NASA/CR-20205009755

Defining and Reasoning about Model-based Safety Analysis: A Review

Minghui Sun, Cody H. Fleming, and Milena Milich
University of Virginia, Charlottesville, Virginia

March 2021

NASA/ CR-20205009755

Defining and Reasoning about Model-based Safety Analysis: A Review

Minghui Sun, Cody H. Fleming, and Milena Milich
University of Virginia, Charlottesville, Virginia

National Aeronautics and Space Administration

Langley Research Center
Hampton Virginia 23681-2199

Prepared for Langley Research Center
under Cooperative Agreement NNX16AK47A

March 2021

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Table of Contents

1. Introduction

Model-based safety analysis (MBSA) has been around for over two decades. The benefits of MBSA have been well-documented in the literature, such as tackling complexity, introducing Formal Methods to eliminate the ambiguity in the traditional safety analysis, using automation to replace the error-prone manual safety modeling process, and ensuring consistency between the design model and the safety model [1].

However, there is still a lack of consensus on what MBSA even is. Prominent modeling languages such as AADL-EMV2 [2]–[4], AltaRica [5][9], and HipHops [10][11] are generally considered MBSA, which, according to [12], are the only three languages that "have matured beyond the level of research prototypes." However, the question "what makes them MBSA" is left unanswered. For example, do Formal Methods apply to safety analysis MBSA? Does safety analysis even mean the same thing in the context of Formal Methods? The ambiguity has significant implications.

From a System Safety Engineering[*] point of view, without a clear definition and boundary MBSA can quickly become a buzzword that any other disciplines can claim as long as the work uses computer models and is safety-related (e.g., refs. [102] and [103]). This is good because MBSA as an active research topic is enriched by different schools of expertise. However, this also jeopardizes the identity of MBSA as a main research thrust of the System Safety Engineering community. Research development has flourished over the years, with Software Engineering (especially Formal Methods) seeming to have a stronger presence in the MBSA literature. As pointed out by reference [100], most MBSA innovation focuses on model specification notations and/or algorithms for possible manipulations of the models, but very little research is asking whether the safety model is valid, a question that is and will always be at the center of System Safety Engineering.

Therefore, "the major open issue is how to reason about the choice of models, and not so much how to reason about the properties of the models " [13]. Toward this end, this paper reviews MBSA by answering the following three specific research questions *from the perspective of System Safety Engineering*:

(1) What is a minimal set of **defining features** that a work must have to be considered MBSA? Three defining features are identified and can be seen as the negating criteria of MBSA. In other words, if a work satisfies the whole set of the defining features, it is MBSA.
(2) What are the different schools of thoughts, i.e., the **notable patterns**, in the current MBSA literature? This can be seen as the specific ways to implement the defining features. Through the review, we will show different notable patterns along each step of the MBSA process. In the end, we will conclude with the mapping on how the notable patterns implement the defining features.
(3) What are the issues of current MBSA practice, and what are the suggestions moving forward from the perspective of System Safety Engineering?

We put forward the main findings here first and will discuss in detail at the end of this paper:

[*]System Safety Engineering is a discipline to identify hazards and then to eliminate the hazards or reduce the associated risks when the hazards cannot be eliminated [124].

- There is a lack of emphasis on deductive safety analysis in MBSA and the deductive analysis cannot and should not be automated.
- There is a lack of "hard facts" to anchor safety modeling as those fundamental laws (e.g., fluid dynamics and heater transfer) in other scientific modeling communities. This makes it difficult in safety modeling to make explicit decisions about abstraction, i.e., what is (and is not) included in the safety model.
- Automation is closely related to MBSA, but automation does not necessarily ensure high quality safety analysis. Instead, it might give a false sense of complacency that compromises the trustworthiness of safety assurance.
- There is a tradeoff between the specificity and flexibility of a modeling language.

Finally, a review paper must have a set scope of papers to review. However, a MBSA review does not have that privilege because MBSA is ill-defined. Therefore, constraints must be added as the minimal assumptions to establish at least a broad scope of MBSA to start with. For this reason, we broadly assert that **"MBSA is an application of model-based design to hazard analysis.**" Because there is no consensus on an exact definition of MBSA, it is difficult to find concise, direct evidence to support this assertion. However, this assertion is consistent with one of the MBSA seminal works [14] which says MBSA is an extension of safety analysis to model-based design. More importantly, our confidence in beginning with this assertion is in its broadness, meaning MBSA, however defined, can only be a subset of it. We start from this assertation and refine all the way down to a minimal set of features that any MBSA work *must* have and a set of notable patterns that a MBSA work *might* have. In this way, we believe our approach is valid in terms of not missing characteristics by starting off with a too narrow view. In section 2, a general **model-based analysis (MBA)** process is derived based on **model-based design (MBD)**. MBSA follows this MBA process with a specific purpose of safety analysis. In other words, MBSA is an instance of MBA.

In sections 3, 4, 5, and 6, for each step identified in the MBA process, detailed discussions are conducted about the activities that have to take place specifically for MBSA. The discussions are tasked with two goals: identifying the defining features of MBSA and describing the notable patterns of MBSA.

In section 7, a discussion is conducted for the most important future directions of MBSA.

2. Setting the Stage: A General Process of MBA

The MBA process described in this paper is derived from the MBD process, as MBA itself is not a widely used concept. In this section we start with MBD to provide a context for MBA and then zoom in to MBA to further provide a context for MBSA later.

2.1 The MBD Process

First of all, we are aware that MBD is an overloaded term, as numerous papers tried to differentiate it from "model-driven engineering," "model-driven design," and "model-driven architecture" [104] [105]. It is not our intention to define these terms. However, we need a clear understanding of MBD to conduct meaningful discussion about the body of literature pertaining to (or not pertaining to) MBSA. Hence, we adopt the process proposed by [106] as the "ground truth" of our discussion. The left of figure 1 shows the proposed MBD process which can be mapped to the detailed 10 steps of reference [106] on the right. Two loops, an inner loop and an outer loop, are identified for the MBD process. We explain all the involved steps in this section and will zoom in on the inner loop in the next section.

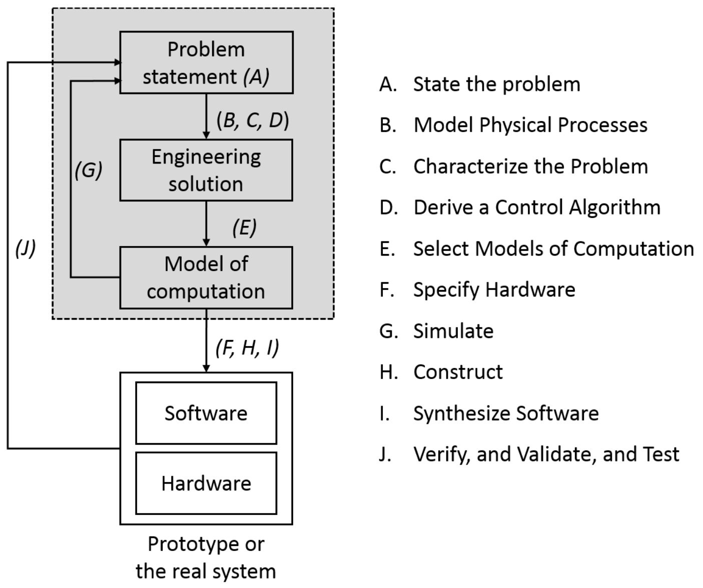

Figure 1. The MBD process. The arrows on the left can be mapped to the steps to the right (adopted from ref. [106]). The inner loop (shaded area) is the model-based analysis.

An MBD process starts with a *problem statement* (Step A) which details the goal of the design activities, such as a set of functions with performance requirements and safety constraints.

Second, *an engineering solution* is derived based the engineer's domain knowledge. Although three independent steps (Steps B, C, and D) are identified by reference [106], we acknowledge this process in reality can be quite fuzzy, because the engineer's thought process varies from person to person, depending on their own expertise and nature of the problem at hand. Nevertheless, we prescribe the process always ends with a solution (valid or not) to the stated problem from the perspective of the specific engineering discipline.

Third, a modeling language is selected to faithfully represent the engineering solution with a *model of computation* (Step E). Compared to a simulation model that is created to loosely "get a feel" of the proposed engineering solution, the model in MBD is a faithful computational copy of the engineering solution and hence used as the primary artifact [114]. Models in MBD are directly evolved into full-fledged implementations without changing the engineering medium, tools, or methods [107]. Specifications and even software code of the implementation are (automatically) derived from the model [108][134]. Therefore, the modeling language must be fully equipped to express the engineering solution.

Fourth, the model of computation is automatically analyzed to gain a required level of confidence of the engineering solution before building the prototype or the real system (Step G). This step closes the inner loop. Analysis techniques with different level of mathematical rigor, such as simulation and Formal Methods, are usually required in accordance with the required level of confidence. Obviously, this not only begs the question of the availability of the tool support for the analysis, which itself is an entire research area, but also whether the semantics of the modeling language can be formalized in a way that tools can be developed for the desired automated analysis.

Fifth, specification is derived from the model of computation (Step F); hardware is constructed in accordance with the specification (Step H), and software is developed or automatically generated from the model and synthesized with the hardware (Step I). This is what makes the "model" in MBD different from models for simulation because (1) specifications for constructing a prototype of the real system are derived from the model, for example a CATIA model can be used for both structural analysis and directly as 3D drawings for manufacture [110], and (2) software code sometimes can even be automatically generated from it, which is the perhaps the main reason for MBD's growth of popularity in the first place [111].

Finally, Step J closes the outer loop. The prototype or the real system is verified, validated, and tested against the problem statement made at the beginning of the MBD process.

Clearly, as shown in figure 1, an MBD process consists of an inner loop and an outer loop, where the former is about modeling and analysis, and the latter is about construction (automated or not) and testing. As defined by reference [112], a hazard analysis is "the process of identifying hazards and their potential causal factors." Although it is a highly iterative process, meaning verification is still required after the construction (e.g., the SSA process of ref. [113]), the hazard analysis mostly happens during the inner-loop so that safety-critical decision can be made early before construction of the prototype or the real system.

Therefore, for MBSA, we mainly focus on the inner loop, which is the shaded area in figure 1, and we call this inner loop "model-based analysis" (MBA).

2.2 The MBA Process

In this section, we take a closer look at the inner loop of figure 1 to derive a more general MBA process. As shown in figure 2, the artifacts on the righthand side are adopted from figure 1. The activities to generate these artifacts are expressed in more general terms, and the supports for the activities are displayed on the left-hand side, both of which will be explained in great detail in this section.

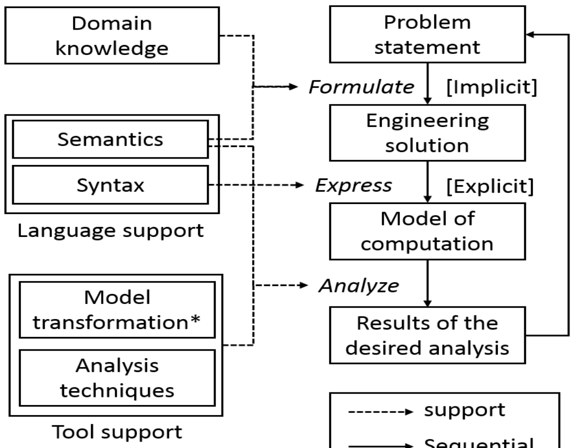

Figure 2. The MBA process, where the artifacts on the right-hand side are adopted from the inner loop of figure 1 and the left-hand side are the supports for the respective activities.

First, from the perspective of an application engineer, the MBA process consists of three steps: "formulate," "express," and "analyze" (the right workflow in figure 2).

Step 1. Given a design problem, the engineers first use their domain knowledge to formulate an engineering solution to that problem. Informally, the engineering solution is the engineer's understanding of the real behavior of the system-to-be-built and his/her decisions about what to capture from this "real" behavior by applying his/her domain knowledge. Although the resulting engineering solution has to be represented eventually in a certain modeling language, and any modeling language has a limit of expressiveness, *in theory* this formulate process has to be language-neutral, meaning solely determined by applying the domain knowledge, not limited by the modeling language. For example, the engineering solution of a control system should solely be determined by the physical dynamics and preferred control policy rather than the modeling language such as Matlab and Modelica.

However, *in reality*, this formulate process is more complicated. As explained in section 2.1 about MBD, because primary artifacts such as specification, software code and safety decision are made from this process, the engineering solution has to be faithfully represented by the model of computation, meaning *the modeling language has to be fully equipped to express the engineering solution*. Two possibilities exist for this concern. One is to find the appropriate modeling language after the engineering solution is developed or design a new modeling language if none of the current ones fit. The other is to have one or multiple candidates modeling languages in mind beforehand, use the modeling languages to formulate the engineering solution, and finally pick the most appropriate one. In reality, for most engineers who use models to develop their own systems, the latter is most dominant. In fact, the structured semantics of a modeling language can help engineers to perceive the problem and manage the cognitive complexity in the problem-solving process.

This observation has a significant implication because although the formulate process focuses on problem solving and highly relies on the domain knowledge, the resulting engineering solution has to be practical enough so that it can be expressed by a modeling language. Hence, two "support" arrows (figure 2) from the domain knowledge and the language semantics go into the engineering solution. We make the following assertion, which is similarly referred to as "the abstraction challenge" in reference [107].

Assertion 1: The modeling language, especially its semantics, has to be able to represent the engineering solution.

Step 2. The engineering solution, already consistent with the semantics of the modeling language, is mapped accordingly to the language syntax and eventually yields the model of computation. We call this process "express" because it is simply a "faithful" representation of a well formulated engineering solution (in whatever form) to a well-defined explicit model in the computer. Note that, this process is only to "express." Information shall neither be subtracted from the engineering solution nor added to it.

Step 3. The model of computation is "analyzed" automatically, so that a required level of confidence can be established. Specifically, different levels of confidence require different analysis techniques such as step-wise trace demonstration, stochastic simulation and model checking, which in turn has an implication on the formalism of the modeling language. In other words, analysis techniques must be available for the desired analysis with the selected language, thus the arrow from the "semantics" to the "analysis" in figure 2. Therefore, we make the following assertion, which is similarly referred to as "the formality challenge" in reference [107].

Assertion 2: The modeling language, especially its semantics, has to be analyzable by computer programs for the desired analysis.

From the perspective of the methodological support (the left workflow in figure 2), it includes language support and tool support. The modeling language, particularly its semantics, has to be expressive enough to fully represent the engineering solution (Assertion 1) and techniques and tools must be developed for the modeling language for the desired analysis (Assertion 2). Obviously, the semantics of the modeling language plays a center role here. As shown in figure 3, it affects the formulation of the engineering solution (the upper loop) and the feasibility of the desired analysis (the lower loop).

