https://doi.org/10.1109/DASC52595.2021.9594315

Integrating Safety into MBSE Processes with Formal Methods

Alexander Ahlbrecht
Institute of Flight Systems
German Aerospace Center (DLR)
Braunschweig, Germany
alexander.ahlbrecht@dlr.de

Umut Durak
Institute of Flight Systems
German Aerospace Center (DLR)
Braunschweig, Germany
umut.durak@dlr.de

***Abstract*—Emerging segments such as Urban Air Mobility require new safety-critical avionic systems. The complexity of these avionic systems has ever been increasing, but even more rapidly in the last two decades in form of the number of components, functions, and interactions. At the same time, demanding time-to-market requirements have to be adhered to by development companies. To cope with these challenges, agile development approaches are required that guarantee safety-by-construction. This paper presents an endeavor to tackle these challenges by holistic utilization of Model-based Systems Engineering, System-Theoretic Process Analysis, and formal methods. The approach is demonstrated in a use-case that analyzes a simplified Collision Avoidance System architecture. Results show that the presented approach is able to improve the development by automating and validating error-prone tasks of the safety assessment.**

***Index Terms*—MBSE, SysML, STPA, Safety, OCL**

I. Introduction

Two contradicting factors highlighted by the Systems Engineering (SE) vision 2025 of International Council on Systems Engineering (INCOSE) are demanding time-to-market requirements and complexity growth in terms of components, functions and interactions [1]. Additionally, these challenges are further reinforced by emerging segments like Urban Air Mobility (UAM), which introduce similar requirements towards the system development. Tackling the increasing complexity, while achieving fast time-to-market for safety-critical avionic systems, is a challenging task for companies. To achieve this task, companies require agile development approaches that can guarantee safety-by-construction. This paper represents an endeavor that leverages, integrates, and enhances state of the art of methods like: *Model-Based Systems Engineering* (MBSE), *System-Theoretic Process Analysis* (STPA) and formal methods. All three approaches and their contributions to the topic will be explained shortly in the following paragraphs.

A recent trend to tackle complexity during development is MBSE. MBSE can be characterized as the formalized application of modeling to support SE development activities [2]. To deal with arising consistency problems in classical SE developments, where a document is created for every discipline, MBSE uses a central model for development and documentation [3]. One prominent modeling language of MBSE is *Systems Modeling Language* (SysML).

To enhance MBSE, formal methods are used. Formal methods are mathematics based techniques for the specification and verification of hardware and software systems. They provide a means for the systematic description of systems and support development and verification of systems [4], [5]. Using formal methods with MBSE is achieved in this paper through SysML. SysML can be currently described as semi-formal notation [6] and therefore builds a good basis for the application of formal methods. Combining formal methods with semi-formal notations, used in industry, is one way of improving semi-formal notations, while simultaneously increasing the popularity of formal methods as explained in [7]. One common way of formally verifying properties within SysML models is the usage of Object Constraint Language (OCL) [8]. OCL is a typed formal language, which uses functions and logical operators to analyze properties of sets [9]. SysML tools, like the *Cameo Systems Modeler*, allow using OCL for formal verification and validation of the modeled systems [10].

To accommodate for new and complex software driven cyber-physical systems in the aviation domain, the STPA hazard analysis was developed. The main issue that STPA tries to address, is the lack of considerations for interactions (human-system, system-system) and software specifications in classical safety approaches [11]. In this paper, we consider how a formalized STPA approach can be integrated into SysML. By integrating the STPA within SysML, a promising connection between the development and safety analysis tasks is established. Even though, this combination of SysML and STPA was already demonstrated by other authors, the formalization in this paper enables to additionally automate parts of the analysis. In the following, the combined approach is called SysML-STPA. Furthermore, OCL is used to automatically validate parts of the safety analysis.

The following sections will elaborate the SysML-STPA concept in more detail. First, required background information will be introduced in Section II. Afterwards, the developed methodology will be described in Section III. The methodology is then demonstrated with a simplified *Collision Avoidance System* (CAS) use-case in Section IV. Finally, the results are discussed in Section V and a short summary is given in Section VI.

II. Related Work

This section starts with an overview of classical safety processes and standards in the aviation industry. Afterwards, it is explained why the STPA was developed and how this safety analysis can be formalized to enable automation of analysis parts. Furthermore, it is described how the STPA can be integrated into the SysML. Finally, a short overview is given that explains where this work positions itself in regard to this related work.

A. Classical Aviation Safety Processes and Standards

Safety processes and standards are an important part of the development of safety-critical systems. Looking at the aviation domain, the two main development standards are ARP4754A and ARP4761. ARP4754A [12] explains the main approach and development constraints that are needed for the certification of complex aviation systems. Additionally, ARP4761 [13] explicitly describes the required safety considerations within the development of aviation systems. These safety considerations include the use and connection of multiple hazard analysis examples: e.g. *Functional Hazard Assessment* (FHA), *Failure Mode And Effects Analysis* (FMEA), and *Fault Tree Analysis* (FTA). Furthermore, standards like DO-178C [14] and DO-254 [15] guide the design assurance of software and hardware, respectively.

B. Safety Consideration with STPA

To accommodate for new and complex software driven cyber-physical systems in the aviation domain, the STPA was developed. STPA is a top-down hazard analysis that tries to address the lack of considerations for system interactions and software specifications in the classical hazard analyses [11]. To incorporate these aspects, STPA uses hierarchical control structure model of the developed system for its safety analysis purposes. To analyze the system, STPA is composed out of two main steps. First, the model of the system is analyzed towards potentially *Unsafe Control Actions* (UCA), i.e. actions able to introduce a hazardous system state. In the second step, affected parts of the system model are analyzed to identify causes for executing UCAs. If such a cause was identified, mitigating safety constraints are introduced, preventing the occurrence of these causes. An extensive description of STPA is presented in [11], [16].

C. Formalization of STPA

The formalization of STPA in [17] is based on the clustering of every system component (source controllers), their interactions (control actions), the considered hazards and the corresponding analysis context into formal sets [17]. Formally, these parts are defined as: source controller $SC \in \mathcal{SC}$, control action $CA \in \mathcal{CA}$, hazards $H \in \mathcal{H}$ and context $Co \in \mathcal{C}o$. For all combination of these sets, a *hazardous provided* function was specified, formalizing these relations as: $HP(H, SC, CA, Co)$ and returning true for every combination that is hazardous when provided. Subsequently, a second *hazardous not provided* function was specified, formalizing these relations as: $HNP(H, SC, CA, Co)$ and returning true for every combination that is hazardous when not provided. Using these definitions, a *requirement* function $R(SC, CA, Co)$ was defined that is used to automatically generate requirements in combination with the previous functions. On the one hand, a requirement that the control action shall be provided can be generated for every true result of the HNP function: $HNP \rightarrow R$. On the other hand, a requirement that the control action shall not be provided can be generated for every true result of the HP function: $HP \rightarrow \neg R$. Even though the general idea behind the formalization of the STPA is really promising, it was not yet shown how this formalized STPA approach can be leveraged within a state of the art modeling language, like SysML. The proposed approach in this paper considers this aspect by extending and integrating it into SysML.

