Towards a Comprehensive Model Based Safety Assessment: A STPA-informed approach

---

A

Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Doctor of Philosophy

by

Minghui Sun

August 2022

APPROVAL SHEET

This

is submitted in partial fulfillment of the requirements
for the degree of

Author:

Advisor:

Advisor:

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Jennifer L. West, School of Engineering and Applied Science

Abstract

With the rapidly increasing complexity of the modern safety-critical system, the "model-based" approach has gained much traction for safety assurance, thanks to advancements in the computation capability and computational techniques. We have seen many model-based applications in the industry, for example, using models to automatically generate software code and using models as 3D drawings directly for manufacture. Similarly, Model-based Safety Assessment (MBSA) has been an important research topic in the safety engineering community over the past two decades. Numerous modeling languages and verification platforms are developed to automatically translate the safety model directly from the design model for faster and more integrated safety assessment. These models have a critical role for safety assurance because safety-critical decisions are derived directly from them, and they are at the center of the evidence chain for regulation compliance. Therefore, they need to be validated before they can be used in any MBSA analysis. Unfortunately, the current MBSA literature falls short in this regard. In general, MBSA literature focuses on verifying properties and automating safety assessment, but there is relatively little focus on making sure the model [1] that is used for the analysis is valid (i.e., free from design errors that may lead to "false assurance" of the verification program) in the first place.

Therefore, there is a gap between MBSA and safety assurance caused by

[1] See the Appendix for a definition.

validation rather than verification. Although "model validation" has been around probably since simulation was introduced into engineering or even as far back as the scientific revolution, our problem is different and potentially more challenging because, in MBSA (or Engineering Design in a more general sense), there is no existing system to extract data from to validate the model in the first place.

This dissertation is motivated to bridge this gap. First, because MBSA is a loosely defined concept, a comprehensive investigation is conducted in the literature to find the *defining features* and *notable patterns* of MBSA, and more importantly, to provide evidence that there is indeed a lack of validation for the model of current MBSA approaches. Second, a safety-guided design methodology, named STPA+ (based on STPA, a well-received hazard analysis methodology), is proposed to develop at the methodology level the requirements[2] for the system design activity (not the system being designed) that, if fulfilled in the specific design application, will lead to a valid set of system specifications[3] that are ready to be translated into the model in the target MBSA language. Finally, a case study is conducted on a new Urban Air Mobility (UAM) concept to demonstrate the effectiveness of the proposed methodology.

Keywords: Model-based Safety Assessment, STPA+, Reference Architecture, Validation, Urban Air Mobility

[2]ibid.
[3]ibid.

Acknowledgements

I was going through my old application documents to UVa Fall 2016 the other day, and realized that this dissertation is exactly the answer to the research question I put in the Personal Statement when I applied for Dr. Fleming's lab. I think I have myself to thank, for being staying true to the course no matter how hard and hopeless the future had looked like. I love the song "My Way" by Frank Sinatra. It is a perfect depiction of my intellectual journey during all these years. Although the future is still unclear at this point, at least I answered the calling my way.

Second, I want to thank my parents. They are the two most important people in my life. I have not been able to see them for almost 6 years at this point. I am grateful for their understanding and support. I know they do not understand many things happening in life as their life experience is completely different from this one I am leading, but they always try, try their best to see things from my perspective. The most precious thing is not how much helpful they can be in my daily life, but really the care and the love I can feel from them, which is all I can ask for. I owe them a lot.

Third, I would like to thank Cody. He took me in in 2016, and we stick together ever since. This advisor-student relationship is more like a friendship, although I have a hard time thinking him as my friend mostly because of my Chinese cultural teaching, always putting my seniors at a higher status. But anyway, we have our ups and downs. We argued a lot. It took quite some time for us to

converge to the same page. Sometimes I felt he is annoying; sometimes I loved him like my best buddy because he is probably the one who understands my intellectual aspiration the most in this world. But most importantly, I grow in this relationship. I now understand more about safety and engineering design thanks to his continuous push to get things concrete. His words: “get your hands dirty”. I can now better articulate my question and communicate my solution thanks to his “ruthless” criticism. Without his support, I cannot go this far. I hope he also grows in this relationship.