For language support, we further assert that the semantics of a modeling language in the MBA process must simultaneously have a *context-specific* **modeling construct** for the engineer to represent the engineering solution (the upper loop) and a *context-free* **mathematical construct** for the computer to conduct the desired analysis (the lower loop). For example, a Simulink block at the same time has both a specific engineering meaning at the front end and a context-free purely mathematical expression at the back-end, such as a high-pass filter has a modeling construct that means signals below a cutoff frequency are attenuated and a mathematical construct that is usually represented by a first-order mathematical transfer function. The transition in a Markov process can mean at the same time both the triggering of a failure event (i.e., the modeling construct) and a context-free Poisson process (i.e., the mathematical construct). In fact, the modeling construct and the mathematical construct are consistent with the concepts of a "pragmatic model" and a "formal model" in reference [15]. The difference is that the modeling construct and the mathematical construct are two *aspects* of the same model rather than two different *types* of model as argued in reference [15].

There is usually a unique mapping between the modeling construct and the mathematical construct, to translate between the context-specific concepts and the context-free mathematical expression. Depending on the specific situations, this translation is bi-directional. The engineer can use an appropriate modeling construct directly to formulate the engineering solution, such as modeling in Simulink. The resulting model is translated (usually automatically by the modeling environment) into the context-free mathematical model (i.e., the downward translation) for the desired analysis later. It is also possible the engineer uses the mathematical construct directly to formulate the engineering solution, such as writing state space model in a Matlab file. They then need to reconstruct the modeling construct (i.e., the upward translation) manually from the mathematical construct to link with the context-specific concepts in order to make sure the resulting mathematical model faithfully represents the solution in the specific engineering context. It is worth mentioning that it can be very difficult to formulate the engineering solution directly with the mathematical construct especially for complex systems because manually mathematically modeling a complex system is error prone, and the resulting pure math model is very difficult to communicate and review, which is why modeling with Simulink has gained much more traction than modeling directly with Matlab script files. Therefore, the upward translation is not considered for the rest of the paper.

For tool support, algorithms are developed around the mathematical construct of the modeling language for the desired analysis. Sometimes in order to reuse existing algorithms, "model transformation" algorithms such as AADL to GSPN [115] are developed to translate the mathematical construct of the current modeling language to the formalism that the target algorithms can be applied to.

Although algorithms can be developed for the desired analysis or model transformation, an inappropriate abstraction level of the modeling language can render the desired analysis infeasible. For example, a traditional fault tree cannot generate the critical event sequence for a top-level event, simply because the sequence information is abstracted by the semantics of the traditional fault tree. More information has to be included (hence less abstract), such as the Dynamic Fault

Tree [16] [17], for the desired analysis. This echoes our previous claim that *the semantics of the modeling language not only affect the formulation of the engineering solution, but the feasibility of the desired analysis.*

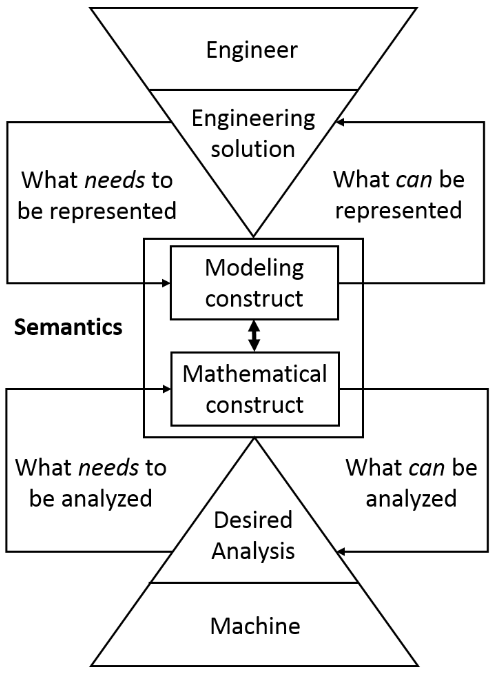

Figure 3. The semantics of the modeling language are at the center of an MBA process in the view of a methodology developer.

To conclude, table 1 is a summary of the MBA process. From the perspective of the domain engineer, the process mainly consists of three activities: formulate, "express," and "analyze." Each of the activities requires different methodological supports, which are essentially application and manipulation of the "modeling construct," "mathematical construct," and "syntax" of the modeling language.

Table 1. The summary of the MBA process.

Domain engineer		**Methodology developer**	
		Language support	**Tool support**
Manual	Formulate	Modeling construct	NA
	Express	Syntax to express the engineering solution	Modeling environment
Automated	Analyze	Mathematical construct to be translated.	Programs for model transformation
		Mathematical construct to be analyzed.	Programs for the desired analysis

Finally, we focus this paper on how the modeling language supports the MBSA process rather than how the modeling language is supported and implemented by tools, as the former is closer to the System Safety Engineering community while the latter is mostly Software Engineering. For this reason, we adapt the MBA process of figure 3 into figure 4 below, with a more explicit depiction of the relationship between the MBA process and the modeling language. The rest of the paper will follow closely with figure 4.

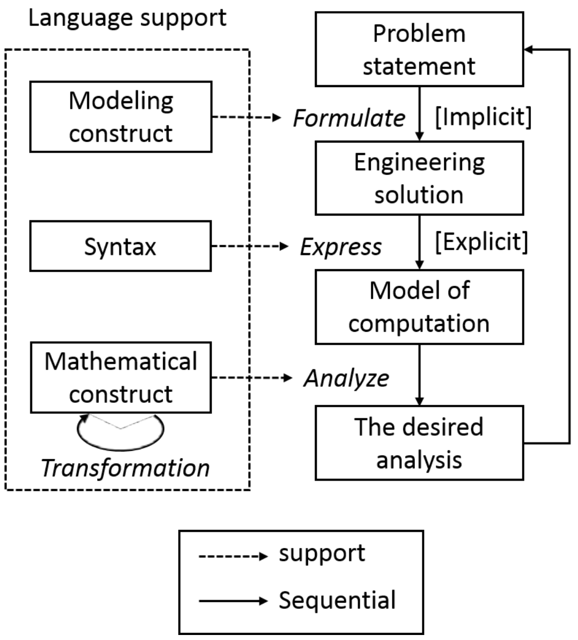

Figure 4. The MBA process and the language support. While the analytical process represents a series of (iterative) steps [right], it is supported by particular choices with respect to the modeling language [left]. The discussion of MBSA will follow closely with this process.

3 MBSA: Problem Statement

Figure 4 is a general MBA process and MBSA is an application of this process to the domain of hazard analysis. In other words, MBSA follows the MBA process and specifies each MBA step for hazard analysis. In the rest of the paper, we will review MBSA following the steps in the MBA process with the goal to identify the *defining features* and *notable patterns* for MBSA. We focus this section on the "problem statement."

3.1 The Defining Feature: The Fail-Safe Property

The problem statement determines the goal and the scope of an analysis. As asserted in section 1, the goal of MBSA is broadly hazard analysis. Although there is no definitive prescription about the scope of hazard analysis in System Safety Engineering, we posit that MBSA, like any other system safety analysis, is mostly concerned with hazard related to system function. In other words, a work that uses a "model-based" approach and has safety implications does not necessarily make it MBSA. It has to focus on the violation of functional safety [116][117] by analyzing the dynamic behavior of the system. For example, the "model-based" approach is also applied to analyzing hazards in other disciplines such as structural safety [118]–[120] and occupational safety [121][122], but because they do not address hazard from the perspective of system function, they are trivially ruled out for MBSA.

Furthermore, according to reference [123], hazard analysis is performed to identify hazards, hazard effects, and hazard causal factors. This definition is widely accepted in the community. For example, aviation industry specifies three prominent tasks [113]: function hazard assessment (FHA), preliminary system safety assessment (PSSA) and system safety assessment (SSA), where FHA sets up the safety requirements by identifying the potential hazards and their effects, PSSA validates the system architecture by identifying the possible causal factor*s*, and SSA verifies that the risk of the causal factors leading to the hazard is acceptable.

Clearly, the overarching goals of hazard analysis are achieved through the following three tasks:

- Task (1) determines the safety requirement by hazard identification.
- Task (2) identifies the possible causal factors through a *deductive* analysis.
- Task (3) makes sure the system is still safe in the presence of the causal factors through an *inductive* analysis.

In the safety community, Task (1) is usually referred to as "hazard identification" [124], and safety analysis usually refers to Tasks (2) and (3). Methodologically, Task (1) is different from Tasks (2) and (3) in both the goals they seek to achieve and methods they follow. Therefore, we do not include hazard identification as a potential area of MBSA. It is worth mentioning that there are works (mostly based on UML/SYSML) to partially automate the hazard identification process [18]–[22]. Neither UML/SYSML language nor the partial automation can change the fact that hazard identification is generally not considered safety analysis in the System Safety Engineering community. As pointed out by reference [23], "the focus of classical safety analysis techniques lies on supporting the reasoning of possible failures and on recording the causal relationships in failure events." Rather, hazard identification is the input of a safety analysis, but not the safety analysis itself. Therefore, we do not include in the scope of MBSA.

In terms of Task (2) and Task (3), although they are fundamentally different analyses, they both share the same goal to ensure the system is safe in the presence the causal factors. While there are many types of causal factors, such as design error, manufacture defect, human error, and component failure, a minimal requirement for a safety analysis is that it must address component failure. In other words, the goal of a safety analysis must at least involve proving the system is **fail-safe**. A defining feature of MBSA is that it must at least be able to argue whether a system is fail-safe. In fact, most MBSA works focus on modeling component failure with an exception of the EAST-ADL framework [24][25], where the process faults (systematic faults such as design, implementation, installation, operation, and overstress faults) are considered in the modeling semantics. However, it is unclear from the literature how process faults are identified, mitigated, and implemented by the framework.

3.2 The Notable Pattern: The Inductive Analysis

Tasks (2) and (3) both have the fail-safe feature but Task (2) is a "deductive analysis" (also called top-down [26][27] or effect-to-cause [28]), and Task (3) is an "inductive safety analysis" (also called bottom-up [26] [27] or cause-to-effect [28]). At this point, **"inductive analysis"** is widely practiced in the MBSA literature, which makes it a notable pattern. Note that the lack of emphasis on the "deductive analysis" has a significant impact on the quality of the "inductive analysis."

3.2.1 The Inductive Analysis

A notable pattern of MBSA is that most works in the literature are inductive analysis. The inductive analysis is a notable pattern rather than a defining feature, meaning MBSA does not necessarily have to be an inductive analysis. But in this section, we focus on this pattern of MBSA.

We already knew that an inductive safety analysis is a bottom-up safety analysis. A bottom-up analysis in the context of MBA will, given a set of properties (or requirements) and a model, verify whether the model satisfies the given requirements. In the context of MBSA, there are generally two types of properties (functional property and safety-critical property) and two types of models (nominal model and off-nominal model). As shown in figure 5, a property-model combination yields four types of inductive analysis.

Arrow 1 verifies the "goodness" of the design. The intended function has to be achieved by the designed behavior in nominal conditions. This is the foundation of all other types of analysis.

Arrow 2 verifies that the designed behavior in nominal conditions will not lead to hazardous situations. For example, reference [103] conducted a series of safety assessments to prove that all the possible trajectories of autonomous cars are not in conflict by using reachability analysis.

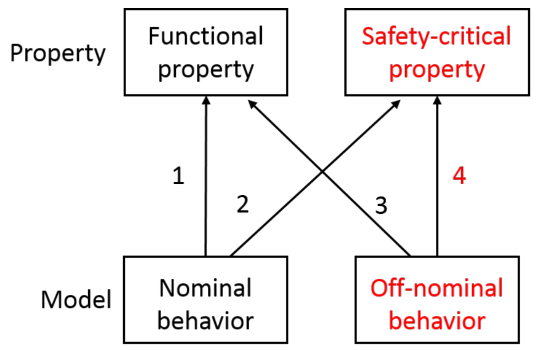

Figure 5. Four types of inductive safety analysis. The start of the arrow is verified against the end of the arrow. No. 4 is one of the defining features of MBSA.

Arrow 3 verifies the "fail-operational" [126] property of a system design, i.e., the desired function is still achievable even in the case of device malfunction. This is a subject of robustness analysis [127], a dependability property that is closely related to safety.

Arrow 4 verifies the fail-safe property. As explained in the previous section, this is one of the minimal requirements for any work to be considered as MBSA.

It is worth mentioning that all four arrows can be implemented by Formal Methods. In Model Checking, a safety property is even explicitly defined in the methodology. However, if a formal analysis does not include Arrow 4, it is *not* MBSA, even if a safety-critical property is verified against.

3.2.2 The Deductive Analysis

The deductive analysis is *not* the notable pattern of MBSA as very few works address it. But the lack of emphasis on the "deductive analysis" has a significant impact on the quality of the current inductive MBSA practice. More specifically, inductive analysis can only work with a set scope of system, while it relies on the deductive analysis to set the scope.

Out-scope. Compared to inductive analysis, where the scope of the system is taken as given, the deductive analysis takes a more holistic view and can identify contributing factors that can be hard to capture if no structured method is provided. Reference [29] calls for a systematic approach to achieve confidence in the completeness of an analysis, but as pointed out by reference [30], completeness of the causal factors may only be proven with respect to those captured: "If a failure mode is not even part of the formal model, then it is impossible to reason about it. But, finding a complete set of failure modes for a given component is not an easy task." Current MBSA practices do not address how the failure modes are derived in the first place.

Furthermore, the inductive analysis focuses too quickly on failures. It is true that failures at the device level are usually well-studied and well-recorded in the industry. A complete list of failure modes of all the devices is presumably accessible from the industry record. This is perhaps why most MBSA works are failure oriented. However not all the devices are well-studied especially for those newly designed. The 737MAX accident perfectly exemplifies that a new system or a new feature (i.e., the MCAS system) can have surprising behaviors that may cause catastrophic accidents. More importantly, it has been widely accepted that hazards can also be caused by non-failures [125]. The inductive safety analysis needs a deductive approach such as STAMP-STPA [128] and FRAM [129] to out-scope the analysis boundary so that other casual factors and unintended interactions are also captured.