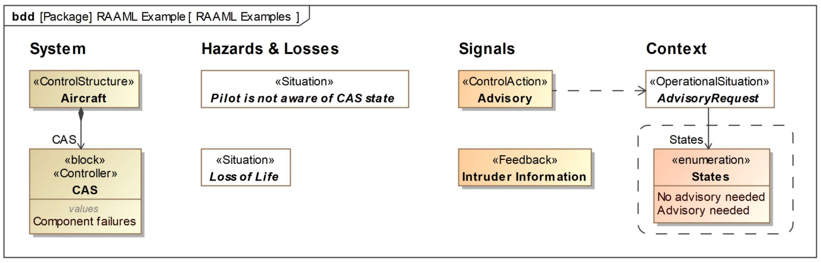
D. Combining STPA with SysML

Integrating safety considerations into SysML is a promising concept, which is why the *Object Management Group* (OMG) started to integrate various safety analysis types into the SysML [18]. Consequently, this led to the development of the *Risk Analysis and Assessment Modeling Language* (RAAML) [19]. The RAAML shows how five safety analysis types can be integrated into the SysML with the STPA being one of them. As a foundation for the further developments, a few stereotypes and concepts were borrowed for the proposed SysML-STPA. These are visualized in Fig. 1a, namely the stereotypes ‹‹ControlStructure›› and ‹‹Controller›› for the system description, the stereotypes ‹‹ControlAction›› and ‹‹Feedback›› for the description of system interactions, the ‹‹Situation›› stereotype for the specification of hazards and losses, and finally the ‹‹OperationSituation›› stereotype for the context labeling. Furthermore, the concept of using an internal block diagram (ibd) for the STPA control structure was borrowed and visualized in Fig. 1b.

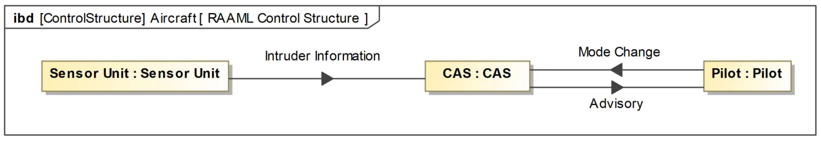
Not only the RAAML did consider the combination of the SysML and STPA. Another example is the *Model-based STPA* (MBSTPA) shown in [20], where the usage of state charts in SysML is proposed for modeling of the STPA control structure and the analysis execution. Additionally, it was shown in [21] that by formalizing the resulting requirements of the STPA in SysML, it is possible to check them via a formal model checking tool.

E. Contribution of Paper

As described in the previous sections, this paper wants to combine MBSE, STPA and formal methods. This is why, the presented related works are important contributions to the proposed approach. Even though, there is already the RAAML describing a way to combine MBSE in form of the SysML with STPA, it is not yet demonstrated how this can be combined with the formalization presented in Section II-C. Combining all ideas of the related work and formalizing the STPA in the SysML enables even further benefits than



(a) Reused RAAML STPA Stereotypes


ibd [ControlStructure] Aircraft [RAAML Control Structure]
Sensor Unit : Sensor Unit
Intruder Information
CAS : CAS
Mode Change
Advisory
Pilot : Pilot

(b) RAAML STPA Control Structure

Fig. 1: Aspects Reused from the RAAML STPA Implementation

previously described in [17]. Some examples will be presented in the following section.

III. Proposed Approach

In this section, SysML-STPA will be introduced. This approach shows how STPA can be formally integrated into SysML and how the formalization is leveraged to automate and validate parts of the analysis. The SysML-STPA is described in an abstract way in this section. An application example is presented in the following Section IV.

A. RAAML - SysML-STPA Foundation

Prior to the SysML-STPA analysis execution, a description of the analysis scope is needed that is established by the usage of the RAAML stereotypes as depicted in Fig. 1a. Defining the scope includes the system description, signal specification, definition of hazards and losses and the specification of considered contexts.

For the context definition, the ‹‹OperationalSituation›› stereotype of RAAML was extended. It was augmented with an ‹‹enumeration›› property named *States* to formally define the context variables of the context. This extension is required to enable an automated generation of hazard tables described later in this section. Since this is not a direct part of the RAAML, this is visualized in Fig. 1a with the dashed line surrounding the ‹‹enumeration››. Moreover, a ‹‹Dependency›› relation between the contexts and the control actions was introduced to specify which control action needs to be analyzed in which context. All the previously described stereotypes should be applied in the model to define the scope of the analysis. After defining the scope of the analysis, the STPA control structure needs to be modeled. This control structure specifies all connections and interactions between the systems as depicted in Fig. 1b, while using the model elements augmented with corresponding stereotypes.