Last but not the least, many thanks to my lovely friends Xiyuan, Yali, Wanjia, Krista, Neil, Peng and Mike who brought joy into my life for the traveling, the food adventure, the cooking, the hikes, the football game, those fishing weekends and the kayaking trips, and all the others who helped me in my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The rapid advancement of modern science and technology poses two challenges to the safety community. For one, systems are increasing in complexity more rapidly than ever. The possible states of a complex system are astronomical and make it impossible to exhaustively test out all the unsafe design errors [1]. For two, the introduction of disruptive technology makes it extremely difficult (if not impossible) for safety engineers to even imagine what can go wrong. The record shows in an industry as highly regulated as commercial aviation that the introduction of new systems or even new design features may lead to catastrophic accidents, as the two most recent Boeing 737MAX accidents are still fresh in the public's memory [2]. The general public's perception that civil aviation is the safest means of transportation results from continuous learning from previous accidents enforced by rigorous regulation and the relatively stable technology perfected throughout decades of operation.

The safety community tackles these challenges at the process level and the method level. At the process level, numerous safety industry standards (e.g. IEC 61508 [3], ISO 26262 [4], DO-178C [5], DO-254 [6], DO-331 [7] and DO-333 [8])

are developed for more rigorous engineering activities when developing safety-critical systems. Compliance must be demonstrated that the required design activities and safety activities are all sufficiently conducted in accordance with the level of safety criticality (e.g., the Development Assurance Level in ARP4754A [9] and Automotive Safety Integrity Level in ISO 26262 [4]). This approach of process compliance is widely practiced by the aviation and automotive industry, among others.

This dissertation focuses on the method level, because better methods make it easier and faster to achieve and demonstrate the process compliance. At the method level, two different streams of safety philosophy have been dominating the research world. One is *automatic inference*: since the complexity of the system has exceeded human cognitive capacity, computational models are introduced to tackle the complexity. Thanks to the rapid advancement of both the computational techniques and hardware capacity, the model-based approach, specifically Model-based Safety Assessment (MBSA), was introduced over two decades ago and had great potentials in this regard. The other is *causal reasoning*: regardless of the complexity, systems always follow certain underlying patterns, which can be used to reason about what can go wrong with the system through careful abstraction and methodical methodological aid. Multiple methods are available in the safety community, such as Functional Hazard Assessment (FHA) [10], Fault Tree Analysis [11], and Hazard and Operability Analysis (HAZOP) [12]. One prominent and probably the most successful method in this regard is the Systems Theoretic Process Analysis (STPA) [13]. Compared with other traditional methods that are mainly based on an extended usage of pure logic, the "underlying pattern" that STPA uses is hierarchical control, a well-studied scientific theory, which provides both scientific rigor and engineering intuition to reason about the possible causes of hazard.

However, both approaches have pros and cons. Automatic inference, usually

paired with a particular type of formalism, can provide a conclusion with (quantifiable and/or logical) certitude that whether the collective behavior of a given set of specifications are free from conflicts and satisfy specific properties [1]. However, because automatic inference starts with a pre-defined set of specifications, automatic inference is not effective and even counter-productive when the given specifications are incomplete, illegal (i.e., violating hard constraints), or having unidentified assumptions. On the other hand, causal reasoning, usually conducted manually, can find new scenarios and missing assumptions through the involvement of human experts with methodological guidance. However, it is tough for the manual process to tell with certitude whether a property will be satisfied or not with the resulting specifications.

Obviously, neither approach is sufficient for safety assurance on its own, but when used together, they can overcome each other's weaknesses. This dissertation stems from this observation.

1.2 Motivations

MBSA is probably the most noticeable development of automatic inference methods in the safety community over the past two decades. It is a promising direction for safety assurance in the era of complex systems. Besides tackling the increasing complexity, models in MBSA are digital proxies of the real systems and can be built even before a prototype. This makes MBSA a perfect candidate to identify design problems early before they are too costly or impossible to change later. Furthermore, the corner cases usually targeted in safety testing are either too hard to generate in the real operational environment due to their low probability or too dangerous to perform because of their severe consequences. A model-based approach can easily test these scenarios safely and

[1] See Appendix A for the definition

sufficiently in the virtual space.

However, MBSA has limitations. Similar to any other automatic inference method, it can only work with a given set of specifications. Even assuming a perfect modeling process, the input specifications can be missing or illegal. Most MBSA works leave the task of validating the input specifications to the modeler, not because it is easy but because it is an ill-structured, hard-to-generalize problem. Furthermore, models in MBSA are nothing more than a set of mathematical expressions and hence have assumptions for the model to be valid in the first place. In fact, assumptions are notoriously challenging to identify and keep track of during the engineering process. Without having these problems addressed, MBSA can lead to "false assurance". It is not a trivial problem that can be blindly brushed away, especially for safety, where accidents happen all the time because of these "surprises" that are missed in the very beginning (e.g., Arian 5 explosion [14]).