Down-scope. Another benefit of deductive analysis is down-scope. The combination of all the possible device failures can dramatically increase the complexity of the inductive analysis. So far, this problem is mitigated by abstraction. But abstraction has a price. The more abstract a model is, the less precise the result will be (see ref. [31] for the comparisons). Furthermore, as pointed out by reference [32], "the combinatorial diversity of each plausible (fault) event interacting with each other set of events within and without the system makes bottom-up analysis intractable, so heuristics on system behavior need to be employed to *narrow* the search space of critical scenarios." This is why reference [30] claims the completeness of the failure modes but only at a lower device level. Clearly, inductive analysis alone is not sustainable to analyze a complex system. It has to be complemented with a deductive analysis, as a deductive analysis only identifies the casual factors that are relevant to the hazard of interest, which significantly reduces the complexity of the analysis.

4 MBSA: Engineering Solution (The Global Effect)

4.1 Engineering Solution: The Off-Nominal Behavior

We move to the "engineering solution" of figure 4 now. Although not unique to MBA (or even engineering), the formulation of engineering solutions is intended to solve the stated problem. In the context of MBSA, the problem statement is to argue whether a system is fail-safe, which, as shown in figure 5, requires a set of safety-critical properties and a model of off-nominal behavior. The properties are usually derived from the hazard identification process, which is not part of MBSA and hence for the purposes of this paper we simply assume the existence of the set of properties, and do not discuss how it is created. The engineering solution in the context of MSBA is the off-nominal behavior.

We reiterate that the engineering solution in MBA focuses on depicting the actual behavior of the system-to-be-built with as much fidelity as possible and the formulate process is language-neutral. Similarly, regardless of the modeling language, to formulate the off-nominal behavior is to decide how a fault happens in reality and its effect on both the local component and other components. This is consistent with the three propositions proposed by reference [33] for any fault logic modeling. Specifically, it includes defining the following three subprocesses:

- Causal scenario: the condition for a component fault to happen;
- Local effect: the effect of the fault on its respective component;

- Global effect: how the local effect affects the system behavior as a whole.

In this section, we will show how the specific ways to implement the **global effect subprocess** lead to a defining feature of MBSA. The casual scenario and the local effect subprocesses will be addressed in the next section.

4.2 The Defining Feature: Architecture Consistency

It is usually argued in the MBSA literature that MBSA adds value to the current safety engineering practices at the following aspects:

(1) Handling the increasing complexity of safety-critical system.
(2) Integrating the design view and the safety view.
(3) Finding design shortcomings and flaws early.
(4) Reusing previously developed artifacts.
(5) Introducing automation to reduce time and cost and improve quality.
(6) Structuring unstructured information.

However, these benefits are also claimed by the general MBD community [105][109][114][130][131][133] except (2). As argued by reference [132], MBD is just a *tool* for realizing the integration among the different domain of systems and must be supported by an integrated design *methodology*. MBSA is exactly such a methodology to support the view integration of design and safety. This is also consistent with the claim made by one of the seminal MBSA works [34] that MBSA is about maintaining the **consistency between the design model and the safety model**, and this consistency is unique to MBSA compared with traditional safety analysis.

However, the *component fault* and the *local effect* subprocesses have no logical consistency with the nominal function of the component, because while the fault of a component might eventually affect the intended function, how it happens and how the fault affects the component are not determined by how the component is supposed to work in the first place. It is worth mentioning that in some works (such as the Generic Failure Model Library [35] [36]), a fault library is developed to associate each component with a fault model. We argue the association is not integration because association is enabled by numerous reuses of the same component in the same (or at least very similar) systems. In fact, the association is a result of the MBD approach being able to structure unstructured information and use computer models as media to store institutional knowledge for reuse, i.e., benefit (4) and (6) discussed above.

In fact, it is the specific ways to implement the *global effect* subprocess that leads to the consistency between the design model and safety model. The global effect is to model how the fault effect of an individual component affects the system behavior as a whole. While the fault effect of an individual component varies from case to case, the propagation path does have (at least partially) resemblance with the interaction path in the nominal behavior. This resemblance is the basis of the consistency between the design model and safety model. Because the propagation path and the interaction path are the architectures of the safety model and the design model, we call the consistency **Architecture Consistency** in this paper, and it is one of the defining features of MBSA.

4.3 The Notable Pattern of Architecture Consistency

Three different ways of achieving Architecture Consistency is found in the MBSA literature (figure 7). This classification is an extension of the model provenance in reference [25] and the ESACS project in references [37] and [[38].

But first, an important distinction has to be made between "views" and "behaviors" (figure 6). The nominal view means the semantics adopted to describe the nominal behavior (Arrow 1) are the same as the off-nominal view (Arrow 4). However, an off-nominal behavior can also be described in a nominal view (Arrow 2). For example, a current overflow can be called out by off-nominal semantics as a faulty state. It can also be simply represented, as any other nominal current is measured, by a number of Amperes that happens to be higher than intended by the designer. Being off-nominal does not change the nature of the current, therefore it can still be represented using the nominal semantics. Similarly, the nominal behavior can also be represented using off-nominal semantics (Arrow 3). For example, off-nominal semantics can be equipped with all the complex off-nominal behaviors, but a "healthy" state can be all it needs to represent all the nominal behaviors.

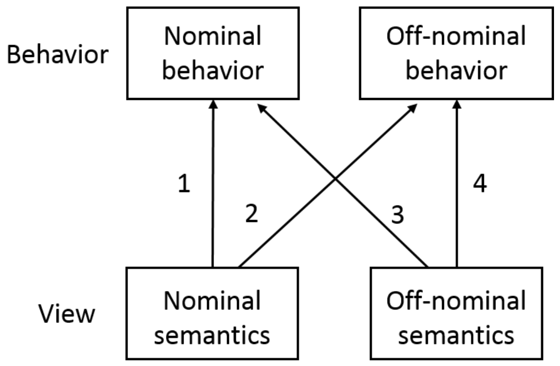

Figure 6. Nominal behaviors can be represented in both the nominal and the off-nominal views. Same for the off-nominal behavior. The arrow means "represent" in this figure.

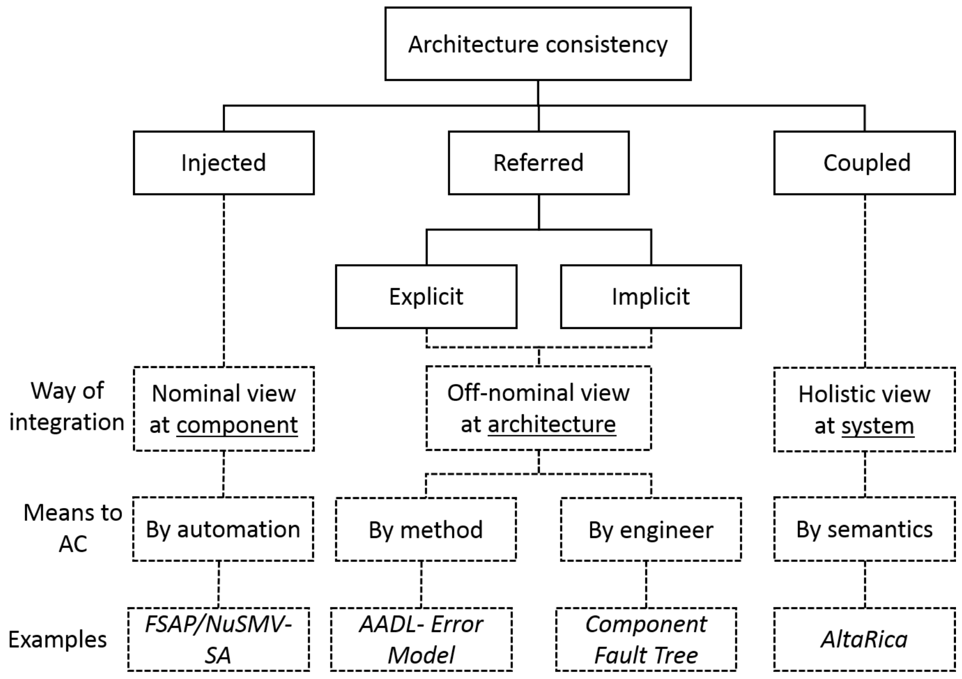

Figure 7. Notable patterns to achieve Architecture Consistency.

Now, we are ready to explain the notable patterns to achieve Architecture Consistency (figure 7).

The first class injects faults into the components of the design model. The off-nominal behavior of the individual component is described in the *nominal view* and then is injected into the design model. No information of the propagation paths of the off-nominal behavior is defined. Instead, the propagation paths are generated automatically by reusing the interaction paths of the nominal behavior. This leads to the Architecture Consistency of the design model and the safety model. From the perspective of the safety engineer, the integration is achieved at the *component* level, as the generation of the architecture of the safety model is shielded from him/her by the *automation*. A typical example is the FSAP/NuSMV-SA language [39].

The second class involves formulating the off-nominal behavior using a dedicated *off-nominal view*. Compared to the injected class, off-nominal behaviors are explicitly called out as a set of behaviors that are separate from the nominal behaviors. The architecture of the design model is manually referred when defining the off-nominal behavior for each individual component. Architecture Consistency is hence achieved through the shard architecture. This class is also called "architecture-based evaluation methodologies" in reference [40]. Two subgroups exist depending on whether the design model is explicitly required for the construction of the safety model, or only has to be implicitly referred.

For the "explicit" case, an explicit design model is required. For each component, the off-nominal behavior and/or the nominal behavior are defined by the engineer using the dedicated off-nominal semantics. The propagation paths between the components are then defined manually. Finally, the whole safety model is built (usually automatically) by aggregating all the well-defined components and interactions. The Architecture Consistency is achieved manually, but the *method* and the modeling tool enforces Architecture Consistency by requiring a dedicated component in the off-nominal model for each component in the design model. A typical example is the AADL-EMV2 [41].

The “implicit” case follows the same process as the explicit case. The safety model is organized and constructed in a compositional way [42]. However, it does not require an explicit design model, hence there is no way to enforce the Architecture Consistency except by completely relying on the *engineer’s* discretion. Note that the traditional FTA also does not require an explicit design model, but because it is not a compositional approach no implication of ensuring Architecture Consistency can be made. Many methods belong to this subclass, such as Component Fault Tree [43], Failure Propagation and Transformation Notation (FPTN) [44], Fault Propagation and Transformation Calculus (FPTC) [45] and State Event Fault Trees (SEFTs) [46].

The final class attempts to “couple” both the nominal behavior and the off-nominal behavior in the same model by describing them within by the same semantics. Models are structured in a compositional way; each component contains both the nominal behavior and the off-nominal behavior; the interactions between the components can be both the nominal interaction and the fault propagation. Architecture of both the nominal behavior and off-nominal behavior are aligned with the same compositional structure of the resulting model, which also leads to Architectural Consistency by construction. From the perspective of the safety engineer, this is true integration as the modeling *semantics* ensure the nominal behavior and the off-nominal behavior are weaved organically in the same model. However, to achieve this, the safety engineer has to have a more “*holistic view*” and approach the modeling task at a more comprehensive *system* level. A typical example is the AltaRica modeling language [1].

Finally, we are aware there are a handful of works claiming to be MBSA that do not show a clear way to achieve Architecture Consistency, such as reference [47]. This lack of Architecture Consistency is mainly caused by a loose usage of the terminology in these works. In fact, reference [47] even defines MBSA as an approach in which the system and safety engineers share a common system model created using a model-based development process, which is consistent with our definition.

4.4 Comparing the Notable Patterns

As shown in figure 7, the four different classes (counting the two subclasses) of Architecture Consistency have different ways of integration and different means to achieve Architecture Consistency. In this section, we will show the different ways integrations lead to different levels of flexibility to describe complex (off-nominal) behaviors and interactions, and that the different means to achieve Architecture Consistency can lead to different efforts from humans and automation. Therefore, we compare the four classes from three perspectives: *flexibility for complex behavior*, *human efforts,* and *automation efforts*.

4.4.1 Flexibility for Complex Behavior

For the injected class, the off-nominal behavior is described with the nominal semantics. However, it is known that faults can create new component behaviors (i.e., the unintended behaviors) and new interactions between them (i.e., the unintended interactions). These new behaviors and interactions cannot be planned in advance in the nominal model. They cannot be addressed in the injected way unless the original design model is adapted accordingly. However changing the original design model just for the construction of the safety model not only violates the current industry practices, but also defeats the purpose of MBSA, because in this way the safety model is only consistent with a model that is different from the original design model, so the inconsistency is guaranteed by construction. Similar argument can also be found in [48].

Compared with the injected class, the referred (including both implicit and explicit) class has more flexibility to define the new behaviors and interactions, as the off-nominal model is defined in a stand-alone model using dedicated off-nominal semantics. However, the off-nominal view taken by this class limits the options of modeling the nominal behaviors. For example, instead of modeling the real dynamics of the nominal behavior, it is usually abstracted as some discrete modes, such as “working” and “healthy” [49] [50]; in some cases, the nominal behavior is even completely left out of the off-nominal model, such as HipHops and Component Fault Tree. This significantly reduces the flexibility of modeling especially those faults whose presence or effects depend on specific nominal conditions.

Finally, the coupled class takes a holistic view to include both nominal behavior and off-nominal behavior in the same model. On the one hand, it gives the safety engineers more flexibility to describe the behaviors that they identify from the specific domain; on the other hand, however, it relies on the safety engineers to make the important modeling decisions, such as decomposition and abstraction. Nevertheless, this class indeed has the greatest flexibility in describing complex behaviors.

In summary, we conclude **coupled > implicit = explicit > injected** in terms of the flexibility to describe complex behaviors.

4.4.2 Human Effort

Different classes of Architecture Consistency require different levels of human effort to achieve Architecture Consistency. Note that the effort here means the work to specifically achieve Architecture Consistency rather than the overall manual efforts in constructing the safety model. The manual efforts in the construction of the safety model will be addressed in the next subsection from the perspective of the overall automation support for that purpose.

Furthermore, “human” here means both the methodology developer who sets out the working process and develops tool support, and the safety engineer who follows the working process to actually execute the MBSA analysis. Two metrics are used to compare the human efforts: the primary one is how much the methodology relies on the efforts of the safety engineer to achieve Architecture Consistency, and the secondary one is how much effort is required from the methodology developer to develop automation for Architecture Consistency. For each class, the primary standard is compared first; if they are at the same level, then the secondary standard is compared.

In this regard, the coupled class requires the least effort from the safety engineer to achieve Architecture Consistency, because the modeling language inherently guarantees it between the design view and safety view. No extra effort is required from the safety engineer, and no automation is needed, specifically for Architecture Consistency.