B. Extended SysML-STPA Profile

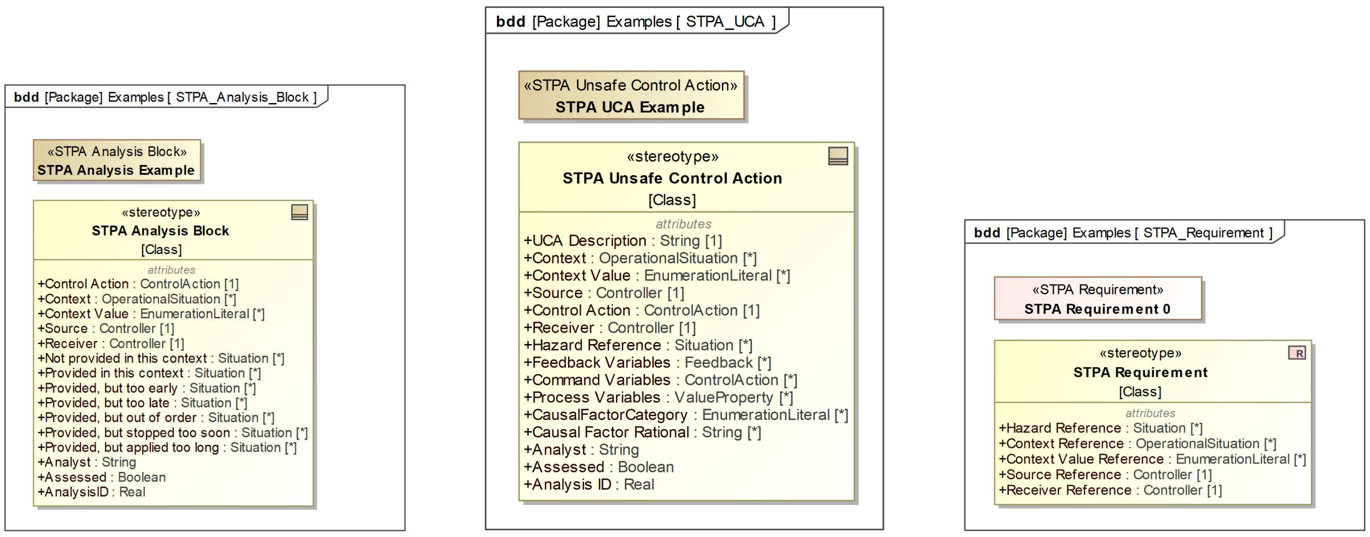
RAAML represents a good foundation for the SysML-STPA, since it already introduces most of the stereotypes necessary for the SysML-STPA. One central point, which the RAAML currently does not address, is the ability to execute the analysis in centralized hazard tables. This requirement is addressed in the SysML-STPA with the introduction of the ‹‹STPA Analysis Block›› and ‹‹STPA Unsafe Control Action›› stereotypes. Both are displayed in Fig. 2 with their corresponding attributes.

Moreover, some additional stereotypes were required to fulfill the formal needs of the SysML-STPA. Stereotypes of this category are: ‹‹STPA Causal Factor Category›› and ‹‹Process Variable ›› of Fig. 4 as well as the ‹‹STPA Requirement›› in Fig. 2c can be mentioned. All extensions will be described in more detail in the following.

C. Analysis Block - Identifying Hazardous Actions

If the scope of the analysis and the corresponding control structure were modeled as presented in Fig. 1, the first implemented function can be used. This function enables to create instances of the ‹‹STPA Analysis Block››, where the first five attribute entries of Fig. 2a are filled automatically. Namely, the: *Control Action*, *Context*, *Context Value*, *Source*, and *Receiver* are inserted automatically.

Traditionally, with the use of guidewords, the first analysis step of the STPA is executed to identify potential hazardous control actions. This step is substituted in SysML-STPA with the analysis of all created ‹‹STPA Analysis Block›› instances. Within the attributes of the ‹‹STPA Analysis Block›› displayed in Fig. 2a, seven guidewords were implemented. These guidewords present ways how a potential hazardous situation can emerge, when providing or not providing a control action as visualized in Fig. 3. During the SysML-STPA process, the guideword attributes are used to establish a connection between an analysis block instance and one or more possibly resulting hazards. These hazards are modeled beforehand in


bdd [Package] Examples [STPA_Analysis_Block]
«STPA Analysis Block»
STPA Analysis Example
«stereotype»
STPA Analysis Block
[Class]
attributes
+Control Action : ControlAction [1]
+Context : OperationalSituation [*]
+Context Value : EnumerationLiteral [*]
+Source : Controller [1]
+Receiver : Controller [1]
+Not provided in this context : Situation [*]
+Provided in this context : Situation [*]
+Provided, but too early : Situation [*]
+Provided, but too late : Situation [*]
+Provided, but out of order : Situation [*]
+Provided, but stopped too soon : Situation [*]
+Provided, but applied too long : Situation [*]
+Analyst : String
+Assessed : Boolean
+AnalysisID : Real
bdd [Package] Examples [STPA_UCA]
«STPA Unsafe Control Action»
STPA UCA Example
«stereotype»
STPA Unsafe Control Action
[Class]
attributes
+UCA Description : String [1]
+Context : OperationalSituation [*]
+Context Value : EnumerationLiteral [*]
+Source : Controller [1]
+Control Action : ControlAction [1]
+Receiver : Controller [1]
+Hazard Reference : Situation [*]
+Feedback Variables : Feedback [*]
+Command Variables : ControlAction [*]
+Process Variables : ValueProperty [*]
+CausalFactorCategory : EnumerationLiteral [*]
+Causal Factor Rational : String [*]
+Analyst : String
+Assessed : Boolean
+Analysis ID : Real
bdd [Package] Examples [STPA_Requirement]
«STPA Requirement»
STPA Requirement 0
«stereotype»
STPA Requirement
[Class]
attributes
+Hazard Reference : Situation [*]
+Context Reference : OperationalSituation [*]
+Context Value Reference : EnumerationLiteral [*]
+Source Reference : Controller [1]
+Receiver Reference : Controller [1]

(a) STPA Analysis Block (b) STPA Unsafe Control Action (c) STPA Requirement

Fig. 2: Additional STPA Stereotypes

TABLE I: Requirement and UCA Structure related to Guidewords

Guide	Requirement Text Structure	UCA Description Structure
(1)	The $[SC]$ shall always be able to apply $[CA]$ to the $[SC]$ when $[Co]$	The $[SC]$ does not apply $[CA]$ to the $[SC]$ when $[Co]$
(2)	The $[SC]$ shall not apply $[CA]$ to the $[SC]$ out of order when $[Co]$	The $[SC]$ applies $[CA]$ to the $[SC]$ out of order when $[Co]$
(3)	The $[SC]$ shall not stop applying $[CA]$ too soon to the $[SC]$ when $[Co]$	The $[SC]$ stops applying $[CA]$ too soon to the $[SC]$ when $[Co]$
(4)	The $[SC]$ shall not apply $[CA]$ to the $[SC]$ too late when $[Co]$	The $[SC]$ applies $[CA]$ to the $[SC]$ too late when $[Co]$
(5)	The $[SC]$ shall not apply $[CA]$ to the $[SC]$ too early when $[Co]$	The $[SC]$ applies $[CA]$ to the $[SC]$ too early when $[Co]$
(6)	The $[SC]$ shall not apply $[CA]$ to the $[SC]$ too long when $[Co]$	The $[SC]$ applies $[CA]$ to the $[SC]$ too long when $[Co]$
(7)	The $[SC]$ shall never be able to apply $[CA]$ to the $[SC]$ when $[Co]$	The $[SC]$ applies $[CA]$ to the $[SC]$ when $[Co]$

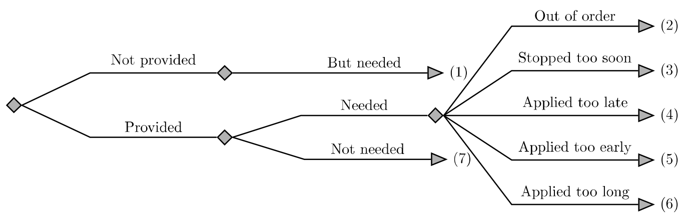


Fig. 3: STPA Guidewords (adapted from [22])

the scope of SysML-STPA as described in Section III-B, where every hazard is marked with an ‹‹Situation›› stereotype. Finally, the analysis block attributes are completed with additional entries for the *Analyst*, the *Assessed* status, and an *ID*.

