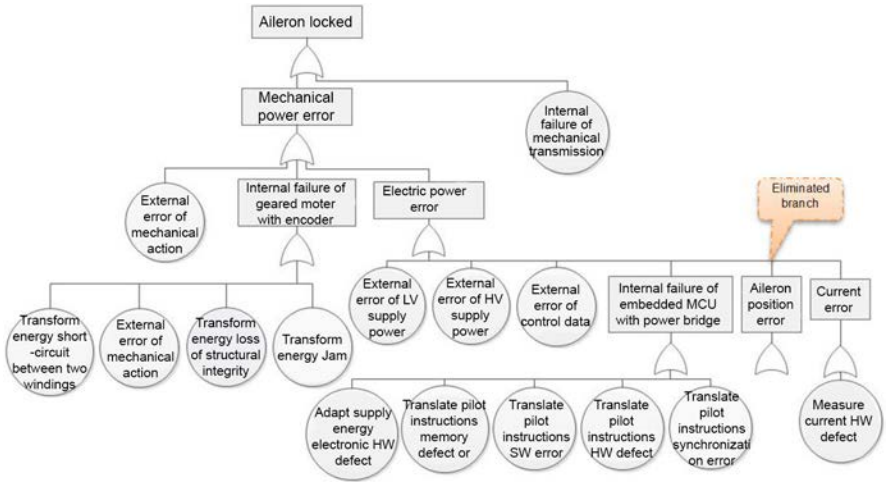

Fig. 17. EMA-specific fault tree for "Aileron locked" top event.

VI. 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 question is how to perform safety analyses efficiently to have full benefit of their results and thus avoid costly and time-consuming redesign iterations. The main contribution of this paper is to deal with this issue by efficiently merging systems engineering process and safety assessment methods in a unique framework that we named SafeSysE. In this SysML-based MBSE approach, we automated the generation of some well-known safety assessment artifacts such as FMEA and FTA. We have extended SysML to integrate some safety-relevant concepts to better support safety analyses. A SysML safety profile has been developed for this purpose. This paper, 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, the results of previous analyses are explored.

As a continuity to this work, 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 such as requirements. We will also examine the different possible ways to model dysfunctional behavior in SysML to improve the V&V of the system. Finally, scalability will be addressed to prove the adequacy of SafeSysE for larger and more complex systems, with different interacting components and state combination problems. Quantitative analysis about the performance of the suggested method will be carried out with different benchmarks.

We would like to thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and suggestions.

References

[1] J. Thomas, "Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis," Ph.D. dissertation, Eng. Syst. Division, Massachusetts Inst. Technol., Cambridge, MA, USA, 2013.

[2] INCOSE, *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. John Wiley and Sons ed., Int. Council Syst. Eng., Hoboken, NJ, USA, Aug. 2015.

[3] OMG, *Systems Modeling Language*, Object Management Group Std. 1.3, Jun. 2012 [Online]. Available: http://www.omg.org/spec/SysML/1.3/

[4] J. Holt and S. Perry, *SysML for Systems Engineering*. London, U.K.: Inst. Eng. Technol., 2008.

[5] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML, The Systems Modeling Language*. San Mateo, CA, USA: Morgan Kaufmann, 2009.

[6] T. Weilkiens, *Systems Engineering With SysML/UML Modeling, Analysis, Design*. San Mateo, CA, USA: Morgan Kaufmann, 2008.

[7] T. A. Johnson, J. M. Jobe, C. J. J. Paredis, and R. Burkhart, "Modeling continuous system dynamics in SysML," in *Proc. ASME Conf. Proc.*, 2007, pp. 197–205.

[8] P. David, V. Idasiak, and F. Kratz, "Reliability study of complex physical systems using SysML," *Reliab. Eng. Syst. Saf.*, vol. 95, no. 4, pp. 431–450, 2010.

[9] E. Adrianarison and J.-D. Piques, "SysML for embedded automotive systems—A practical approach," in *Embedded Real Time Software and Systems*. Toulouse, France, May 2010.

[10] J.-F. Pétin, D. Evrot, G. Morel, and P. Lamy, "Combining SysML and formal models for safety requirements verification," in *Proc. 22nd Int. Conf. Softw. Syst. Eng. Appl.*, Paris, France, Dec. 2010.

[11] P. Helle, "Automatic SysML-based safety analysis," in *Proc. 5th Int. Workshop Model Based Archit. Constr. Embedded Syst.*, Jan. 2012, pp. 19–24.

[12] C. A. Ericson, *Hazard Analysis Techniques for System Safety*. Hoboken, NJ, USA: Wiley, 2005.

[13] E. Balz and J. Goll, "Use case-based fault tree analysis of safety-related embedded systems," in *Proc. Softw. Eng. Appl.*, Nov. 2005.

[14] S. Sharvia, "Integrated application of compositional and behavioural safety analysis," Ph.D. dissertation, Comput. Sci., Univ. Hull, Hull, U. K. Feb. 2011.

[15] *IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems.*, The International Electrotechnical Commission Standard, 2010, [Online]. Available: http://www.iec.ch/functionalsafety/standards/

[16] *Guidelines for Development of Civil Aircraft and Systems*, Society of Automotive Engineers SAE International Standard, sAE-ARP-4754A, 2010.