Second is the injected class where Architecture Consistency is ensured by the automation. No extra effort is required from the safety engineer but automation has to be developed by the methodology developer beforehand to enable the derivation of the safety model from the design model. Using the metrics above, the injected class requires less human effort, specifically for Architecture Consistency.

Third is the “referred” class where Architecture Consistency is achieved manually by the safety engineer. However, compared to the implicit class, the explicit class has a specific modeling process for the safety engineer to follow and the consistency usually can be examined by automation, while the implicit class instead is completely reliant on the discretion of the safety engineer. Therefore, the explicit class requires less effort from the safety engineer than the implicit class, specifically for Architecture Consistency.

In summary, we conclude **coupled > injected > explicit > implicit** in terms of the least requirement of human effort.

4.4.3 Automation

In general, automation serves two purposes in supporting the construction of the safety model. One is achieving Architecture Consistency, and the other is aggregating a complete safety model from the individual components and the interactions among them.

In this regard, the injected class has the most automation support, because both Architecture Consistency and the aggregation are supported by automation. The second is the explicit class, because it usually provides automation support to examine Architecture Consistency and build the complete safety model from the defined off-nominal components and the propagation paths between them.

Finally, the implicit and the coupled classes do not have the automation support to examine the Architecture Consistency because no explicit design model is available in these two cases. The aggregation of the safety model can be automated from the off-nominal components and the propagation paths between them, such as in reference [51]. Therefore, they are at the same level of automation support.

In summary, we conclude **injected** > **explicit** > **implicit** > **coupled** in terms of overall automation support.

4.4.4 Observation

The evaluation results are summarized in table 2.

Table 2. Evaluation of the four different classes of Architecture Consistency.

Perspective	**Results**
Flexibility	coupled > implicit = explicit > injected
Human efforts for Architecture Consistency	coupled > injected > explicit > implicit
Automation	injected > explicit > implicit = coupled

Architecture Consistency and automation. Based on the information in table 2, automation is neither a sufficient condition nor a necessary condition for Architecture Consistency. This defeats the general belief in the MBSA community that the automation leads to the consistency between the design model and safety model. In fact, only Architecture Consistency in the injected class is achieved by automation; Architecture Consistency in the explicit class can only be examined by automation and in other classes are completely achieved by humans. Furthermore, automation can be used for automatic aggregation, but aggregation is a different concept from Architecture Consistency. No correlation can be made between automatic aggregation and Architecture Consistency. *Therefore, while Architecture Consistency is a defining feature of MBSA, it is inaccurate to equate MBSA with automatic construction of safety model.*

The Pareto tradeoff. A simple pareto analysis on the evaluation results in table 2 reveals that no single class of MBSA is globally superior or inferior. The four different classes perform differently based on the specific perspective taken.

However, with the three identified standards, the implicit class is dominated by the explicit class. Compared to the explicit class, the implicit class requires more involvement from the safety engineer to maintain the architectural consistency, receives less automation support as it does not have an explicit design model, and has about the same flexibility in modeling complex off-nominal behavior. Therefore, if only the three standards are considered for the tradeoff, the implicit class should never be selected over the explicit.

Finally, the "Flexibility" row of table 2 has the opposite ranking to the "Automation" row. The coupled class can model more complex behaviors, but the safety engineer has to be responsible for the most modeling work as it is the least supported by automation. While the injected class can model less complex behaviors, the automation can take care of the most modeling activities including Architecture Consistency and aggregation. The referred class is in between and, to a certain extent, can be seen as a balance of the trade-off between flexibility and automation.

5 MBSA: Engineering Solution (The Causal Scenario and the Local Effect)

To reiterate, the engineering solution in the context of MBSA is the depiction of the actual off-nominal behavior with as much fidelity as possible, which is supposed to be language neutral. In this section, we focus on the other two language-neutral subprocesses to formulate the off-nominal behavior, the causal scenario, and the local effect. Because the two subprocesses describe how a fault develops in a component and affects the component, we call them together as a "*component fault process*" for simplicity.

5.1 A Framework for the Component Fault Process

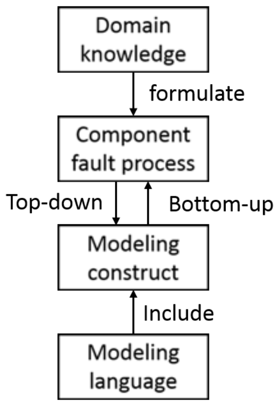

Figure 8. Formulating the component fault process top down and bottom up.

Ideally, the component fault process is supposed to depict the actual off-nominal behavior based on the safety engineer's domain knowledge. Then a MBSA modeling language can be selected to represent the component fault process in the safety model. This is the top-down process in figure 8 and is widely practiced in the scientific modeling community; however, this is challenging for safety engineering. In the traditional scientific modeling community, there are usually a set of fundamental laws and principles to describe how the subject under study works in the real world. These laws and principles are hard facts that the engineers can use to formulate a language-neutral process. However, the authors are not aware of any such hard facts or if it is even possible to ask for such hard facts in the safety engineering community. Admittedly, there are a variety of high-level accident models [112] [129], but none of them are specific enough for a safety engineer to formulate how a fault develops within a component. As a result, this top-down approach relies heavily on the expertise of the safety engineer, which in our opinion makes the safety modeling hard to repeat, to review, and to assure.

Consequently, in practice, many safety engineers rely on the modeling language (specifically the modeling construct) to formulate the component fault process. This is the bottom-up process in figure 8 where the modeling language provides a structure for modeling, reviewing, and assurance. This is problematic, because all modeling languages are (rightfully) an abstraction of the reality (the subprocesses in our context), and this automatically abstracts away the real phenomenon that is not captured by the modeling language. *Using a language correctly does not mean the correct language is selected in the first place.* For example, by selecting FTA, the safety engineers immediately abstract away the fault propagation among the components. They can select AADL-EMV2 instead to capture the propagation, but what else is abstracted away by AADL-EMV2? Most MBSA modeling languages present their own modeling construct without even addressing what real phenomenon they cannot model, and so safety engineers make abstractions already without even knowing what is abstracted away by only deciding what modeling language is going to be used. Abstraction is not the problem here, as a model by definition is an abstraction, but the point is **to make these abstractions explicit**.

Therefore, the problem is that there are no hard facts akin to the scientific modeling community for the component fault process in the safety engineering community. In fact, we doubt whether there will be any a priori principle such as Newton's law that can completely depict the component fault process. However, we take an in-the-middle approach in the paper to solve this problem by proposing a *phenomenon-centric* framework to describe the hard facts of the component fault process. By "phenomenon-centric" we mean it focuses on the real process of how a fault appears, develops, and affects the component. For safety engineers, the framework can be used to guide the formulation of the specific component fault process for their project; for the language developer, the framework can be used as the basis to make explicit decisions about what phenomenon is and is not supported by the language. In section 5.2, inspired by the Ericson's hazard theory [123], we will propose a high-level a priori structure for the framework of the component fault process. Then, in section 5.3, we use the notable patterns of the component fault process identified in the MBSA literature to enrich and, more importantly, to specify the framework.

5.2 The Structure of the Phenomenon-centric Framework

According to Ericson [123], a hazard is comprised of three components: hazardous element (HE), initiating mechanism (IM) and target/threat (T/T). HE is the basic hazardous resource creating the impetus for the hazard; IM is the trigger or initiator event(s) causing the hazard to occur. The IM causes actualization or transformation of the hazard from a dormant state to an active mishap state. T/T is the mishap outcome and the expected consequential damage and loss. There is a hazard actuation process to transition the system from a benign state to a mishap (figure 9).

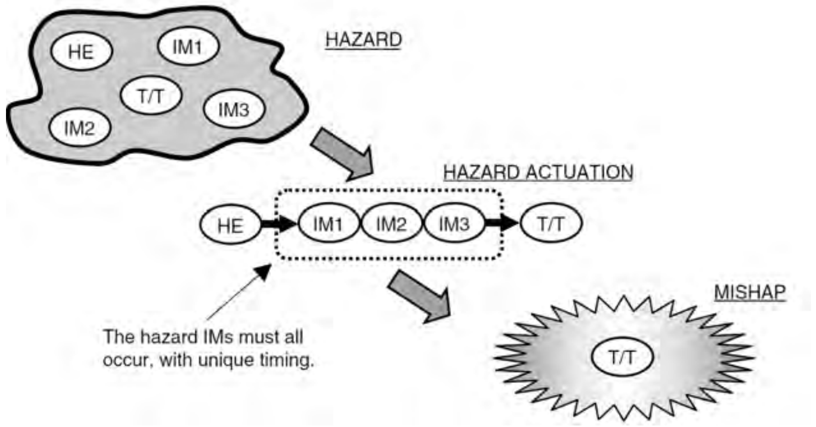

Figure 9. The hazard actuation (adopted from figure 2.5 of [123]).

Inspired by the hazard actuation process, we propose a structure (figure 10) to depict the actual process of how a fault develops in a component and affects the component (i.e., the component fault process). The fundamental belief of this structure is that a component fault is present when certain conditions are satisfied, and the fault takes effects on the component function when certain conditions are satisfied. This observation is consistent with the error propagation process proposed by reference [136] (note the "error" in reference [136] means fault in our context).

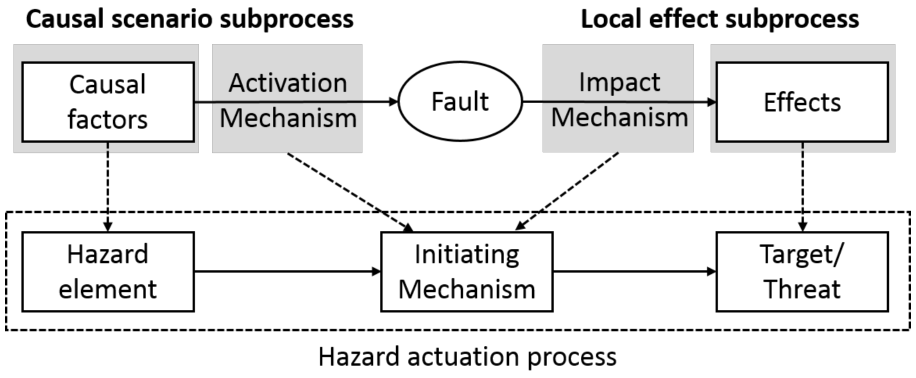

Figure 10. The structure to depict the component fault process appears at the top. It can be mapped to the hazard actuation process at the bottom. The solid arrow represents the causal sequence and the dotted arrow represents the mapping relationship between the proposed structure and the hazard actuation process.

Specifically, the HE in the hazard actuation process is defined as a casual factor in our structure. Each fault is correlated with a set of casual factors, all of which have to be present for the fault to happen. Furthermore, the IM is decomposed into the activation mechanism and the impact mechanism. This is driven by the fact that "a fault does not necessarily lead to a failure" [137], implying two processes in play: one leading to the fault (i.e., the activation mechanism) and the other leading to the effect of the fault (i.e., the impact mechanism). Finally, the T/T is defined as the effects on the respective components. As a result, the causal factors and the activation mechanism comprise the original causal scenario subprocess in the formulate process; the impact mechanism and the effect comprise the original local effect subprocess.

In summary, figure 10 is a structure that we believe is generally true for any component fault process. Again, unlike other scientific communities, there is no experiment to prove its validity definitively. However, it is developed based on well-received works in the System Safety Engineering community. In fact, in 2007, Ortmeier proposed that failure modeling "split the (failure) process into two parts: one part is modelling how and when the failure occurs and the other is to model the direct, local effects of the failure" [52]. In the next section, we will enrich and specify this structure based on the notable patterns of MBSA literature. Although many works do not have the explicit structure as in figure 10, the fact that most MBSA works can fit into this structure indirectly provides evidence of validity for this structure.

5.3 The Notable Patterns to Specify the Phenomenon-centric Framework

5.3.1 Causal Factors

While the specific contents of casual factors vary from application to application, two questions are usually answered in the MBSA literature about the causal factors: (1) what causes the fault and (2) what is the likelihood of those causes occurring? Therefore, two notable patterns are identified for the causal factors: the source and the occurrence (figure 11).

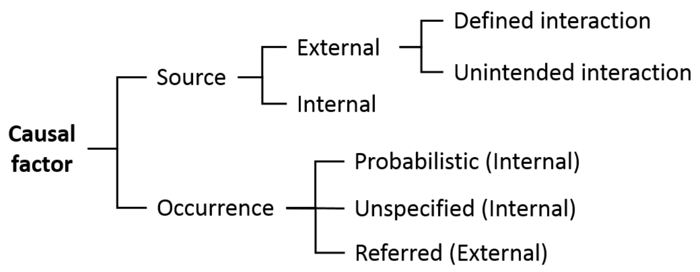

Figure 11. Notable patterns for the casual factors associated with a component fault.

The source of the causal factor is determined with respect to the boundary of the component. As pointed out by reference [53], the component fault can be caused by internal causes or external causes. Most works stop at this level with the exception of reference [54] which goes on classifying the external causes into "connected" and "unconnected." Similar to reference [54], we classify the external causes into "defined interaction" and "unintended interaction." The former means the off-nominal input causes the fault of the component. For example, a voltage (i.e., the input) too high can break a capacitor in the circuit. A processor depends upon the functioning of a fan, and if the fan fails, the processor overheats and fails as well. Similar concepts can be found in references [55] and [138]. Furthermore, new unintended interactions can also be created by the fault of other components, such as fire and fluid leak. This type of interaction is the secondary effect of a fault stemming from an external component that is not supposed to interact with the current component whatsoever. Many MBSA works consider both external and internal fault sources, such as references [41], [56], and [57], but rarely make the explicit distinction between the faults caused by the defined interaction and the unintended interaction. We argue that an explicit distinction should be made between the two different sources of the external causes because (1) the methodology developer must make sure their modeling language can model both phenomena as they are inherently different in nature, and (2) identifying the unintended interactions is challenging, and without explicitly emphasizing it in the formulate process, they are more likely to be missed by the safety engineer.

For the same reason, another important distinction has to be made between the external cause and the propagated input fault as in AADL-EMV2. The former leads to component malfunctions, but the latter does not necessarily so. For example, a current flows into a resistor but is too high (i.e., the fault) for the operation (say to heat a camera in the space shuttle [139]) yet not high enough to change the property of the resistor. In this case, the resistor is not faulty although its input is faulty. The resistor only correctly passes the fault to the camera. Therefore, the faulty current is not an external cause of the resistor because nothing fails in the resistor although its input is faulty. If the current is so high that it changes the performance property of the resistor, then the current becomes an external cause because it leads to improper functioning of the resistor.