D. STPA Requirement - Automatically Derived

Due to the establishment of connections between analysis block instances and potentially resulting hazards, the second automated function can be applied to create ‹‹STPA Requirement›› instances. Using the formal functions that were introduced in Section II-C [17], requirements can be generated automatically using the attributes of the analysis block instances. Essentially, the analysis blocks entries define the two functions: *hazardous provided* $HP(H, SC, CA, Co)$ and *hazardous not provided* $HNP(H, SC, CA, Co)$. Hence, the requirements can be generated similarly: $HP \rightarrow \neg R$ and $HNP \rightarrow R$. In SysML-STPA, the requirement function $R(SC, CA, Co)$ uses predefined text building blocks and inserts the corresponding elements as shown in TABLE I, when creating the text for the ‹‹STPA Requirement›› instances. Simultaneously, the other reference attributes of the requirement are added, which are shown in Fig. 2c.

E. UCA Block - Identifying Potential Causes

Not only the text for the requirements can be generated automatically in this way. Additionally, the *UCA Description* attribute of the ‹‹STPA Unsafe Control Action›› stereotype instances can be generated as displayed in TABLE I. Formally, this can be described as the *unsafe control action* function $UCA(SC, CA, Co)$, which has an inverse relationship to the HP and HNP functions: $HP \rightarrow \neg UCA$ and $HNP \rightarrow UCA$. This means that every time a hazard connection is established within a guideword of an analysis block, a UCA instance can be generated automatically. By further utilizing the formalized attributes of the analysis block, also other attributes of the ‹‹STPA Unsafe Control Action›› stereotype instances of Fig. 2b can be copied from the analysis block and entered automatically: *Control Action*, *Context*, *Context Value*, *Source*, *Receiver*. When analyzing causes of the UCA execution of a source controller, possible reasons can be:

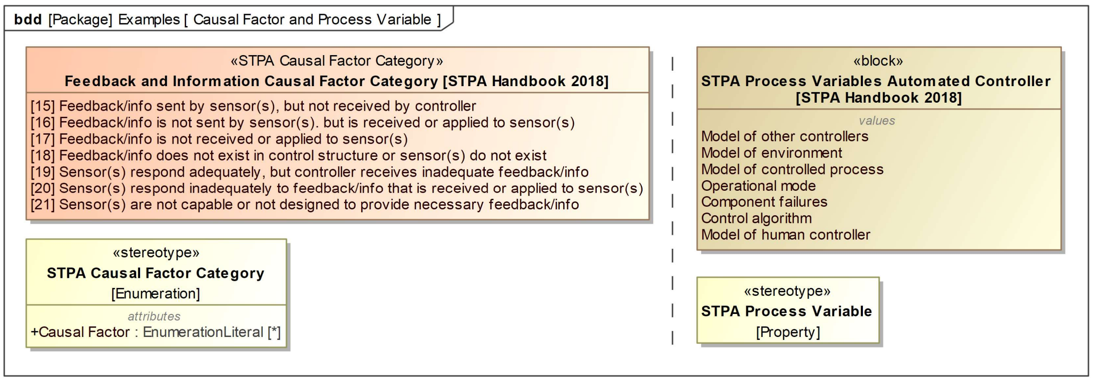

bdd [Package] Examples [Causal Factor and Process Variable]
«STPA Causal Factor Category»
Feedback and Information Causal Factor Category [STPA Handbook 2018]
[15] Feedback/info sent by sensor(s), but not received by controller
[16] Feedback/info is not sent by sensor(s). but is received or applied to sensor(s)
[17] Feedback/info is not received or applied to sensor(s)
[18] Feedback/info does not exist in control structure or sensor(s) do not exist
[19] Sensor(s) respond adequately, but controller receives inadequate feedback/info
[20] Sensor(s) respond inadequately to feedback/info that is received or applied to sensor(s)
[21] Sensor(s) are not capable or not designed to provide necessary feedback/info
«stereotype»
STPA Causal Factor Category
[Enumeration]
attributes
+Causal Factor : EnumerationLiteral [*]
«block»
STPA Process Variables Automated Controller [STPA Handbook 2018]
values
Model of other controllers
Model of environment
Model of controlled process
Operational mode
Component failures
Control algorithm
Model of human controller
«stereotype»
STPA Process Variable
[Property]

Fig. 4: Causal Factors and Process Variables Stereotypes with Examples Above

incorrect incoming *Commands*, incorrect returned *Feedback*, and incorrect internal *Process Variables*.

To complement these three attributes, an additional stereotype was required for the *Process Variables*. This ‹‹STPA Process Variable›› stereotype is depicted in Fig. 4. Additionally, exemplary *Process Variables* for automated controllers were inherited from the STPA Handbook [16], implemented as helping examples, and displayed in Fig. 4. If these three mentioned attributes are modeled, decorated with the corresponding stereotype, and connected to the controllers, they can also be inserted automatically into the generated UCA instances.

Moreover, the additional *Causal Factor Category* attribute is given within the ‹‹STPA Unsafe Control Action›› stereotype, where causal factors can be entered to support the causal analysis within SysML-STPA. Corresponding examples are the feedback related causal factors displayed in Fig. 4. For this purpose, 21 causal factor categories were identified within the STPA-Handbook [16] and formally deposited as ‹‹EnumerationLiterals›› in the SysML-STPA profile.

All possibilities to fill attributes of the UCA instances are combined into the third automated function. The inserted information can then be used by the engineer to find causes for the occurrence of UCAs. UCA instances are, like the instances of analysis blocks, proposed to be analyzed in tables for usability reasons with the following process. The UCA description of the UCA instance tells the engineer which UCA shall be considered in the corresponding UCA instance. Additionally, all automatically inserted variables (*Feedback*, *Command*, *Process*) present possible causes for the UCA occurrence. These variables, in combination with the helping causal factor category sentences, help the engineer to come up with possible causes for the UCA execution. Finally, these identified causes are documented as text within the ‹‹Causal Factor Rational›› attribute of the UCA instance. This process, to identify potential causes, is repeated, till all causes are found and documented. The final SysML-STPA step includes the usage of the identified causal rationals to derive safety constraints preventing UCAs from happening. This final step is not yet formalized and therefore not demonstrated within this paper, but has potential to be partially formalized in future endeavors.