The limitation of MBSA is caused by a lack of validation of the input specifications. It is under the topic of "requirement validation". However, the general "requirement validation" community focuses more on the requirements related to a system's functionality rather than safety. Moreover, causal reasoning methods like FHA and HAZOP stop at the hazard level, and hence cannot be used to generate the specifications. FTA, a deductive causal reasoning method, is the de facto method to validate the specifications from the perspective of safety, such as the PSSA process in ARP4761 [15]. However, as argued by [13], it becomes less and less effective due to the complex interactions in the modern complex system. Even if FTA is suitable for the particular target system, it is automated away by MBSA (probably the most prominent feature of MBSA), transformed from a causal reasoning method into an automatic inference method. In our opinion, it is the irony of MBSA, solving one problem (i.e., consistency between the design model and the safety model) but only creating another (i.e., model

validity). Another causal reasoning method, and probably the most prominent one over the past decade, is STPA. STPA is well known in the safety engineering community for its power to identify inadequate design errors that might lead to a hazard. Specifically, STPA abstracts a complex system into a hierarchical control structure, a well-studied engineering construct, and uses it to pinpoint the possible places and patterns that a design can go wrong. It seems STPA is a perfect fit to address the problems of invalid specifications, complementing the weakness of the current MBSA works.

As promising as STPA is to overcome the weakness of MBSA, it is not ready to integrate with MBSA because (1) the STPA model is inexecutable which makes it impossible to run a model-based safety verification on the STPA model directly; (2) because the STPA model is informal and "different than the architectural models usually proposed for model-based system engineering today" [16], it is also difficult to integrate STPA with MBSA through model transformation; (3) STPA is a hazard analysis technique, i.e., finding out what-not-to-do, and hence can not be directly used to generate MBSA specifications which by definition is about what-to-do. Although showing great potential, all these differences between the STPA world and the MBSA world make it challenging to complement MBSA with STPA.

Therefore, this dissertation is motivated towards the integration of MBSA (automatic inference) and STPA (causal reasoning) at the methodology level.

1.3 Challenges and solutions

Conceptually, it is possible to integrate STPA and MBSA under the current paradigm. As shown in Fig.1.1, given a set of design specifications, STPA first models them into a control structure, on which the STPA hazard analysis will be conducted. Second, the specifications are revised (possibly in multiple iterations) based on

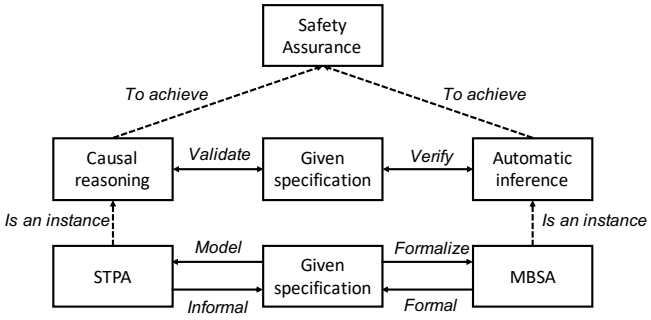


Figure 1.1: Conceptually, STPA and MBSA can be integrated.

the STPA findings to eliminate the found design errors. Once no more design errors can be found by applying STPA, the resulting specifications are passed to MBSA for safety verification. More iterations are possible if the properties are not satisfied during the MBSA analysis.

However, there are challenges to implementing such a concept for the integration of MBSA and STPA. As we will show now, the solution to these challenges (especially Challenge B and C) requires innovation at the methodology level, i.e., a paradigm shift from STPA to STPA+.

Challenge A: The scope of MBSA. Although MBSA has been around for over two decades, except for a few notable approaches such as AADL, HipHops, and AltaRica, it is still not clear to the safety community what makes an approach MBSA and what does not. Particularly, with other "model-based" disciplines rising in the past several decades, such as Model-based Systems Engineering, Model-based Design, and Formal Verification, it gets more and more challenging to tell what MBSA even is. There are so many self-claimed MBSA modeling languages that can contribute to safety assurance in so many different ways, making the situation worse. It is crucial to have a comprehensive view of the scope and the features of MBSA so that a basic generalization can be made in terms of what MBSA can do and (more importantly) cannot do for safety assurance, in order to eventually define what is needed to complement MBSA.

Solution: An analysis is conducted based on a comprehensive review of the

current literature, to define the general scope of MBSA as a collective research effort . The general scope sets the stage for the methodology innovation of this dissertation.