The second notable pattern is the occurrence of the causal factor. This aspect determines what types of safety analysis can be conducted later, i.e., a qualitative one or a quantitative one [58]. For the external causal factors, the occurrence depends on the source component(s), and therefore is referred. For the internal causal factors, if they are characterized with probability distributions, then both qualitative analysis and quantitative analysis can be conducted; however, if the probabilistic distribution is unspecified, then only qualitative analysis can be conducted. The implication is quite straightforward: if quantitative analysis is required, then the occurrence of the internal causal factor has to be characterized with probability distribution; otherwise "unspecified" occurrence will suffice.

5.3.2 Activation Mechanism

The causal factors being present is the necessary but not sufficient condition for the fault to be present. Activation mechanism is the name given to the set of additional conditions (if applicable) that the causal factors must satisfy to activate the fault, given all the causal factors are present. This is reflected by some MBSA works defining additional conditions for the fault to occur. Although we do not believe there is an *a priori* definition of those conditions, we are able to find the following list of attributes for the activation mechanism in the literature:

- **Sequence**: Sometimes the causal factors have to happen in a certain sequence so that the fault can be activated. This is one of the main advancements of Dynamic Fault Tree over the traditional FTA.
- **Delay**: Sometimes it takes time for the fault to happen even after the causal factors are all present. For example, a pump overheats after no water flows in for a certain period of time. This time period can be a deterministic one or a probabilistic one [7]. A special case is the zero-delay, where the fault is activated right after the causal factors are present. The presence of the causal factors in the zero-delay case is usually modelled as a trigger event of the fault [46].
- **Duration**: Sometimes a duration is defined to model the phenomenon that a fault can be deactivated. A fault can disappear a certain period of time after the activation because of its transient nature [59] or after being repaired [60]. The characterization of the time can be deterministic, probabilistic [61], or non-deterministic as suggested by reference [62].
-

5.3.3 Impact Mechanism

The fault being present is the necessary but not sufficient condition for the fault to show effects on the component. Impact mechanism is the name given to the set of additional conditions (if applicable) that needs to be satisfied for a given fault to cause the defined effects. This is reflected by some MBSA works defining additional conditions for the effects to take place. Like the activation mechanism, we do not believe there is an *a priori* definition of those conditions, but we are able to find the following list of patterns for the impact mechanism in the literature:

- **Guard**: Sometimes the fault can only lead to the defined effects when the system is in a certain state. In fact, as long as the fault is modeled as an event to trigger a transition, the source state is the guard. In this case, the transition is the effect of the fault; if the system is not in the source state, the transition will not be triggered even if the fault is present. For example, loss of hydraulic supply will only affect the ground deceleration function when the aircraft is in the state of landing. This guard condition is widely modeled in the literature such as in state/event fault tree [46] and AltaRica Data-flow [63].
- **Delay**: As pointed out by reference [33], "the effect of a failure may not immediately cause an output failure mode and may remain dormant." The time between the fault being present and the appearance of the fault effect is usually defined as a "Fault Tolerant Time Interval" in the literature [64]. For example, this delay is considered a "safety-relevant property" in SafeDeML [65].
- **Determinism**: Sometimes it can be uncertain to determine the exact effects of a fault due to either epistemic uncertainty or aleatoric uncertainty. This uncertainty is traditionally addressed by modeling the worst-case scenario instead of the uncertain process. Not many MBSA works have non-deterministic impact process. Reference [66] coins it as "probabilistic transition conditions." Reference [67] provides a feature called "branching

transition" where multiple target states can be transitioned to following certain probability distribution from one source state. This feature was originally designed for branching transient and persistent failure, but it also seems to have the potential to capture the uncertainty of the impact process.

5.3.4 Effects on the Component

The effects of a fault on the respective component is modelled widely differently in the literature, but we are able to find the patterns in figure 12.

Semantics

Abstraction	Off-nominal	Hybrid	Nominal
Function			
Architecture			
Component			

Figure 12. Notable patterns to model the fault effects on a component. A detailed classification of a sample pool of MBSA works is given in the Appendix.

As shown in figure 12, two dimensions are identified from the literature: the abstraction and the semantics. A detailed classification of a sample set of MBSA works is given in the Appendix. The abstraction dimension determines how many details can be included in the effect model. For the component level, the fault affects the internal behaviors of a component. This is also called white box error model in references [45] and [68]. For the architecture level, the fault effects are represented at the output port of a component and propagated to the input port of other components. This is also called black box error model in references [45] and [68]. For the function level, the component is abstracted as a Boolean logic node; the fault sets the node to be false, affecting all the functional flows that pass through the node. In fact, AADL-EMV2 supports modeling at exactly all the three levels of abstraction with a slightly different naming system [69][70]. Obviously, modeling at the function level is more abstract than modeling at the architecture level which is more abstract than modeling at the component level.

The semantics dimension is created based on different interpretations of the local effect. First, the local effect is interpreted as a deviation to the intended performance of the defined behavior. In this way, the semantics for the nominal behavior is reused. Second, the local effect is interpreted as a new behavior (such as omission, commission, early, late, or value deviations) beyond the original nominal behavior. In this way, only off-nominal states are modeled in the safety model, hence "off-nominal." Third, the local effect is interpreted as a new off-nominal state interacting with the nominal states of the system. In this way, both the nominal states and the off-nominal states are present in the safety model, hence "hybrid."

In fact, this semantics dimension is consistent with the classification result of reference [48], which is based on the semantics of the component interface. When the local effect is represented by the nominal semantics, the interaction between the components is the "nominal flow," which corresponds to the FEM class of reference [48]. When the local effect is only represented by off-nominal semantics, the interaction between the components is the "fault logic," which corresponds to the FLM class. Finally, when the local effect is new off-nominal states interacting with nominal states, both the nominal state and off-nominal state have to be communicated between the components, which corresponds to the "hybrid" class in reference [48].

5.4 The Phenomenon-centric Framework for the Component Fault Process

The full framework of the component fault free is derived (figure 13). The structure is proposed in an a priori way based on several well-received works, and the specifics of the framework are achieved by the notable patterns from the MBSA literature. This phenomenon-centric framework not only provides a way to organize the notable patterns of the component fault process in MBSA, but more importantly it is a solution to the problem posed in section 5.1 of lacking the hard facts. We argue that this framework quasi-functions as the hard facts as in other scientific model communities.

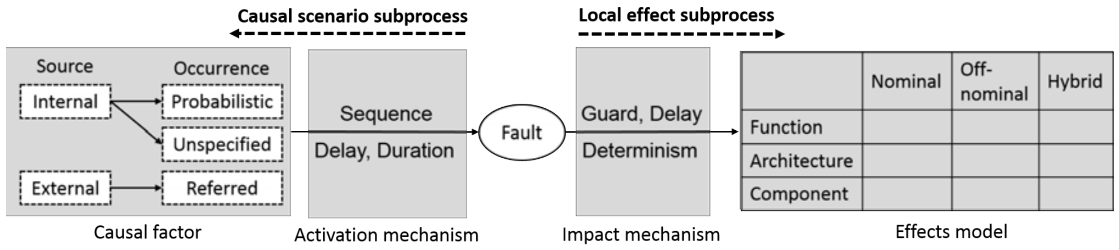

Figure 13. The resulting phenomenon-centric framework to describe the hard facts of the component fault process by combing the structure in section 5.2 and notable patterns in section 5.3.

For the safety engineer, the framework is language neutral. It provides structure to systematically formulate the real component fault process. The specific patterns presented in the framework can be used as guidewords to capture different types of phenomena in the component fault process. Furthermore, the safety engineer can also use the framework as a neutral standard to compare different alternative modeling languages, and hence become explicitly aware about not only what fault phenomenon can be modeled by the languages but more importantly what is abstracted away by the language.

For the language developer, this framework helps them make explicit decisions about what fault phenomenon will be supported by the modeling construct and what will not. Furthermore, this framework can also be used as a meta-model of the modeling language (specifically the modeling construct). By mapping the specific language semantics and syntax to this framework, it gives the reader more transparency about how the language is designed to represent the real process and potentially makes the language easier to learn, to use, and to improve.

Finally, although we cannot guarantee the aspects represented in the framework are exhaustive, thanks to the flexibility of the basic structure, more aspects can be added to the basic structure as more aspects are found in the MBSA community. This flexibility allows the authors to keep this framework a living artifact and evolve it as we go.

6 MBSA: The Desired Analysis

Next, the "model of computation" of the MBA in figure 4 corresponds to the safety model. However, because most of the modeling decisions are made in the previous steps and this step is only to express the results by using the right syntax in the modeling environment, we skip the step and proceed to "the desired analysis" of figure 4. In the context of MBSA, the desired analysis is the safety analysis.

In this paper, we are not interested in the details of how the algorithms are designed and how the tools are developed, which are closer to the Software Engineering community than System

Safety Engineering community. Rather, we focus on the mathematical construct and its implication to model transformation (section 6.1) and the safety analysis (section 6.2) in the MBSA literature.

6.1 Mathematical Construct

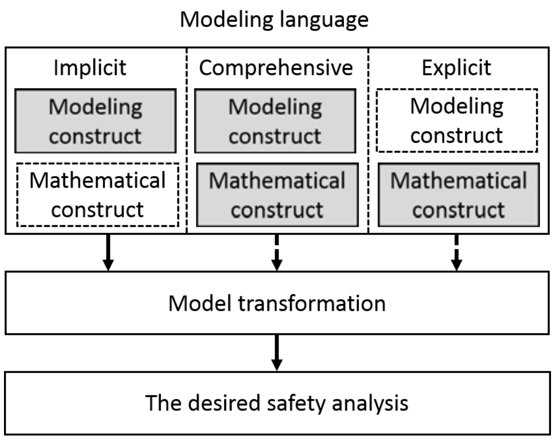

Figure 14. The notable patterns of the mathematical construct.

To reiterate, the mathematical construct is the mathematical formalism (implicit or explicit) of a modeling language for the automatic analysis by the computer programs. For example, the mathematical construct of NuSMV is Finite State Machine (FSM) and AltaRica 3.0 is a General Transition System [71]. The goal is to use the mathematical construct for the desired safety analysis, either by designing new analysis algorithms or reusing existing tools. If such a goal cannot be obtained by the current mathematical construct, transformation will be performed until the desired analysis can be conducted on the resulting mathematical construct.

Based on how the mathematical construct is associated with the modeling construct, we found the following three notable patterns of the MBSA languages from the literature (figure 14):

- **Implicit:** This class of languages is developed mostly for representing the system (off-nominal) behavior in a specific way, and no dedicated mathematical construct is designed specifically for the language (the dotted box of mathematical construct). The modeling language *has to* be transformed (solid line to the model transformation) into a certain, usually existing and well-supported formalism so that tools can be found for the desired safety analysis, such as UML in reference [72], Sysml in references [73] and [74], HipHops in reference [75] and AADL in reference [76].
- **Comprehensive**: This class of languages are developed with both a modeling construct to represent the system (off-nominal) behavior and an equivalent mathematical construct for the analysis (hence the solid boxes), such as the SMV [39] and Statemate [77] [78] with finite state machine, SLIM with Event Data Automation (EDA) [57], AltaRica with General Transition System [7], and Arcade with Input/output interactive Markov chains (I/O-IMC) [55]. However, because it is not guaranteed that the embedded mathematical construct fits for the desired safety analysis, it is possible further transformation is still needed (the dotted arrow to model transformation), such as the I/O-IMC to CTMC in [55] and EDA to MRMC in [57].
- **Explicit:** This class of languages is, in essence, the mathematical construct, and no modeling construct is built for the languages (dotted box for the modeling construct). The

safety engineer has to build the off-nominal model directly using the mathematical construct such as the Interface Automaton in reference [79] and Hybrid Automaton [80]. Usually, the formalism is well-supported by existing tools for the desired safety analysis. Therefore, the model transformation is not necessarily required (a dotted arrow downward).

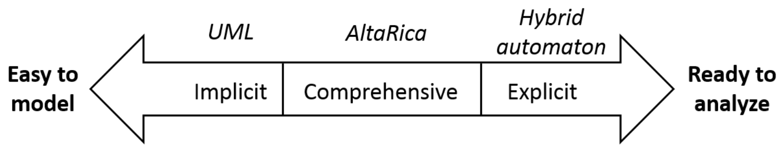

Figure 15. Comparing the three types of MBSA languages.

In fact, the notable patterns identified above are consistent with references [107] and [81]. Five criteria are developed [81] to qualitatively compare the effectiveness of a modeling language that are basically evaluations on the effectiveness of the modeling construct and mathematical construct described in this paper. Centered on the modeling construct and the mathematical construct, we further summarize the five criteria into two dimensions: easy to model and ready to analyze. The three types of languages have opposite performance along the two dimensions.

As shown in figure 15, languages that are intuitive and flexible for modeling (such as UML on the left of the spectrum) tend to be limited for analyzing, and languages that are ready to be analyzed tend to make modelling the realistic behavior difficult especially as the system becomes intrinsically more complex (such as hybrid automaton on the right of the spectrum). The comprehensive language (such as AltaRica 3.0) meets both metrics in the middle. Unlike the implicit language that is heavy on the modeling construct and the explicit language that is heavy on the mathematical formalism, the comprehensive language is more balanced and hence is easier to use for the safety engineer, but poses a greater challenge to the language developer to have a language with both the intuition and flexibility for modeling and rigor and readiness for analysis.

At a deeper level, the classification is driven by the amount of information assumed in both the system to be modelled and the analysis to be conducted. The implicit class mentioned above does not assume what kind of analysis needs to be conducted later but does assume to a varying extent what kind of engineering system needs to be modeled. The explicit class does not assume what kind of engineering system needs to be modeled but does assume to a varying extent what kind of analysis needs to be conducted. The comprehensive class assumes information to a varying extent about both the engineering system to be modeled and analysis to be conducted as shown in figure 15 above.