F. Analysis Validation with OCL

Prior to this section, it was described how the creation of analysis parts can be automated through formalization. Besides this, it is also possible to validate parts of the analysis, when following a formal structure. To enable the validation, the OCL was chosen as formal validation language, due to its inherent connection with SysML. Because of the limited size of the paper, only two simple example checks will be presented. One situation that can be checked during the use of SysML-STPA is, if there are hazard entries in both the *Provided in this context* and *Not provided in this context* attribute of one of the analysis block stereotype instances displayed in Fig. 2a. This would indicate a design flaw in the analyzed system, since this would mean that both executing and not executing the control action would lead to a hazard. Hence, these entries should be looked at in more detail again in the analysis. This correlation was also formally described in [17]. Using the OCL, corresponding entries of the analysis blocks can be validated with the validation rule in Fig. 5.

```
context STPA Analysis Block inv CheckForInconsistenciesInSTPA:
if
    not self._'Provided in this context'→oclIsUndefined() and
    not self._'Not provided in this context'→oclIsUndefined()
then false
else true
endif
```

Fig. 5: OCL Validation of Potential Conflicts

Another example is the validation, if an *Analyst* name was entered in the corresponding attribute for every analysis block that was already marked as assessed. This might not sound like a very important check, but could be really helpful for certification purposes. To elucidate this, one important part of the safety assessment and the development in general is the traceability throughout all phases. This means that every decision should be documented and comprehensible. Hence, it is important to be able to trace and recap safety related decisions to the corresponding analyst. This is facilitated within SysML-STPA with the analyst name check.

G. *Further Potential of Formalization*

Some advantages of formalization were described in the previous sections. These include automation and validation of analysis parts. Even though these are already promising advantages, there is even further potential to use the formalization. One example is the use of metrics, which can be used in various ways within the development. Exemplary, a metric was implemented that allows to track the analysis progress by looking at the *Assessed* attribute status of the analysis block. This is only one example of many possible applications based on the formalization of the safety assessment. Consequently, this is one area that has a lot of potential to further improvements.

IV. CAS Use-Case

In this section, the previously described SysML-STPA will be demonstrated with a simplified Collision Avoidance System (CAS) use-case. CAS is an avionic system that provides advisories to the pilot to prevent mid-air collisions of aircraft. To enable the automated functions, it is also required to follow the formal style described in Section III, by using the stereotypes and connections required for the SysML-STPA process.

A. *SysML-STPA - System and Control Structure*

For the first step of SysML-STPA, it is needed to model the system and its control structure. In this CAS use-case, the main system is an *Aircraft*. Within this *Aircraft*, only the *CAS*, the corresponding *Sensor Unit* and the *Pilot* are considered. This system description is also modeled in Fig. 6, where every *Aircraft* component is formally marked with a ‹‹Controller›› stereotype.

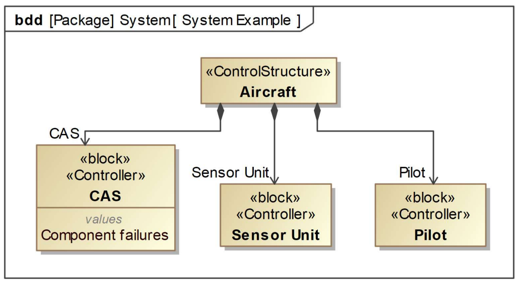


Fig. 6: Aircraft System Components

To visualize the considered system interactions, the SysML-STPA control structure is used in Fig. 7. In this example, two control actions and one feedback are considered. *CAS* can provide an *Advisory* to the *Pilot* to indicate that a collision could happen with the current flight trajectory. Based on the *Intruder Information* feedback by the *Sensor Unit*, these advisories can be calculated. Moreover, the *Pilot* has the ability to change the mode of the *CAS* to enable or disable the *CAS* advisories. Again, the formal style is followed by applying the ‹‹ControlAction›› and ‹‹Feedback›› stereotype to the signals. For the control actions, the SysML stereotypes are shown in Fig. 10.

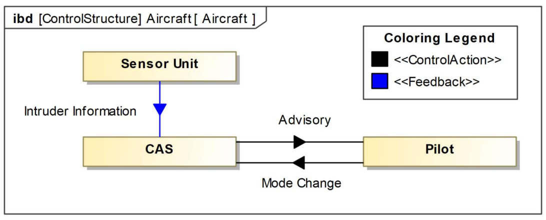
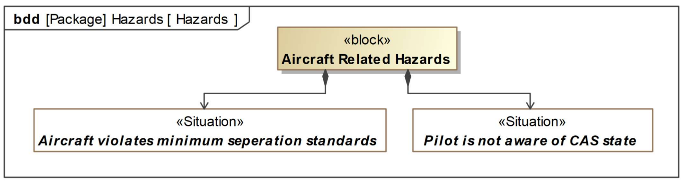


Fig. 7: Aircraft Control Structure

B. *SysML-STPA - Scope and Context*

Following the system description, the scope of the analysis needs to be defined in terms of the hazards and losses that shall be considered. Hence, the hazards *Aircraft violating the minimum separation standards* and *Pilot is not aware of CAS state* were chosen and modeled in Fig. 8. Hazards can lead to losses and for this analysis the losses *Property Damage* and *Loss of Life* are considered and modeled in Fig. 9.



bdd [Package] Hazards [Hazards]
«block»
Aircraft Related Hazards
«Situation»
Aircraft violates minimum seperation standards
«Situation»
Pilot is not aware of CAS state

Fig. 8: Aircraft Related Hazards

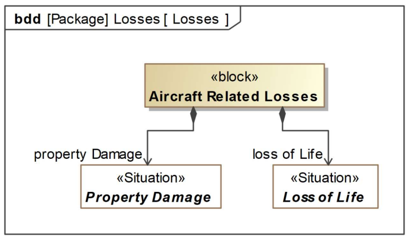


Fig. 9: Aircraft Related Losses

To finish the specification of the analysis scope, the relevant contexts situations for the analysis are required. Since every control action shall be analyzed in relevant context situations within SysML-STPA, it is necessary to define the relation between every control action and corresponding contexts. This was done for the *Advisory* and *Mode Change* control actions in Fig. 10 with a ‹‹Dependency›› relation. As depicted, the *Advisory* control action is connected to the *AdvisoryRequest* as well as the *IntruderPresent* context, which again have their own internal context variables within the *States* ‹‹Enumeration››. The meaning behind the established relations is that the *Advisory* shall be analyzed according to both the *AdvisoryRequest* and *IntruderPresent* contexts. Similarly, a ‹‹Dependency›› relation was established between the *Mode Change* control action and the complementary *ModeChange* context.