Challenge B: Model consistency. To start an STPA hazard analysis, a control structure must be modeled from the specifications. However, information can be lost or misinterpreted during this process, which creates a potential gap that can only be avoided through the "art of modeling", which can be error-prone and hard to review. In essence, this gap is because STPA is inherently an *analysis* technique, which by definition has to start with an existing design solution to analyze. As a matter of fact, MBSA was originated from a similar concern that the design model and safety model can be inconsistent due to such a modeling process.

Solution: A STPA-informed safety-guided design methodology, named STPA+, is developed to generate the requirements at the methodology level that, if fulfilled by the specific design application, will lead to valid system specifications. In this way, no modeling process is needed, and hence the room for inconsistency is naturally eliminated (Fig.1.2).

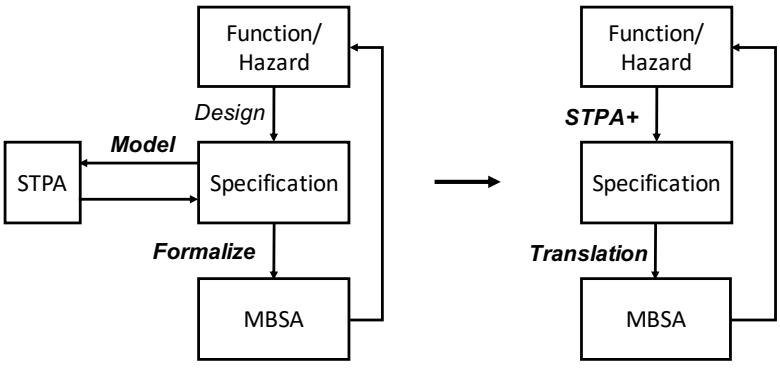


Figure 1.2: STPA+ eliminates the modeling process of STPA, which hence solves the model consistency problem.

Challenge C: Correct specifications. Replacing a hazard analysis technique (STPA) with a design methodology (STPA+) eliminates the chance to examine

the specifications after they are created, which requires that STPA+ results incorrect specifications by construction. The obvious challenge is how to achieve such "correct-by-construction" by a design methodology?

Solution: STPA+ treats a system as a control structure. Using the control structure as an underlying map, STPA+ has a way to systematically reason about the possible safety-critical scenarios without the component-level failure and puts in mechanisms to avoid the hazard and achieve the functional goal from the beginning. In this way, STPA+ complements MBSA by starting with a correct design model.

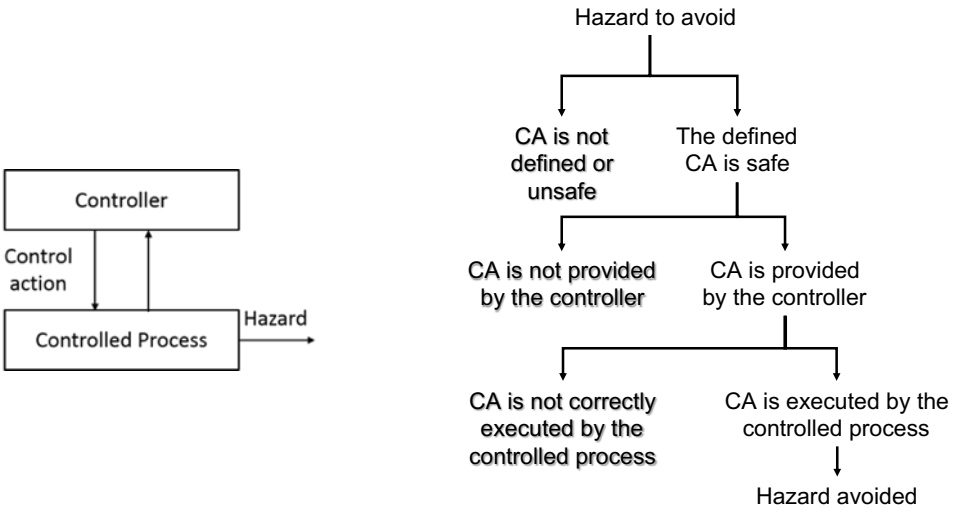
Challenge D: Formalization. The models used in MBSA are usually formal models, i.e., mathematical models. Manually translating informal specifications into a formal model is tedious and often error-prone. STPA does not result in formal specifications, a challenge to further integrate it with any MBSA languages.