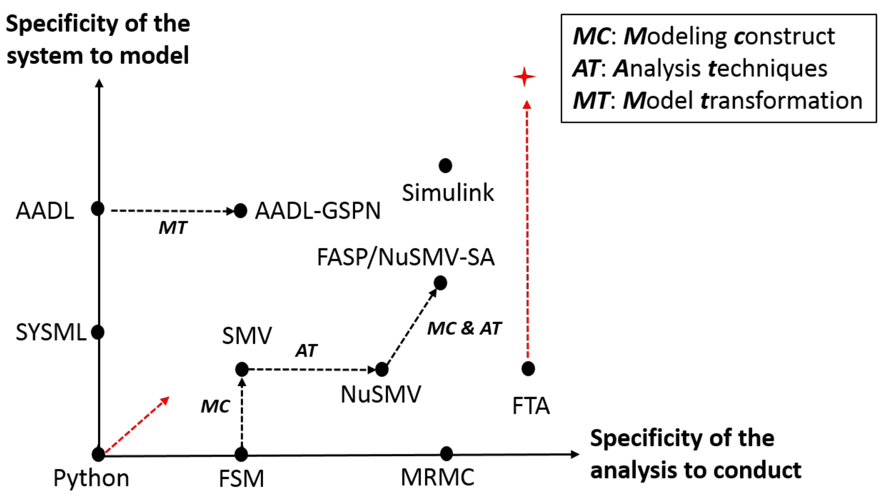

Figure 16. Comparison of modeling language with regard to how much information is assumed about the system to be modelled (the *y*-axis) and the desired analysis (the *x*-axis). The origin is the general language, like Python (not to imply it is MBSA language), that does not assume any information at either dimension. The red star at the top-right corner represents a specific MBSA task that has specific information about the target engineering system and the desired safety analysis. Given a language (at any point of the plane) for a specific MBSA task, efforts are made either by the methodology developer (the black dotted line) or by the safety engineer (the red dotted line) to drive the point moving towards the red star at top-right corner.

For this reason, a more general comparison between the modeling languages is conducted in figure 16. The *x*-axis corresponds to the explicit class of language, which assumes no information about the engineering system to be modeled. Along the *x*-axis, languages are positioned with regard to the relative level of specificity in what kind of the analysis is to be conducted. For example, Python at the origin is such a general language that can be used for a wide range of analysis; FSM is a specific mathematical formalism with a more limited range of potentially available analysis; and Markov Reward Model Checker (MRMC) [141], based on the Markov process (a mathematical formalism), has a specific range of feasible analysis. Along the *x*-axis, no information is assumed about the engineering system to be modelled, but the information about the analysis to be conducted becomes more specific. Note that, we take a broad definition toward "modeling language" in this paper. Strictly FSM is a formal mathematical construct and MRMC is a tool, but a pure mathematical construct can also be used to model engineering systems directly, and the tool has an input modeling language with specific analysis capability. Hence both of them are considered languages in this paper, and this definition also applies to Simulink and NuSMV below.

The *y*-axis of figure 16 corresponds to the implicit class of language, which assumes no information about the analysis to be conducted. Along the *y*-axis, languages are positioned with regard to the relative level of specificity of what kind of engineering system is to be modelled. For example, Python is such a general language that can be used to model a wide range of systems, while SYSML is more specialized in engineering system modeling, and AADL is further restricted to avionics systems.

The comprehensive class of language is positioned within the plane. Depending on the level of specificity at each dimension, different languages can be positioned accordingly. For example, Simulink has a specific and unambiguous (although wide) boundary about what systems can be modelled and what analysis can be conducted, and is thus placed towards the top-right of the plane.

Moreover, figure 16 reveals how a language moving from one point to another fits into the MBSA (or the more general MBA process). For example, the horizontal movement from AADL to GSPN is achieved through the "model transformation" mentioned earlier in this section. It does not make the modeling construct of AADL more specific but gives a mathematical formalism that adds specificity to analysis of conduct. Another example is the FSM on the *x*-axis at the bottom. FSM first moves vertically to SMV [140]. Compared with FSM, SMV has the built-in semantics to model complex hierarchical systems, which is the added specificity on the engineering system to be modelled. Furthermore, the NuSMV is a model checker built based on the SMV, through which specific model checking analysis can be conducted automatically. Compared with SMV, NuSMV adds specificity to what analysis can be conducted through the dedicated analysis techniques, thus a horizontal movement. Finally, based on NuSMV, FSAP/NuSMV-SA is developed for MBSA, which is equipped with additional modeling constructs to describe different types of components' faults and dedicated analysis techniques for specific safety analysis. It adds specificity to both dimensions, thus a movement towards the top-right direction.

For a safety engineer, the MBSA task is usually conducted in a project-by-project manner, meaning *specific* information about the system to model and the analysis to conduct is available for both dimensions. Hence, a MBSA task can be represented by an imaginary red star at the top-right corner of the plane, and given modeling languages always move towards the top-right direction to accomplish a *specific* MBSA task, the movement can be achieved by the language developer as explained in the last paragraph (the black dotted line in figure 16) or the safety engineer (the red dotted line in figure 16). For example, to use Python for a MBSA analysis, the safety engineer has to build the safety model and design the analysis algorithms (hence the top-right directed arrow) from scratch, which can be extremely challenging. The FTA is very general in the modeling dimension but very specific in the analysis. For this reason, the safety engineer has to expend a significant amount of effort in modeling (hence an upward arrow) to perform an FTA, which is one of the main reasons that MBSA is proposed in the first place. In fact, the more general a language is, the more effort the safety engineer needs to make for a specific MBSA task and hence more room for errors.

Therefore, from the perspective of the methodology developer, there is an incentive to make the language as specific as possible, i.e., placing it as close to the top-right corner as possible so that the safety engineer needs to expend the least amount of effort in terms of modeling and analysis. However, the specificity comes with a price, which is the flexibility of the language. A Simulink package for control system design is too specific for a financial system, but Python is general enough to model all kinds of systems. Clearly, the challenge is how to strike a balance between the specificity (of both dimensions) and flexibility of the modeling language so that the safety engineer has to expend the least effort for the specific MBSA task, and the language is still general enough for a wide range of systems and analysis. This is an open challenge for future work.

6.2 Safety Analysis

Safety analysis in a more general sense usually implies a safety modeling process and an analysis process based on the safety model. In this section we use the term in latter sense. Furthermore, in MBSA, the safety analysis is supposed to be automatically conducted by the tools after the safety model is constructed. This is a defining feature, because if the safety analysis cannot be conducted automatically, it is not even MBA let alone a MBSA. Therefore, **automatic safety model analysis is a defining feature of MBSA.**

In the rest of this section, we focus on the notable patterns of safety analysis. We first discuss one of the most prominent analyses, automatic FTA, then a complete list is given for all the different types of safety analysis we found in the MBSA literature.

6.2.1 Automatic FTA

One of the most popular MBSA safety analyses is automatic FTA [82]–[84]. It is true that with proper automation support, the automatic generation of a fault tree can lead to the Architecture Consistency between the design model and safety model, a defining feature of MBSA. But it does not necessarily guarantee the quality of the fault tree. In fact, the involvement of automation can give the safety engineer a false sense of complacency, an illusion that because it is done automatically, it must be correct [100]. Failure modes being organized in the form of a fault tree do not necessarily mean a fault tree analysis is appropriately conducted.

At its core, FTA is a deductive analysis, identifying the possible casual scenarios for a high-level event. On the surface, the automation eliminates the deductive process. However, it does not eliminate the necessity of the deductive analysis. It only pushes it to the phase where the component fault is identified and defined. To make the situation worse, the original methodological support provided by the traditional FTA now is automated away by the idea of automatic generation of a fault tree. In fact, most MBSA works take the lower level causal factors as given. No rationale is given about how these failure modes are generated in the first place, and thus there is no way to review whether all the possible causal factors are identified to a reasonable extent. Reference [30] tries to tackle this problem with a bottom-up formal analysis, but it only applies to low-level devices. This significantly compromises its effectiveness in the development of system architectures, which is actually the whole point of FTA in reference [113].

In fact, most fault trees in MBSA are automatically reconstructed from a bottom-up inductive analysis, which is actually a logic equivalence of failure modes and effects analysis (FMEA). Therefore, MBSA can be an appropriate approach to automate FMEA, but not necessarily FTA. It can even be counterproductive as the false sense of complacency in automated FTA can lead to a less rigorous review process.

6.2.2 Notable Patterns of Safety Analysis

The following types of safety analysis are found in the MBSA literature. Note that only the commonly conducted safety analysis is recorded here. The less commonly supported analysis is not included, such as the safety optimization in reference [75] and the error propagation analysis in reference [85].

(1) *Fault tree analysis*: This includes the automatic generation of a fault tree, the derivation of the minimal cut sets, and the calculation of the failure rate. Note that not all works conducts all three tasks. For example, reference [43] only mentions the generation of the fault tree; reference [86] only derives the minimal cut set; and reference [87] only addresses the failure rate calculation for the hazard. However, because all three activities are part of a conventional FTA, they are all classified as fault tree analysis.

(2) *Failure modes and effects analysis* (FMEA): This is the automatic generation of FMEA [88][89]. Because FMEA is a bottom-up process, which is consistent with the inductive nature of the MBSA practice, it can be truly automated by MBSA.

(3) *Reliability Block Diagram* (RBD): This is similar to FMEA and can be automatically accomplished by such as references [41] and [90].

(4) *Probabilistic indicators*: This is a broad class of analysis for dependability, such as reliability and availability. We direct readers to reference [136] for a detailed explanation.

Example works that cover analysis of probabilistic indicators include references [57] and [91].

(5) *Property verification for nominal behavior*: This is the formal verification of nominal behavior (e.g., refs. [56] and [92]) against function properties. It checks the "goodness" of a design, which is a precondition for any safety analysis.

(6) *Property verification for off-nominal behavior*: This is the formal verification of off-nominal behavior (e.g., refs. [101] and [80]) against function properties. It rigorously checks whether certain safety constraints can be broken under certain off-nominal conditions.

(7) *Critical sequence*: A critical sequence is a sequence of events leading from the initial state to a critical state. In the case of dynamic models, the order of occurrences of events is important, and thus the approximation consisting in extracting minimal cut sets is not suitable: minimal or most probable sequences or sequences of a given length (also called order) can be extracted by simulation of the model [1]. Example works include references [1], [72], [93], and [94] .

(8) *Trace simulation*: This is to display the traces of the individual failure scenarios for the safety engineering to debug the model and understand the propagation of the fault effect. It can be a step-wise interactive simulation [63], or the trace of a given number of steps is output at the end of the simulation [39].

(9) *Common cause analysis*: This is required in reference [113] and aims at investigating possible dependencies between the faults and evaluates the consequences in terms of system safety/reliability [95]. Example works include references [96] and [78].

7 Conclusion

7.1 Defining Features and Notable Patterns

Figure 17 is a summary of the defining features and notable patterns, which are the main goals of this paper.

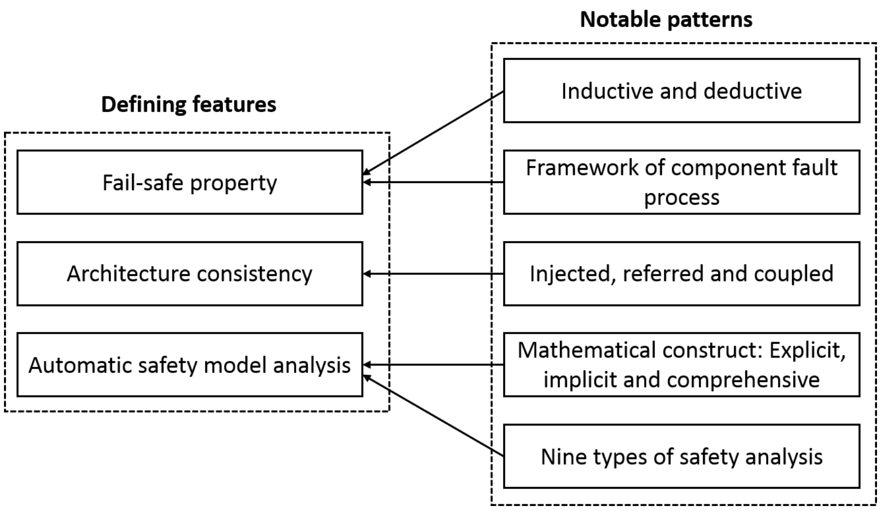
Notable patterns
Defining features
Inductive and deductive
Fail-safe property
Framework of component fault process
Architecture consistency
Injected, referred and coupled
Automatic safety model analysis
Mathematical construct: Explicit, implicit and comprehensive
Nine types of safety analysis

Figure 17. A summary of the defining features and notable patterns. Arrow means "implements" here.

The defining features (left of figure 17) of MBSA include the following three criteria:

7.1.1.1 Whether the method can be used to verify the fail-safe property of a system.
7.1.1.2 Whether the design model and safety model are consistent in terms of the architectures; that is the consistency between the interaction paths defined in the design model and the propagation paths captured in the safety model.
7.1.1.3 Whether automatic analysis on the safety model is supported.

If a work checks all the criteria above, then it is MBSA; if any of them is not satisfied, then that work cannot qualify as MBSA.

Moreover, the notable patterns (right side of figure 17) are different schools of thoughts about different aspects of a MBSA analysis that we found in the literature:

(1) The safety analysis conducted in MBSA is overwhelmingly inductive (bottom-up), but this does not mean that a deductive (top-down) analysis cannot nor should not be embedded in MBSA.
(2) The framework of the component fault process provides a template (i.e., a collection of different patterns) to determine how a fault develops in a component.
(3) Architecture Consistency between the design model and safety model can be achieved in three ways: injected, referred, and coupled.
(4) There are three types of MBSA languages depending on how mathematical construct is addressed: explicit, implicit, and comprehensive.
(5) Nine types of safety analysis have been found in the MBSA literature.

In fact, the notable patterns (the arrows in figure 17) implement the defining features. First, to argue whether a system is fail-safe, given the safety requirements (which come from hazard identification), a model of off-nominal behavior has to be made, and the model has to be verified against the given safety requirements. This is implemented by the first two notable patterns. The deductive analysis is to identify the contributing scenarios including component faults that can lead to the hazard of interest; the framework helps to make abstraction decisions about how to model the component fault; the inductive analysis verifies whether the given safety requirements are satisfied. Second, the three different ways of achieving Architecture Consistency has been explained in great detail in section 4.3, with pros and cons in section 4.4. Finally, different ways to embed a mathematical construct in the modeling languages determines what kind of automatic safety analysis is available and whether model transformation is required before the automatic safety analysis can be conducted.

7.2 Suggestions Moving Forward.

We explain the findings listed in section 1.