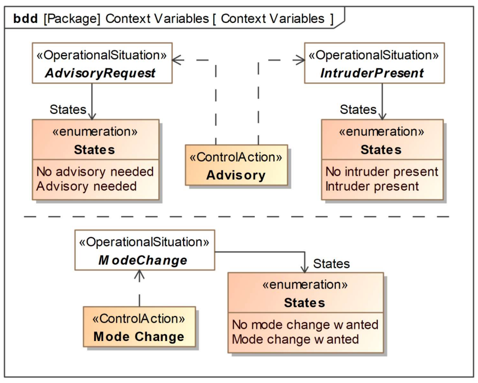


Fig. 10: Aircraft Context Variables

C. SysML-STPA - Identifying Hazardous Actions

Now that all fundamental parts of the system were modeled previously in this section, the implemented automated functions can be applied to generate the first analysis part. When executing the function to generate instances of analysis blocks, all analysis blocks displayed as *STPA Elements* in Fig. 11 are created. As explained in Section III-C, all attribute entries upon the guideword attribute columns are automatically inserted. To prevent overloading the figure, only two of the seven guidewords of Fig. 3 are displayed in this example. For control actions, where more than one relation is drawn to a context, all combinations of context values are created as visible at the rows three till six of Fig. 11.

Utilizing the automatically generated analysis block elements, the first analysis step was executed. This means that for every analysis block element, it was analyzed whether a executing the control action in the corresponding context can lead to a hazard. When one of the previously modeled hazards can occur in consideration with the guideword, a connection to the corresponding hazard is established in guideword field of the analysis block element. Exemplary, in the first row of Fig. 11 providing the *Mode Change* control action when *No mode change wanted*, the hazard could occur that the *Pilot* is not aware of the *CAS* state. This analysis process was repeated for every row in the table of Fig. 11. Moreover, when finishing the analysis of one element, the *Assessed* attribute was checked and the *Analyst* was inserted.

Additionally, the validation of the analysis is visualized in Fig. 11, where two rows are marked due to the implemented validation rules of Section III-F. The first validation result is demonstrated with the yellow color (warning) in the second row of Fig. 11, due to the missing *Analyst* entry. Secondly, the fifth row of Fig. 11 is marked with a red color (error), because entries in both guideword columns were provided. As explained in III-F, this can indicate a design flaw or mistake in the analysis, where both cases need to be handled. Moreover, Fig. 12 shows how a metric can be used to calculate the analysis progress in percentage over the development time. Since all analysis block instances of Fig. 11 are assessed, the progress consequently is 100%. Metrics like these gain in benefit in larger developments, where tracking the analysis progress is required and more challenging.

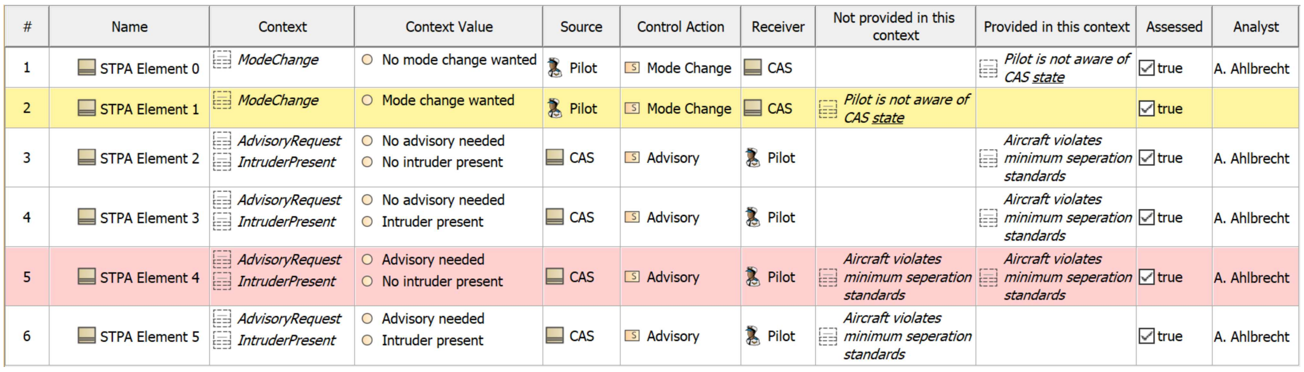
D. SysML-STPA - Identifying Potential Causes

Foundational to the second analysis step, the system and analysis scope were modeled and the first analysis step was executed. Beforehand, it was described in Section III-D and Section III-E, how the analysis block entries can be leveraged for the execution of functions to create UCAs and requirements. Every time one or more hazards were entered in a guideword column in Fig. 11, the automated functions create requirements and UCAs that follow the formal structure of TABLE I. The resulting seven requirements are displayed in Fig. 13, where e.g. *STPA Req. 0* corresponds to the accidental *Pilot* induced *Mode Change* in the *CAS* of Fig. 11 row one.

On the other hand, also one UCA instance is generated for every time one or more hazards were entered in a guideword column. As a result, the UCA in Fig. 14 was created describing the scenario of the sixth row of Fig. 11, where no *Advisory* is provided to the *Pilot*, even though it should have been. In addition to the automatically generated description, also the *Feedback*, *Command*, and *Process Variables* were inserted. This further information is collected and inserted automatically out of the system model shown earlier in this section. Finally, up to 21 causal factor categories were provided from the STPA Handbook [16] in the model and three of them exemplary displayed in Fig. 14. With the help of all the information provided in the UCA instance, the second analysis step of identifying potential causes for the execution of an UCA can be performed. In this example, three causes were provided for the missing application of the *Advisory* of the *CAS*. The first one being that the *CAS* does not provide the required *Advisory* due to an accidental *Mode Change* of the *Pilot*. Following the current SysML-STPA methodology, the potential reasons were documented as *Causal Factor Rational* within the UCA element in Fig. 14. When an assessment of a UCA instance is finished, the *Analyst* changes the *Assessed* attribute to true and inserts his name. Normally, the STPA would be concluded with the development of safety constraints preventing UCA causes from happening. This last step could be achieved with the introduction of safety-driven design decisions, like redundancy or monitoring.