Solution: This dissertation does **not** particularly address this challenge. But the output specification of STPA+ is written in a language agnostic formalism that has great potential to be automatically translated into a wide range of MBSA languages. We will address this translation problem in the future as it is the last step to truly integrate STPA+ with MBSA.

1.4 From STPA to STPA+

This dissertation addresses the scope of MBSA (Challenge A) , model consistency (Challenge B) and correct specifications (Challenge C). Challenge A will be addressed by Chapter 2, and Challenge B and C will be addressed by the design methodology STPA+. In this section, we give an overview of STPA+, specifically how STPA+ is informed by STPA.

First, STPA is based on Systems Theory that views a system in terms of hierarchical control, which leads to the fundamental observation of STPA: hazards happen because of lack of control. At one level of control, the possible causes of a hazard can be summarized as below (Fig.1.3).



(a) The control structure

(b) The possible causes of a hazard

Figure 1.3: (a) is a basic control structure that is used by STPA; (b) is how a hazard can happen caused by the different components of the control structure (the bold font on the left side). CA stands for “control action”.

1. Given a hazard, the intended control action is not defined *at the design time* or the intended control action cannot prevent the output of the controlled process from reaching the hazardous states.

2. The safe control action is defined, but not provided as intended by the controller.

   (a) The process model is improperly defined *at the design time*;

   (b) The controller is improperly defined *at the design time*.

   (c) The controller is executed improperly *at the operation time*.

3. The intended control action is provided by the controller, but executed improperly by the controlled process *at the operation time*.

4. The actuation path or the observation path is executed improperly *at the operation time*. [2]

STPA+, as a design methodology, is to design out these causal factors. It focuses on getting the correct definition for 1, 2(a), and 2(b) at the design time and leaves out 2(c), 3, and 4 of improper execution at the operation time. This is a decision made by choice, not a limitation. First, the target problem of "invalid specification" is a definition problem rather than an execution problem. Second, given the correct definition, the execution problem (assuming perfect implementation) can only be caused by component failure or human error, whose occurrence cannot be prevented by a design methodology. However, the impact of failures can be mitigated and is already covered by MBSA, e.g., the failure-safe principle. Finally, human error is covered by Human Factor as a discipline. Therefore 1, 2(a), and 2(b) are a proper scope of STPA+.

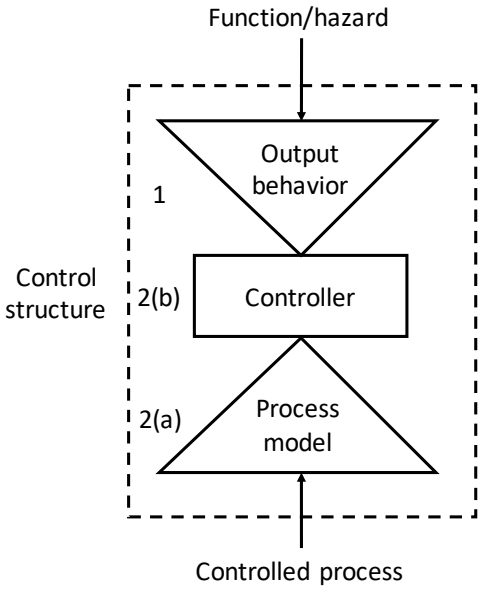


Figure 1.4: Designing out the causal factors 1, 2(a) and 2(b) described on Page 30 is consistent with the functional design of the control structure.

Furthermore, to design out the casual factors, STPA+ follows the same process as the functional design of the control structure. As shown in Fig.1.4, the

[2]We assume perfect definition of the actuation path and the observation path.

functional design of a control structure is subject to constraints from two directions: *top-down* and *bottom-up*, where the former represents the functional goal from the higher level regardless of how the control structure is implemented, and the latter represents the real process to be controlled (called "controlled process" hereafter). Neither the top-down nor the bottom-up constraints can be changed by the control structure; therefore, they are called *prescriptive constraints* and *descriptive constraints* respectively hereafter. With the prescriptive constraints and the descriptive constraints defined, the functional design of the control structure eventually boils down to the design of the controller, which is to adequately interpret the implications of the constraints from both directions and define a control algorithm to generate the control action that can satisfy all the constraints at the same time.