[17] *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, Society of Automotive Engineers International Standard, sAE-ARP-4761, 1996.

[18] E. M. Clarke, J. O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[19] F. Mhenni, N. Nguyen, H. Kadima, and J.-Y. Choley, "Safety analysis integration in a SysML-based complex system design," in *Proc. IEEE Int. Syst. Conf.*, Orlando, FL, USA, Apr. 2013, pp. 70–75.

[20] F. Mhenni, "Safety analysis integration in a systems engineering approach for mechatronic systems design," Ph.D. dissertation, Sciences pour l'Ingnieur, Ecole Centrale de Paris, Châtenay-Malabry, France, 2014.

[21] R. Laleau, F. Semmak, A. Matoussi, D. Petit, A. Hammad, and B. Tatibouet, "A first attempt to combine SysML requirements diagrams and B," *Innov. Syst. Softw. Eng.*, vol. 6, pp. 47–54, 2010.

[22] A. Albinet, J.-L. Boulanger, H. Dubois, M.-A. Peraldi-Frati, Y. Sorel, and Q.-D. Van, "Model-based methodology for requirements traceability in embedded systems," in *Proc. 3rd Eur. Conf. Model Driven Archit.*, Jun. 2007.

[23] *ISO 26262 Road Vehicles-Functional Safety*, International Standardization Organization Standard, 2011.

[24] L. Vismari, J. Camargo, J. de Almeida, A. da Silva Neto, R. Gimenes, and P. Cugnasca, "A practical analytical approach to increase confidence in software safety arguments," *IEEE Syst. J.*, pp. 1–12, May 2015.

[25] R. Guillerm, H. Demmou, and N. Sadou, "Safety evaluation and management of complex systems: A system engineering approach," *Concurr. Eng. Res. Appl.*, vol. 20, no. 2, pp. 149–159, Jun. 2012.

[26] F. Thomas and F. Belmonte, "Performing safety analyses and SysML designs conjointly: A viewpoint matter," in *Proc. Complex Syst. Des. Manage.*, Dec. 2011.

[27] F. Belmonte and E. Soubiran, "A model based approach for safety analysis," in *Computer Safety, Reliability, and Security*, F. Ortmeier and P. Daniel, Eds. New York, NY, USA: Springer, 2012, vol. 7613, pp. 50–63.

[28] A. Arnold, A. Griffault, G. Point, and A. Rauzy, "The AltaRica language and its semantics," *Fundam. Inf.*, vol. 34, pp. 109–124, 2000.

[29] H. Xu, L. Xing, and R. Robidoux, "DRBD: Dynamic reliability block diagrams for system reliability modeling," *Int. J. Comput. Appl.*, vol. 31, no. 2, p 202, 2009.

[30] M. Rausand and A. Hoyland, *System Reliability Theory—Models, Statistical Methods, and Applications*. Hoboken, NJ, USA: Wiley, 2008.

[31] A. Garro and A. Tundis, "Enhancing the RAMSAS method for system reliability analysis—An exploitation in the automotive domain," in *Proc. SIMULTECH*, Jul. 2012, pp. 328–333.

[32] F. Tajarrod and G. Latif-Shabgahi, "A novel methodology for synthesis of fault trees from MATLAB Simulink model," *World Acad. Sci., Eng. Technol.*, vol. 17, no. 5, pp. 1256–1262, 2008.

[33] A. Joshi, P. Binns, and S. Vestal, "Automatic generation of static fault trees from AADL models," in *Proc. IEEE / IFIP Conf. Dependable Syst. Netw.*, Edinburgh, Scotland, Jun. 2007.

[34] M. Bozzano and A. Villafiorita, "The FSAP/NuSMV-SA safety analysis platform," *Int. J. Softw. Tools Technol. Transfer*, vol. 9, no. 1, pp. 5–24, 2007.

[35] Y. Papadopoulos and J. A. McDermid, "Hierarchically performed hazard origin and propagation studies," in *Proc. 18th Int. Conf. Comput. Saf., Reliab. Secur. (SAFECOMP)*, Toulouse, France, 1999, vol. 1698, pp. 139–152.

[36] N. Yakymets, H. Jaber, and A. Lanusse, "Model-based system engineering for fault tree generation and analysis," in *Proc. 1st Int. Conf. Model-Driven Eng. Softw. Dev.*, Barcelona, Spain, Feb. 19–21, 2013, pp. 210–214.

[37] F. Mhenni, J.-Y. Choley, O. Penas, R. Plateaux, and M. Hammadi, "A SysML-based methodology for mechatronic systems architectural design," *Adv. Eng. Informat.*, vol. 28, no. 3, pp. 218–231, 2014.

[38] Object-Management-Group, *XML Metadata Interchange (XMI) Specification*, Object Management Group Standard [Online]. Available: http://www.omg.org/spec/XMI/

[39] I. Moir and A. Seabridge, *Aircraft Systems—Mechanical Electrical and Avionics Subsystems Integration*, 3rd ed. England, U.K.: John Wiley & Sons,, Aug. 2008.

[40] M. Hibti, T. Friedlhuber, and A. Rauzy, "Overview of the open PSA platform," in *Proc. Int. Joint Conf. PSAM'11/ESREL'12*, Jun. 2012.