Deductive analysis. As argued in section 3.2.2, the deductive analysis is indispensable in assuring the safety of a system. Without a deductive (top-down) analysis, the quality of the current inductive MBSA analyses is dubious at best. Substantially more evidence must be provided to demonstrate that the identified casual factors coming from inductive methods are complete to an acceptable extent. Moreover, automatic FTA (section 6.2.1) automates away the deductive process of the traditional FTA without proposing any effective replacement. MBSA is driven by fields like Systems Engineering, System Safety Engineering, Software Engineering, and Formal Methods. The general lack of deductive analysis in the MBSA literature is perhaps caused by the faster advancement in the last two communities (especially Formal Methods which primarily focus on inductive analysis) than the first two communities (especially the System Safety Engineering community to which the deductive safety analysis is almost exclusively belong). Moving forward, it is crucial especially for the System Safety Engineering community to develop new or modify existing deductive analysis techniques to integrate with the current inductive MBSA methods and tools, so that together system safety can be assured in a cheaper, faster, and more trustworthy manner.

Explicit abstraction. Another bottleneck of MBSA is the quality of safety model, which is also a core topic of the System Safety Engineering community. As argued in section 5, because there is no set of hard facts for safety engineering, the abstraction of safety models is only guided by a loose "fit for purpose" principle, which eventually leaves the construction and the review of the safety model to the discretion of individual safety engineers. To be explicit about what has been abstracted away, we need the hard facts of the component fault process so that the abstraction can be made explicitly. The framework proposed in section 5 aims to serve as the hard facts. While the structure is based on a priori concepts from well-received works, the specifics are from the phenomenon described in the MBSA literature. We plan to add more phenomenon to the framework as more works are reviewed from not only the MBSA community but the general System Safety Engineering community.

Automation. It is a repeated theme in this paper that although automation is conceptually closely associated with MBSA, it is too broad to be a defining feature of MBSA. It is important moving forward to not over-claim the contribution due to automation because it can cause a false sense of complacency, which can be dangerous for safety assurance. Particularly, automation can play three roles in MBSA: to achieve Architecture Consistency in the specific injected way (section 4.3), to aggregate the well-defined component faults (section 4.4.3) and to transform and analyze the safety model (section 6). Finally, because the inductive nature of the current MBSA practice, it is perfect to automate FMEA but dangerous to automate FTA if no deductive analysis technique is proposed as a complement. After all, the authors fail to see the difference between automatic FTA and automatic FMEA in current MBSA practice.

Specificity in the modeling language. This is mainly concerned with the discussion in section 6.1. There is a tradeoff to make between the specificity (in terms of the system to model and analysis to conduct) and flexibility of the modeling language so that the safety engineer only has to make minimal effort for the specific MBSA task and the language is still general enough for a wide range of systems and analysis. The more specific a language is, on one hand, the less room for errors there is for the safety engineer, but the less flexible the language is. Currently, most MBSA languages are general enough for flexible applications in different domains, and the desired safety analysis is pretty specific (see section 6.2.2). It is the specificity along the modeling dimension (*y*-axis of figure 16) that needs more investigation. For example, intuitively, AADL-EMV2 is more specific than FTA as it can model more behaviors (also known as expressiveness [97] in the literature). But is there any language more specific than AADL-EMV2? How could one know whether the added specificity is valid? Moreover, what does specificity even mean in safety engineering? Adding specificity to the modeling dimension will definitely sacrifice the flexibility, but to what extent; furthermore, is there an optimal solution to the tradeoff between flexibility and specificity? These are tough questions that need to be answered by the System Safety Engineering community moving forward. The authors believe the difficulty in answering the questions is partially caused by the lack of hard facts (see section 5.1) in the System Safety Engineering community as the baseline for comparison. The framework proposed in section 5.4 and the AADL error taxonomy in reference [3] are both efforts to build the hard facts for the System Safety Engineering community. In addition, references [98] and [99] propose System Structure Modeling

Language (S2ML), a generic modeling structure to connect the modeling construct and mathematical construct of a wide range of MBSA languages, which the authors believe is another promising direction to address the tradeoff between specificity and flexibility.

In summary, current MBSA practice tends to focus on the verification phase of the traditional safety assessment process [113], where the safety model is taken as a given input. Although the notion of Architecture Consistency between the safety model and design model *improves* the quality of safety model, it is not convincing to the authors that Architecture Consistency is adequate to *ensure* the quality of the resulting safety model. Moving forward, we advocate, especially for the System Safety Engineering community, to put more emphasis on the activities conducted before the verification phase, which emphasizes how to generate safety models that are of high quality and also compatible with the current MBSA practices to achieve industry level maturity for MBSA in the future.

8 References

[1] Prosvirnova, Tatiana. AltaRica 3.0: a model-based approach for safety analyses. Diss. 2014.

[2] Larson, Brian, et al. "Illustrating the AADL error modeling annex (v. 2) using a simple safety-critical medical device." *ACM SIGAda Ada Letters* 33.3 (2013): 65-84.

[3] Procter, Sam, and Peter Feiler. "The AADL error library: An operationalized taxonomy of system errors." *ACM SIGAda Ada Letters* 39.1 (2020): 63-70.

[4] Feiler, Peter, and Julien Delange. "Automated fault tree analysis from aadl models." *ACM SIGAda Ada Letters* 36.2 (2017): 39-46.

[5] Machin, Mathilde, et al. "Modeling Functional Allocation in AltaRica to Support MBSE/MBSA Consistency." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2019.

[6] Bieber, Pierre, et al. "Safety assessment with AltaRica." *Building the Information Society*. Springer, Boston, MA, 2004. 505-510.

[7] Prosvirnova, Tatiana, et al. "The altarica 3.0 project for model-based safety assessment." *IFAC Proceedings Volumes* 46.22 (2013): 127-132.

[8] Mortada, Hala, Tatiana Prosvirnova, and Antoine Rauzy. "Safety assessment of an electrical system with AltaRica 3.0." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2014.

[9] Tlig, Mohamed, et al. "Autonomous Driving System: Model Based Safety Analysis." 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 2018.

[10] Kabir, Sohag, et al. "A Conceptual Framework to Incorporate Complex Basic Events in HiP-HOPS." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2019.

[11] Chen, DeJiu, et al. "Systems modeling with EAST-ADL for fault tree analysis through HiP-HOPS." *IFAC Proceedings Volumes* 46.22 (2013): 91-96.

[12] Bozzano, Marco, et al. "Safety assessment of AltaRica models via symbolic model checking." *Science of Computer Programming* 98 (2015): 464-483.

[13] Braun, Peter, et al. "Model-based safety-cases for software-intensive systems." *Electronic Notes in Theoretical Computer Science* 238.4 (2009): 71-77.

[14] Joshi, Anjali, et al. "Model-based safety analysis." (2006).

[15] Batteux, Michel, Tatiana Prosvirnova, and Antoine Rauzy. "Model synchronization: a formal framework for the management of heterogeneous models." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2019.

[16] Bäckström, Ola, et al. "Effective static and dynamic fault tree analysis." *International Conference on Computer Safety, Reliability, and Security*. Springer, Cham, 2016.

[17] Junges, Sebastian, et al. "Uncovering dynamic fault trees." 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2016.

[18] Beckers, Kristian, et al. "A structured and model-based hazard analysis and risk assessment method for automotive systems." *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2013.

[19] Cancila, Daniela, et al. "Sophia: a modeling language for model-based safety engineering." 2nd international workshop on model based architecting and construction of embedded systems, CEUR. Denver, Colorado. 2009.

[20] Guiochet, Jérémie. "Hazard analysis of human–robot interactions with HAZOP–UML." *Safety science* 84 (2016): 225-237.

[21] Johannessen, Per, et al. "Hazard analysis in object oriented design of dependable systems." *2001 International Conference on Dependable Systems and Networks*. IEEE, 2001.

[22] Kaleeswaran, Arut Prakash, et al. "A domain specific language to support HAZOP studies of SysML models." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2019.

[23] Cuenot, Philippe, et al. "Towards improving dependability of automotive systems by using the EAST-ADL architecture description language." *Architecting dependable systems IV*. Springer, Berlin, Heidelberg, 2007. 39-65.

[24] Cuenot, Philippe, Loic Quéran, and Andreas Baumgart. "Safe Automotive soFtware architEcture (SAFE)." (2013).

[25] Chen, D., et al. "Integrated safety and architecture modeling for automotive embedded systems." *e & i Elektrotechnik und Informationstechnik* 128.6 (2011): 196-202.

[26] Grigoleit, Florian, et al. "The qSafe Project–Developing a Model-based Tool for Current Practice in Functional Safety Analysis." (2016).

[27] Chaari, Moomen, et al. "Transformation of failure propagation models into fault trees for safety evaluation purposes." *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 2016.

[28] Fenelon, Peter, et al. "Towards integrated safety analysis and design." *ACM SIGAPP Applied Computing Review* 2.1 (1994): 21-32.

[29] Lisagor, O., J. A. McDermid, and D. J. Pumfrey. "Towards a practicable process for automated safety analysis." *24th International system safety conference*. Vol. 596. 2006.

[30] Ortmeier, Frank, and Wolfgang Reif. "Failure-sensitive specification: A formal method for finding failure modes." (2006).

[31] Güdemann, Matthias, and Frank Ortmeier. "Quantitative model-based safety analysis: A case study." Sicherheit 2010. Sicherheit, Schutz und Zuverlässigkeit (2010).

[32] Wille, Alexander. *Contributions to Model-Based Safety Assessment*. Diss. Technische Universität München, 2019.

[33] Lisagor, Oleg. *Failure logic modelling: a pragmatic approach*. Diss. University of York, 2010.

[34] Joshi, Anjali, et al. "A proposal for model-based safety analysis." *24th Digital Avionics Systems Conference*. Vol. 2. IEEE, 2005.

[35] Bozzano, Marco, and Adolfo Villafiorita. "Improving system reliability via model checking: The FSAP/NuSMV-SA safety analysis platform." *International Conference on Computer Safety, Reliability, and Security*. Springer, Berlin, Heidelberg, 2003.

[36] Akerlund, O., et al. "ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects." 2006.

[37] Bozzano, Marco, et al. "Improving safety assessment of complex systems: An industrial case study." *International Symposium of Formal Methods Europe*. Springer, Berlin, Heidelberg, 2003.

[38] Bozzano, Marco, et al. "ESACS: an integrated methodology for design and safety analysis of complex systems." *Proc. ESREL*. Vol. 2003. Balkema Publisher, 2003.

[39] Bozzano, Marco, and Adolfo Villafiorita. "The FSAP/NuSMV-SA safety analysis platform." *International Journal on Software Tools for Technology Transfer* 9.1 (2007): 5.

[40] Grunske, Lars, and Jun Han. "A comparative study into architecture-based safety evaluation methodologies using AADL's error annex and failure propagation models." *2008 11th IEEE High Assurance Systems Engineering Symposium*. IEEE, 2008.

[41] Delange, Julien, et al. *AADL fault modeling and analysis within an ARP4761 safety assessment*. No. CMU/SEI-2014-TR-020. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2014.

[42] Parker, David, Martin Walker, and Yiannis Papadopoulos. "Model-based functional safety analysis and architecture optimisation." *Embedded Computing Systems: Applications, Optimization, and Advanced Design*. IGI Global, 2013. 79-92.

[43] Kaiser, Bernhard, Peter Liggesmeyer, and Oliver Mäckel. "A new component concept for fault trees." *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33*. 2003.

[44] McDermid, John A., et al. "Experience with the application of HAZOP to computer-based systems." COMPASS'95 Proceedings of the Tenth Annual Conference on Computer Assurance Systems Integrity, Software Safety and Process Security'. IEEE, 1995.

[45] Fenelon, Peter, and John A. McDermid. "New directions in software safety: Causal modelling as an aid to integration." *Workshop on Safety Case Construction, York (March 1994)*. 1992.

[46] Kaiser, Bernhard, Catharina Gramlich, and Marc Förster. "State/event fault trees—A safety analysis model for software-controlled systems." *Reliability Engineering & System Safety* 92.11 (2007): 1521-1537.

[47] Piriou, Pierre-Yves, Jean-Marc Faure, and Jean-Jacques Lesage. "Control-in-the-loop model based safety analysis." 2014.

[48] Lisagor, Oleg, Tim Kelly, and Ru Niu. "Model-based safety assessment: Review of the discipline and its challenges." *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*. IEEE, 2011.

[49] Chaudemar, Jean-Charles, et al. "Altarica and event-b models for operational safety analysis: Unmanned aerial vehicle case study." *Workshop on Integration of Model-based Formal Methods and Tools*. 2009.

[50] Chaudemar, Jean-Charles, Eric Bensana, and Christel Seguin. "Model based safety analysis for an unmanned aerial system." (2010).

[51] Mohrle, Felix, et al. "Automated compositional safety analysis using component fault trees." *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2015.

[52] Ortmeier, Frank, Matthias Güdemann, and Wolfgang Reif. "Formal failure models." *IFAC Proceedings Volumes* 40.6 (2007): 145-150.

[53] Papadopoulos, Yiannis, and John A. McDermid. "Hierarchically performed hazard origin and propagation studies." *International Conference on Computer Safety, Reliability, and Security*. Springer, Berlin, Heidelberg, 1999.

[54] Joshi, Anjali, and Mats PE Heimdahl. "Behavioral fault modeling for model-based safety analysis." *10th IEEE High Assurance Systems Engineering Symposium (HASE'07)*. IEEE, 2007.

[55] Boudali, Hichem, et al. "Architectural dependability evaluation with Arcade." *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008.

[56] Stewart, Danielle, et al. "Architectural modeling and analysis for safety engineering." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2017.

[57] Bozzano, Marco, et al. "Safety, dependability and performance analysis of extended AADL models." *The Computer Journal* 54.5 (2011): 754-775.

[58] Gudemann, Matthias, and Frank Ortmeier. "A framework for qualitative and quantitative formal model-based safety analysis." *2010 IEEE 12th International Symposium on High Assurance Systems Engineering*. IEEE, 2010.

[59] Ortmeier, Frank, Wolfgang Reif, and Gerhard Schellhorn. "Deductive cause-consequence analysis (DCCA)." *IFAC Proceedings Volumes* 38.1 (2005): 62-67.

[60] Chaux, P., et al. "Qualitative analysis of a bdmp by finite automaton." *Advances in Safety, Reliability and Risk Management* (2011): 329-329.

[61] Ortmeier, Frank, Wolfgang Reif, and Gerhard Schellhorn. "Formal safety analysis of a radio-based railroad crossing using deductive cause-consequence analysis (DCCA)." *European Dependable Computing Conference*. Springer, Berlin, Heidelberg, 2005.

[62] Güdemann, Matthias, and Frank Ortmeier. "Probabilistic model-based safety analysis." *arXiv preprint arXiv:1006.5101* (2010).

[63] Seguin, Christel, et al. "Formal assessment techniques for embedded safety critical system." *2nd National Workshop on Control Architectures of Robots (CAR'2007)*. 2007.