STPA+ follows the same process to design out the three causal factors 1, 2(a), and 2(b) described on page 30. *First* (corresponding to Causal factor 1), the hazard has to be avoided by the control structure as a whole regardless of the specific implementation. Therefore, the hazard can be translated into the prescriptive constraints and enter the design process the same way as the functional goal. The only question is, on what the prescriptive constraints are defined? The functional design defines them on the behavior of the output variable (called *output behavior* hereafter, the top-down arrow in Fig.1.4), but STPA defines them on the control action by assuming the "worst-case scenario" for the controlled process. STPA+ follows the functional design and defines the prescriptive constraints translated from the hazard on the output behavior. In this way, no knowledge or assumption is required about the controlled process, and the translation from the hazard to the constraints on the output behavior is more direct than to the control action. More importantly, the resulting constraints will enter the design process just like the constraints from the functional goal, enabling seamless integration between the safety and the design. Having decided to define the hazard on the output behavior, the remaining question is, how to

adequately translate the hazard into the prescriptive constraints on the output behavior?

Second (corresponding to Causal factor 2(a)), the descriptive constraints from the real process are manifested as a model of the real process (called "process model" hereafter, the bottom-up arrow in Fig.1.4). The process model has to adequately represent the real process in the first place for both functional design and safety. Traditionally, in theoretical control and most "model-based" work, the process model is usually a mathematical representation of the real process, called system dynamics. However, the process model should be more than a mathematical model especially for safety-critical systems because (a) the mathematical model may have assumptions that have to be true for the model to be valid in the first place, (b) constraints in the mathematical model may change over time or attributes in the environment. For safety, an inadequate process model is much more dangerous than no model at all. Therefore, the question is, how to adequately define the model of the process under control?

Third (corresponding to causal factor 2(b)), same as the functional design, this is to design the controller to find the control action to satisfy the constraints from both directions. However, the controller is more than a "control algorithm" in theoretical control. It decides not only the right value (or magnitude) of the control action but also the right timing to issue the control action and validity condition before the control action can be issued, which are all subject to the prescriptive constraints and the descriptive constraints. Therefore, the question is, how to adequately define the controller to address the constraints from both directions?

Inadequacy in addressing any of the causal factors above will lead to invalid specifications, eventually leading to false assurance. For example, causal factor 1 may lead to inadequate safety property, causal factor 2(a) may lead to in-

adequate representation of the reality, and 2(b) may lead to inadequately constrained control algorithm concerning the constraints from either direction. As a result, three methods are developed for STPA+ to address the three casual factors respectively (Fig.1.5).

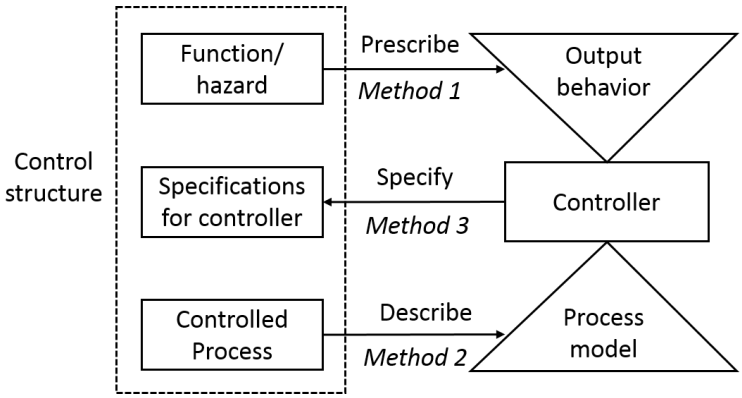


Figure 1.5: STPA+ is comprised of three methods to address the three casual factors respectively when designing a system.

Method 1: A method to derive the *prescriptive constraints* from the hazard and the function for the control structure as a whole. This will be explained in Chapter 3.

Method 2: A method to define the process model as the *descriptive constraints* for the control structure as a whole. This will be explained in Chapter 4.

Method 3: A method to derive the specifications for the controller so that the constraints from both directions will be enforced by the control actions. This will be explained in Chapter 5.

When using STPA+ to design a system (Fig.1.6), given the functional goal and the hazard under study, the designer first applies Method 1 to derive the prescriptive constraints for the control structure. Then based on a basic scientific understanding about the controlled process, the designer applies Method 2 to derive the descriptive constraints (i.e. the process model) for the control structure. Finally, the designer follows Method 3 to define the specifications for