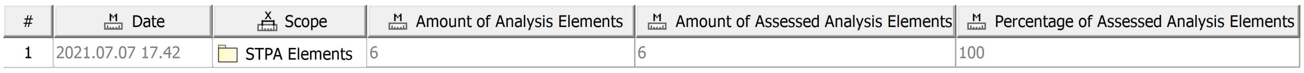
V. Discussion

This paper proposes an integrative approach that, not only enables the execution of the STPA within the SysML, but also allows automating and validating parts of the analysis through formalization. As explained in Section II-D, also other authors did consider the combination of SysML and STPA. Hence, combining the demonstrated SysML-STPA analysis with the



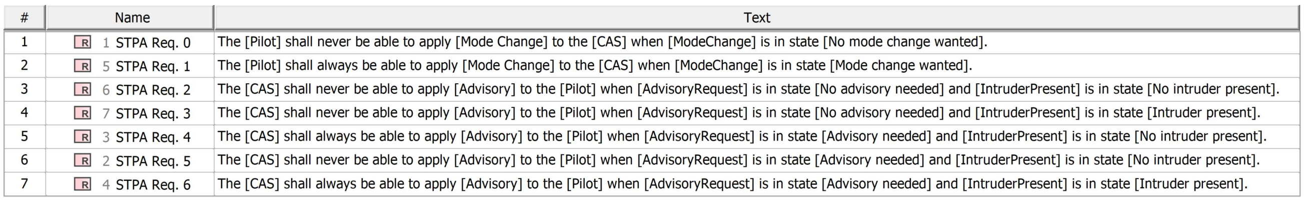
#	Name	Context	Context Value	Source	Control Action	Receiver	Not provided in this context	Provided in this context	Assessed	Analyst
1	STPA Element 0	*ModeChange*	No mode change wanted	Pilot	Mode Change	CAS		*Pilot is not aware of CAS state*	✓ true	A. Ahlbrecht
2	STPA Element 1	*ModeChange*	Mode change wanted	Pilot	Mode Change	CAS	*Pilot is not aware of CAS state*		✓ true	
3	STPA Element 2	*AdvisoryRequest* *IntruderPresent*	No advisory needed No intruder present	CAS	Advisory	Pilot		*Aircraft violates minimum seperation standards*	✓ true	A. Ahlbrecht
4	STPA Element 3	*AdvisoryRequest* *IntruderPresent*	No advisory needed Intruder present	CAS	Advisory	Pilot		*Aircraft violates minimum seperation standards*	✓ true	A. Ahlbrecht
5	STPA Element 4	*AdvisoryRequest* *IntruderPresent*	Advisory needed No intruder present	CAS	Advisory	Pilot	*Aircraft violates minimum seperation standards*	*Aircraft violates minimum seperation standards*	✓ true	A. Ahlbrecht
6	STPA Element 5	*AdvisoryRequest* *IntruderPresent*	Advisory needed Intruder present	CAS	Advisory	Pilot	*Aircraft violates minimum seperation standards*		✓ true	A. Ahlbrecht

Fig. 11: Automatically Generated STPA Hazard Analysis Elements with Validation



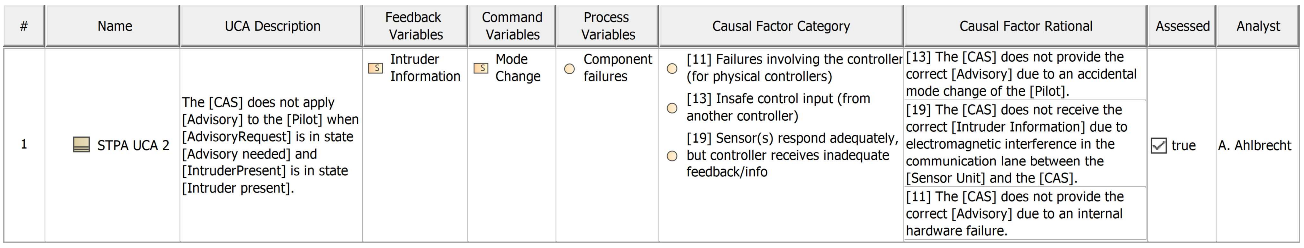
#	Date	Scope	Amount of Analysis Elements	Amount of Assessed Analysis Elements	Percentage of Assessed Analysis Elements
1	2021.07.07 17.42	STPA Elements	6	6	100

Fig. 12: Metric to Track Analysis Progress



#	Name	Text
1	1 STPA Req. 0	The [Pilot] shall never be able to apply [Mode Change] to the [CAS] when [ModeChange] is in state [No mode change wanted].
2	5 STPA Req. 1	The [Pilot] shall always be able to apply [Mode Change] to the [CAS] when [ModeChange] is in state [Mode change wanted].
3	6 STPA Req. 2	The [CAS] shall never be able to apply [Advisory] to the [Pilot] when [AdvisoryRequest] is in state [No advisory needed] and [IntruderPresent] is in state [No intruder present].
4	7 STPA Req. 3	The [CAS] shall never be able to apply [Advisory] to the [Pilot] when [AdvisoryRequest] is in state [No advisory needed] and [IntruderPresent] is in state [Intruder present].
5	3 STPA Req. 4	The [CAS] shall always be able to apply [Advisory] to the [Pilot] when [AdvisoryRequest] is in state [Advisory needed] and [IntruderPresent] is in state [No intruder present].
6	2 STPA Req. 5	The [CAS] shall never be able to apply [Advisory] to the [Pilot] when [AdvisoryRequest] is in state [Advisory needed] and [IntruderPresent] is in state [No intruder present].
7	4 STPA Req. 6	The [CAS] shall always be able to apply [Advisory] to the [Pilot] when [AdvisoryRequest] is in state [Advisory needed] and [IntruderPresent] is in state [Intruder present].

Fig. 13: Automatically Generated Requirements



#	Name	UCA Description	Feedback Variables	Command Variables	Process Variables	Causal Factor Category	Causal Factor Rational	Assessed	Analyst
1	STPA UCA 2	The [CAS] does not apply [Advisory] to the [Pilot] when [AdvisoryRequest] is in state [Advisory needed] and [IntruderPresent] is in state [Intruder present].	Intruder Information	Mode Change	Component failures	[11] Failures involving the controller (for physical controllers) [13] Insafe control input (from another controller) [19] Sensor(s) respond adequately, but controller receives inadequate feedback/info	[13] The [CAS] does not provide the correct [Advisory] due to an accidental mode change of the [Pilot]. [19] The [CAS] does not receive the correct [Intruder Information] due to electromagnetic interference in the communication lane between the [Sensor Unit] and the [CAS]. [11] The [CAS] does not provide the correct [Advisory] due to an internal hardware failure.	✓ true	A. Ahlbrecht

Fig. 14: Automatically Generated UCA Analysis Element

formalized checking of requirements presented in [21] could be an interesting idea for future endeavors.