[64] Gonschorek, Tim, et al. "Integrating Safety Design Artifacts into System Development Models Using SafeDeML." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2019.

[65] Gonschorek, Tim, et al. "SafeDeML: On integrating the safety design into the system model." *International Conference on Computer Safety, Reliability, and Security*. Springer, Cham, 2019.

[66] Braman, Julia MB, and Richard M. Murray. "Probabilistic safety analysis of sensor-driven hybrid automata." *Hybrid Systems: Computation and Control* (2009).

[67] Delange, Julien, and Peter Feiler. "Architecture fault modeling with the AADL error-model annex." *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2014.

[68] Cuenot, P., et al. "Applying model based techniques for early safety evaluation of an automotive architecture in compliance with the ISO 26262 standard." 2014.

[69] Kushal, K. S., Manju Nanda, and J. Jayanthi. "Architecture level safety analyses for safety-critical systems." *International Journal of Aerospace Engineering* 2017 (2017).

[70] Delange, Julien, and Peter Feiler. "Architecture fault modeling with the AADL error-model annex." *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2014.

[71] Batteux, Michel, Tatiana Prosvirnova, and Antoine Rauzy. "Modeling patterns for the assessment of maintenance policies with AltaRica 3.0." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2019.

[72] Leitner-Fischer, Florian, and Stefan Leue. "Quantitative analysis of UML models." *Proceedings of Modellbasierte Entwicklung eingebetteter Systeme (MBEES 2011). Dagstuhl, Germany* 27 (2011).

[73] Yakymets, Nataliya, Matthieu Perin, and Agnes Lanusse. "Model-driven multi-level safety analysis of critical systems." *2015 Annual IEEE Systems Conference (SysCon) Proceedings*. IEEE, 2015.

[74] David, Pierre, Vincent Idasiak, and Frederic Kratz. "Reliability study of complex physical systems using SysML." *Reliability Engineering & System Safety* 95.4 (2010): 431-450.

[75] HiP-HOPS. "Automated Fault Tree, FMEA and Optimisation Tool." (2013).

[76] Mokos, Konstantinos, et al. "Ontology-based model driven engineering for safety verification." *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2010.

[77] Bretschneider, Matthias, et al. "Model-based Safety Analysis of a Flap Control System." *INCOSE International Symposium*. Vol. 14. No. 1. 2004.

[78] Peikenkamp, Thomas, et al. "Towards a unified model-based safety assessment." *International Conference on Computer Safety, Reliability, and Security*. Springer, Berlin, Heidelberg, 2006.

[79] Zhao, Lin, et al. "Failure Propagation Modeling and Analysis via System Interfaces." *Mathematical Problems in Engineering* 2016 (2016).

[80] Liu, Jintao, et al. "Functional safety analysis method for CTCS level 3 based on hybrid automata." *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. IEEE, 2012.

[81] Boudali, Hichem, et al. "Arcade-A formal, extensible, model-based dependability evaluation framework." *13th IEEE International Conference on Engineering of Complex Computer Systems (iceccs 2008)*. IEEE, 2008.

[82] Dickerson, Charles E., Rosmira Roslan, and Siyuan Ji. "A formal transformation method for automated fault tree generation from a UML activity model." *IEEE Transactions on Reliability* 67.3 (2018): 1219-1236.

[83] Clegg, Kester, et al. "Integrating Existing Safety Analyses into SysML." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2019.

[84] Bozzano, Marco, Alessandro Cimatti, and Francesco Tapparo. "Symbolic fault tree analysis for reactive systems." *International Symposium on Automated Technology for Verification and Analysis*. Springer, Berlin, Heidelberg, 2007.

[85] Wallace, Malcolm. "Modular architectural representation and analysis of fault propagation and transformation." *Electronic Notes in Theoretical Computer Science* 141.3 (2005): 53-71.

[86] Yang, Liu, and Antoine Rauzy. "FDS-ML: A New Modeling Formalism for Probabilistic Risk and Safety Analyses." *International Symposium on Model-Based Safety and Assessment*. Springer, Cham, 2019.

[87] Gomes, Adriano, et al. "Constructive model-based analysis for safety assessment." *International Journal on Software Tools for Technology Transfer* 14.6 (2012): 673-702.

[88] Adedjouma, Morayo, and Nataliya Yakymets. "A framework for model-based dependability analysis of cyber-physical systems." *2019 IEEE 19th International Symposium on High* Assurance Systems Engineering (HASE). IEEE, 2019.

[89] Bonfiglio, Valentina, et al. "Software faults emulation at model-level: Towards automated software fmea." 2015 IEEE International Conference on Dependable Systems and Networks Workshops. IEEE, 2015.

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

[91] Dong, Li, et al. "Model-based System Reliability Analysis by using Monte Carlo Methods." *2019 Prognostics and System Health Management Conference (PHM-Qingdao)*. IEEE, 2019.

[92] Joshi, Anjali, and Mats PE Heimdahl. "Model-based safety analysis of simulink models using SCADE design verifier." *International Conference on Computer Safety, Reliability, and Security*. Springer, Berlin, Heidelberg, 2005.

[93] Leitner-Fischer, Florian, and Stefan Leue. "QuantUM: Quantitative safety analysis of UML models." *arXiv preprint arXiv:1107.1198* (2011).

[94] Beer, Adrian, et al. "Analysis of an Airport Surveillance Radar using the QuantUM approach." (2012).

[95] Fondazione Bruno Kessler. "xSAP User Manual" (2019).

[96] Bittner, Benjamin, et al. "The xSAP safety analysis platform." *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, Berlin, Heidelberg, 2016.

[97] Yang, Liu, Antoine Rauzy, and Cecilia Haskins. "Finite degradation structures: a formal framework to support the interface between MBSE and MBSA." *2018 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, 2018.

[98] Rauzy, Antoine, and Chaire Blériot-Fabre. "Model-based safety assessment: Rational and trends." *2014 10th France-Japan/8th Europe-Asia Congress on Mecatronics (MECATRONICS2014-Tokyo)*. IEEE, 2014.

[99] Batteux, Michel, Tatiana Prosvirnova, and Antoine Rauzy. "From models of structures to structures of models." *2018 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, 2018.

[100] Lisagor, Oleg, Linling Sun, and Tim Kelly. "The illusion of method: Challenges of model-based safety assessment." *28th international system safety conference (ISSC)*. System Safety Society, 2010.

[101] Bozzano, Marco, et al. "Formal design and safety analysis of AIR6110 wheel brake system." *International Conference on Computer Aided Verification*. Springer, Cham, 2015.

[102] Pajic, Miroslav, et al. "Model-driven safety analysis of closed-loop medical systems." *IEEE Transactions on Industrial Informatics* 10.1 (2012): 3-16.

[103] Althoff, Matthias. Reachability analysis and its application to the safety assessment of autonomous cars. Diss. Technische Universität München, 2010.

[104] Whittle, Jon, John Hutchinson, and Mark Rouncefield. "The state of practice in model-driven engineering." IEEE software 31.3 (2013): 79-85.

[105] Scippacercola, Fabio. "A Model-Driven Methodology for Critical Systems Engineering." (2016).

[106] Jensen, Jeff C., Danica H. Chang, and Edward A. Lee. "A model-based design methodology for cyber-physical systems." *2011 7th International Wireless Communications and Mobile Computing Conference*. IEEE, 2011.

[107] France, Robert, and Bernhard Rumpe. "Model-driven development of complex software: A research roadmap." *Future of Software Engineering (FOSE'07)*. IEEE, 2007.

[108] Larsen, Peter Gorm, et al. "Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project." *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*. IEEE, 2016.

[109] Akdur, Deniz, Vahid Garousi, and Onur Demirörs. "A survey on modeling and model-driven engineering practices in the embedded software industry." *Journal of Systems Architecture* 91 (2018): 62-82.

[110] Manjunath, T. V., and P. M. Suresh. "Structural and thermal analysis of rotor disc of disc brake." *International journal of innovative research in science, Engineering and Technology* 2.12 (2013): 2319-8753.

[111] Bhatt, Devesh, et al. "Model-based development and the implications to design assurance and certification." *24th Digital Avionics Systems Conference*. Vol. 2. IEEE, 2005.

[112] Leveson, Nancy G. Engineering a safer world: Systems thinking applied to safety. The MIT Press, 2016.

[113] ARP4761, S. A. E. "Guidelines and Methods for Conducting the Safety Assessment Process on Airborne Systems and Equipment." *USA: The Engineering Society for Advancing Mobility Land Sea Air and Space* (1996).

[114] Liebel, Grischa, et al. "Assessing the state-of-practice of model-based engineering in the embedded systems domain." *International Conference on Model Driven Engineering Languages and Systems*. Springer, Cham, 2014.

[115] Rugina, Ana-Elena, Karama Kanoun, and Mohamed Kaâniche. "A system dependability modeling framework using AADL and GSPNs." *Architecting Dependable Systems IV*. Springer, Berlin, Heidelberg, 2007. 14-38.

[116] ISO, ISO26262. "26262: Road vehicle - Functional safety." *International Standard ISO/FDIS* 26262 (2011).

[117] Ladkin, Peter B. "An overview of IEC 61508 on E/E/PE functional safety." *Bielefeld, Germany* (2008).

[118] Hai, Bhuiyan Shameem Mahmood Ebna, and Markus Bause. "Finite element model-based structural health monitoring (SHM) systems for composite material under fluid-structure interaction (FSI) effect." 2014.

[119] Gardner, Paul. On novel approaches to model-based structural health monitoring. Diss. University of Sheffield, 2018.

[120] Heitner, Barbara, et al. "Probabilistic modelling of bridge safety based on damage indicators." *Procedia engineering* 156 (2016): 140-147.

[121] Liao, Chung-Min, Yu-Hui Chiang, and Chia-Pin Chio. "Model-based assessment for human inhalation exposure risk to airborne nano/fine titanium dioxide particles." *Science of the total environment* 407.1 (2008): 165-177.

[122] Melzner, Jürgen, et al. "Model-based construction work analysis considering process-related hazards." *2013 Winter Simulations Conference (WSC)*. IEEE, 2013.

[123] Ericson, Clifton A. *Hazard analysis techniques for system safety*. John Wiley & Sons, 2015.

[124] DoD, U. S. "Mil-std-882e, department of defense standard practice system safety." *US Department of Defense* (2012).

[125] Fleming, Cody Harrison, et al. "Safety assurance in NextGen and complex transportation systems." *Safety science* 55 (2013): 173-187.

[126] Sinha, Purnendu. "Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives." *Reliability Engineering & System Safety* 96.10 (2011): 1349-1359.

[127] Krach, Sebastian Dieter. "Model-based architecture robustness analysis for software-intensive autonomous systems." *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2017.

[128] Leveson, Nancy, and John Thomas. "An STPA primer." *Cambridge, MA* (2013).

[129] Hollnagel, Erik. FRAM, the functional resonance analysis method: modelling complex socio-technical systems. Ashgate Publishing, Ltd., 2012.

[130] Panesar-Walawege, Rajwinder Kaur, Mehrdad Sabetzadeh, and Lionel Briand. "Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation." *Information and Software Technology* 55.5 (2013): 836-864.

[131] Broy, Manfred, et al. "What is the benefit of a model-based design of embedded software systems in the car industry?" *Emerging Technologies for the Evolution and Maintenance of Software Models*. IGI Global, 2012. 343-369.

[132] Barbieri, Giacomo, Cesare Fantuzzi, and Roberto Borsari. "A model-based design methodology for the development of mechatronic systems." *Mechatronics* 24.7 (2014): 833-843.

[133] Jayakumar, Athira Varma. "Systematic Model-based Design Assurance and Property-based Fault Injection for Safety Critical Digital Systems." (2020).

[134] Hutchinson, John, et al. "Empirical assessment of MDE in industry." *Proceedings of the 33rd international conference on software engineering*. 2011.

[135] Bozzano, Marco, et al. "Formal safety assessment via contract-based design." *International Symposium on Automated Technology for Verification and Analysis*. Springer, Cham, 2014.

[136] Avizienis, Algirdas, et al. "Basic concepts and taxonomy of dependable and secure computing." *IEEE transactions on dependable and secure computing* 1.1 (2004): 11-33.

[137] ARP4754A, S. A. E. "Guidelines for Development of Civil Aircraft and Systems. 2010." (2019).

[138] Walter, Max, Markus Siegle, and Arndt Bode. "OpenSESAME—the simple but extensive, structured availability modeling environment." *Reliability Engineering & System Safety* 93.6 (2008): 857-873.

[139] Jones, G., et al. "Human-Rated Automation and Robotics." *Jet Propulsion Laboratory, JPL D-66871, Pasadena, CA* (2010).

[140] Cimatti, Alessandro, et al. "NuSmv: a reimplementation of smv." Proceeding of the International Workshop on Software Tools for Technology Transfer (STTT-98). 1998.

[141] Katoen, Joost-Pieter, et al. "The ins and outs of the probabilistic model checker MRMC." *Performance evaluation* 68.2 (2011): 90-104.

Appendix

The following sample of MBSA works is classified based on the patterns of fault effect model discussed in section 5.3.4. Note that, some works appear in more than one cells because the fault effects are defined at multiple level of abstraction.

	Off-nominal	**Hybrid**	**Nominal**
Function	[43], [90]	[41], [46], [55]	NA
Architecture	[27], [43], [45], [64], [75], [85], [86], [87], [135]	[1], [41], [46], [55], [57], [63], [72], [79], [80]	[89], [92], [91]
Component	[64]	[1], [41], [46], [57], [63], [72], [79]	[39], [54], [56], [77], [78]

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01-03-2021	**2. REPORT TYPE** Contractor Report	**3. DATES COVERED** *(From - To)* *2016-2020*

4. TITLE AND SUBTITLE Defining and Reasoning about Model-based Safety Analysis: A Review	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER NNX16AK47A
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Minghui Sun; Cody H. Fleming; Milena Milich	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, Virginia 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA-CR-20205009755

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified
Subject Category: 62
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES
Langley Technical Monitor: C. Michael Holloway

14. ABSTRACT

Model-based safety analysis (MBSA) has been around for over two decades. The benefits of MBSA have been well-documented in the literature, such as tackling complexity, introducing Formal Methods to eliminate the ambiguity in the traditional safety analysis, using automation to replace the error-prone manual safety modeling process, and ensuring consistency between the design model and the safety model. However, there is still a lack of consensus on what MBSA even is. This paper provides an approach towards developing such a consensus.

15. SUBJECT TERMS
model, model-based, safety engineering, safety, safety analysis

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk(email help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	49	(757) 864-9658