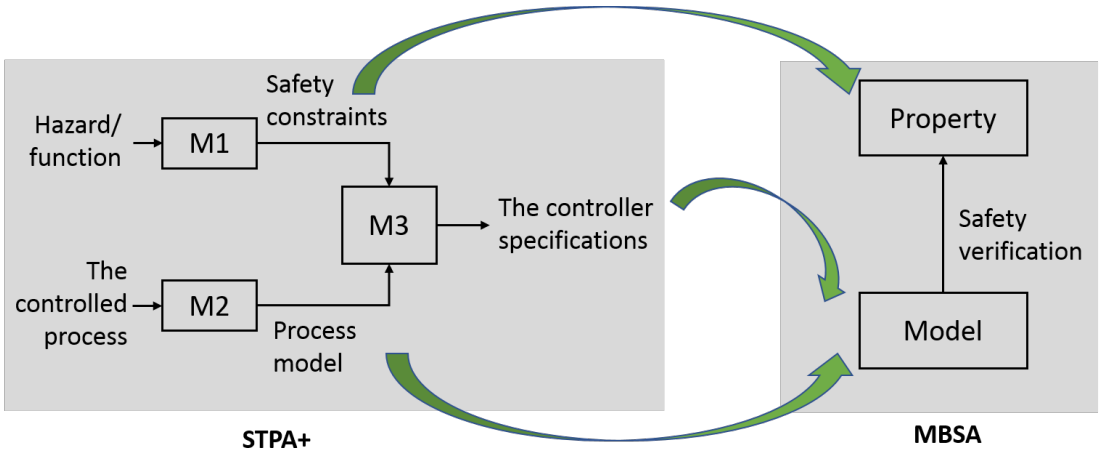


Figure 1.6: The process that how STPA+ is used to design a system and how the results can support MBSA. M1, M2 and M3 are short for Method 1, 2 and 3.

the controller so that the constraints from both directions will be satisfied by the control structure through the issued control action.

In the future, the safety constraints will be translated as the properties that a MBSA program will verify against, and the process model and the controller specifications will be translated as the input model of the MBSA program.

1.5 Contribution

The mainstream MBSA approaches (Fig.1.7) start with a given design solution by formalizing it into a design model in the target languages; then faults and failures are modeled to achieve a safety model; finally, verification is conducted to see whether the given properties (functional or safety, deterministic or probabilistic) are satisfied.

However, there is a gap. Even assuming MBSA (or formal verification in a more general sense) can exhaustively check all the possible scenarios captured by the design solution, there are still open challenges. If certain safety-critical scenarios are not included or considered in the given design solution (the left shoulder of Fig.1.7), or the properties do not correctly reflect the hazard (the

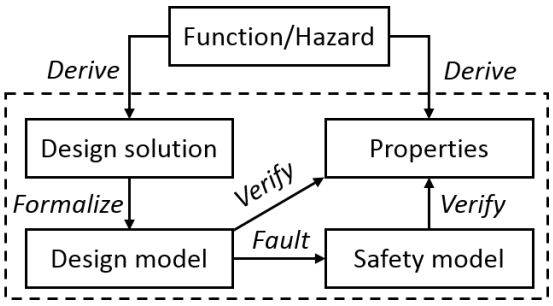


Figure 1.7: Given the function and the hazard, the mainstream MBSA approaches (in the dotted box) contribute to safety assurance by verifying the given design solution against the given properties.

right shoulder of Fig.1.7), the results of MBSA cannot be fully trusted for safety assurance.

This dissertation contributes to the literature by bridging that gap. Overall, this dissertation develops a safety-guided design methodology (called STPA+) to identify and address the safety-critical scenarios without component-level failure for the definition of the design solution. Compared to the original STPA, STPA+ derives the design solution directly to avoid the potential inconsistency between the design model and the STPA model. Moreover, STPA+ provides better methodological support in refining the hazardous scenarios associated with decision making and control into the safety-critical scenarios that the design solution can directly act upon. Therefore, STPA+ contributes to identifying and addressing "hazards without failure".

Specifically, STPA+ is comprised of three methods. Method 1 converts the logic of the STPA "unsafe control action" to "what is safe". It derives safety constraints from the hazard to make sure the safety constraints adequately reflect the hazard under study. Compared to the original STPA, Method 1 generates the safety constraints directly instead of indirectly from the "unsafe control action". The method provided to define the constraints on the start and stop times is a refined way to identify the "context" associated with the original STPA "unsafe control action". Compared with the guide words of the traditional hazard identifica-

tion techniques (e.g., HAZOP), Method 1 has a more precise and more concrete structure that explains why a specific intended output behavior starts/stops too early/late, which hence is also a contribution to the general hazard identification literature.