In general, the SysML-STPA helps to enable a better connection between MBSE developments and safety considerations, since both can use the same model as central artifact. This is possible due to the similar nature of MBSE and STPA, where both are based on systems-theory and use a model as backbone for their approaches. Regarding the combination of development activities and safety approaches in the aviation industry, the ARP standards lead the way. Even though STPA is currently not a direct suggestion within the ARP, it might become more relevant in the future with the development of the AIR6913 [23]. Moreover, it was already demonstrated in the automotive industry how the STPA could be included in the established ISO 26262 process with the analysis of a steer-by-wire system, by the NHTSA [22]. This idea of combining the classical safety approaches with STPA can even be improved with the integration directly within a MBSE development process as demonstrated within this paper.

Despite the advantages that the SysML-STPA approach provides through its automation and reduction of error-prone tasks, current drawbacks need to be considered. One drawback is the need to use the API of the SysML tool. The *Cameo Systems Modeler* tool API is used to enable the creation and changing of elements, so essentially the automated functions except the OCL validation. For this purpose, a standardized implementation could be useful for future applications. This could be more easily implemented with the currently developed second SysML version. Plans for the second version include the development of a standardized API and a better usability [24]. Another interesting discussion point is that, by automating parts of the analysis, a lot of analysis elements might be generated automatically. This could also include unnecessary overhead for the analysis execution and needs to be evaluated in the future.

VI. Conclusion

Conventionally, safety analyses are conducted by a separate team of safety engineers and executed on documents. To enhance the connection between safety assessment and development, this paper suggests using SysML-STPA, which is a formalized STPA integrated into the MBSE development. Both MBSE and STPA are based on systems-theory concepts and a holistic view of system development. Hence, their underlying approaches are compatible and can be integrated well with each other. In addition, it was demonstrated for a CAS use-case that formalization helps to improve the safety assessment in various ways. Demonstrated examples include: automated creation and validation of analysis parts as well as the usage of metrics to keep track of the analysis progress. These implementations proved to be helpful in the analysis execution of the CAS use-case, since error-prone tasks could be automated. For instance, this helps to manage increasing complexity in system development and ensures that engineers are able to focus on important tasks of safety assessment. This paper's combination of MBSE, STPA and formal methods has a lot of potential for improvement in future endeavors with further validation, automation and metric possibilities.

References

[1] INCOSE, *SE Vision 2025*, n.d. [Online]. Available: https://www.incose.org/products-and-publications/se-vision-2025 (visited on 02/15/2021).

[2] ——, *Systems Engineering*, n.d. [Online]. Available: https://www.incose.org/systems-engineering (visited on 02/25/2021).

[3] A. Madni and S. Purohit, "Economic Analysis of Model-Based Systems Engineering," *Systems*, volume 7, Feb. 2019. DOI: 10.3390/systems7010012.

[4] J. Wing, "A Specifier's Introduction to Formal Methods," *Computer*, volume 23, pages 8, 10–22, 24, Oct. 1990. DOI: 10.1109/2.58215.

[5] J. Wang and W. Tepfenhart, *Formal Methods in Computer Science*. Boca Raton, Florida: CRC Press, 2019.

[6] sysml.org, *What is the Systems Modeling Language (SysML)?* n.d. [Online]. Available: https://sysml.org/sysml-faq/what-is-sysml.html (visited on 04/07/2021).

[7] M. Huisman, D. Gurov, and A. Malkis, "Formal Methods: From Academia to Industrial Practice – A Travel Guide," *CoRR*, 2020. eprint: 2002.07279. [Online]. Available: https://arxiv.org/abs/2002.07279.

[8] Object Management Group, *OCL Object Constraint Language*, Specification, 2014.

[9] B. Demuth, *OCL (Object Constraint Language) by Example*, Lecture, Technische Universität Dresden, 2009. [Online]. Available: https://st.inf.tu-dresden.de/files/general/OCLByExampleLecture.pdf (visited on 12/23/2020).

[10] D. Silingas and R. Butleris, "Towards Implementing a Framework for Modeling Software Requirements in MagicDraw UML," *Information technology and control*, volume 38, Jan. 2009.

[11] N. G. Leveson, *Engineering a Safer World, Systems Thinking Applied to Safety*. The MIT Press, 2016.

[12] SAE, *Guidelines for Development of Civil Aircraft and Systems, ARP 4754A*, Standard, 2010.

[13] ——, *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, ARP 4761*, Standard, 1996.

[14] RTCA, *Software Considerations in Airborne Systems and Equipment Certification, RTCA/DO-178C*, Standard, 2011.

[15] ——, *Design Assurance Guidance For Airborne Electronic Hardware, RTCA/DO-254*, Standard, 2000.

[16] N. G. Leveson and J. P. Thomas, *STPA Handbook*. 2018. [Online]. Available: https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf.

[17] J. Thomas, "Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis," PhD, Massachusetts Institute of Technology. [Online]. Available: http://sunnyday.mit.edu/JThomas-Thesis.pdf.

[18] G. Biggs, T. Juknevicius, A. Armonas, and K. Post, "Integrating Safety and Reliability Analysis into MBSE: Overview of the New proposed OMG standard," *INCOSE International Symposium*, volume 28, pages 1322–1336, Jul. 2018. DOI: 10.1002/j.2334-5837.2018.00551.x.

[19] Object Management Group, *Risk Analysis and Assessment Modeling Language*, Beta Version, 2021. [Online]. Available: https://www.omg.org/spec/RAAML/About-RAAML/.

[20] M. Hurley and J. Wankel, *Safety Guided Design Using STPA and Model Based System Engineering (MBSTPA)*, Presentation, 2019. [Online]. Available: http://psas.scripts.mit.edu/home/2019-stamp-workshop-presentations/.

[21] F. Souza, J. Bezerra, C. Hirata, P. de Saqui-Sannes, and L. Apvrille, "Combining STPA with SysML Modeling," in *SysCon2020*, Aug. 2020, pages 1–8. DOI: 10.1109/SysCon47679.2020.9275867.

[22] C. Becker, J. Brewer, D. Arthur, F. Attioui, and L. Yount, "Functional Safety Assessment of a Generic Steer-by-Wire Steering System with Active Steering and Four-Wheel Steering Features," John A. Volpe National Transportation Systems Center (U.S.), Tech. Rep., 2018.

[23] SAE, *Using STPA During Development and Safety Assessment of Civil Aircraft AIR6913*, 2018-02-13. [Online]. Available: https://www.sae.org/standards/content/air6913 (visited on 06/11/2021), Work in Progress.

[24] S. Friedenthal, *SysML v2 Submission Team (SST), SysML v2 Update*, Presentation, 2021. [Online]. Available: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:incose_mbse_iw_2021:iw2021_sysml_v2_2021-01-30-sfriedenthal-reva.pdf (visited on 07/15/2021).