Method 2 defines the model of the controlled process to make sure the model is properly constrained both explicitly and implicitly. Compared with the original STPA, Method 2 has a specific mathematical construct of the controlled process, making the analysis more precise and less ambiguous. Furthermore, compared with traditional analytical approaches (e.g., theoretical control) that usually take a model as given, Method 2 focuses on the definition of the boundary conditions (i.e., operational envelop) and the assumptions of the model of the controlled process. Therefore, Method 2 contributes to both the safety community and the general model-based design community.

Method 3 defines a safe controller by providing a reference architecture to ensure the controller defined based on the reference architecture has all the safety-critical scenarios (without component-level failure) addressed. Compared with the original STPA, which only provides a small size of guide words in identifying design errors of a controller, Method 3 provides more detailed support to identify the design errors of a controller and addresses them from the beginning with a reference architecture. To the best of our knowledge, there are no such works in the current literature, which makes Method 3 a contribution to the safety community.

Eventually, Method 1 will strengthen the right shoulder of Fig.1.7; Method 2&3 will strengthen the left shoulder of Fig.1.7. Together, "STPA+MBSA" will provide a strong and comprehensive argument for the safety assurance of a safety-critical system.

Chapter 2

Literature review

2.1 Characterizing and critiquing MBSA

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 making sure the consistency between the design model and the safety model [17]. However, there is still a lack of consensus on what MBSA even is. Major modeling languages such as AADL-EMV2 [18–20], AltaRica [21–25] and HipHops [26, 27] are generally considered as MBSA, which according to [28] are the only three languages that"have matured beyond the level of research prototypes". However, a question such as what makes them MBSA is left unanswered. For example, is Formal Methods applying 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 discipline can claim as long as the work uses a computer model and is safety-related (e.g., [29, 30]). This is good to the extent that differ-

ent schools of expertise enrich MBSA as an active research topic. However, this also jeopardizes the identity of MBSA as a leading research thrust of the System Safety Engineering community. For this reason, a comprehensive review was conducted on MBSA to decide what MBSA can do and cannot do.

As a summary, this work characterizes MBSA with the following *defining features* and *notable patterns*. **Further details can be found in Appendix B:** "Defining and Reasoning about Model-based Safety Analysis: A Review".

- Three defining features are identified that an MBSA work *must* have:
    1. Verifying fail-safe property, i.e., no single point failure and the overall system must satisfy the risk objective.
    2. Assuring architecture consistency between the design model and the safety model, which is the initial motivation of MBSA.
    3. Automating safety analysis such as FTA, FMEA, and Common Cause Analysis.
- Five notable patterns are identified that an MBSA work *may* have:
    1. As important as deductive analysis is, current MBSA literature is overwhelmingly inductive analysis.
    2. A general framework is created to summarize how the literature models fault activation from the component and fault effect on the component.
    3. Three ways to achieve architecture consistency are identified: injected, referred, and coupled.
    4. Three types of the formalism of the modeling languages are summarized: explicit, implicit, and comprehensive.
    5. Nine types of safety analysis are identified from the literature.

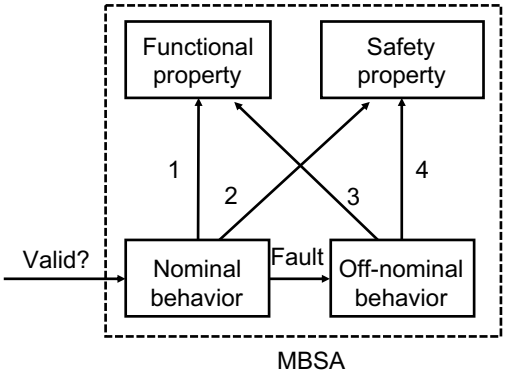


Figure 2.1: Four types of automatic inference are identified in the MSBA literature.

2.2 Contribution and finding

Contribution: We characterized MBSA with three defining features and five notable patterns. This characterization gives MBSA a discernible identity and separates it from other "model-based" safety-related work, which helps prevent MBSA from evolving into a research buzzword.

Finding: One of the most important findings of this work is that the current MBSA works are overwhelmingly *automatic inference*, and there is a lack of explicit *causal reasoning* in current MBSA literature to make sure the specifications are valid in the first place.

The automatic inference is necessary because it *verifies* whether the modeled behavior (nominal or off-nominal) will satisfy the functional and safety properties. Fig.2.1 are the four possible types of automatic inference, which are all addressed by the MBSA literature.

- Arrow 1 focuses on verifying 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 ensures that the designed behavior in nominal conditions will not lead to hazardous situations. For example, [30] conducted a series of

safety assessments to prove that all the possible trajectories of the autonomous cars are not in conflict by using reachability analysis.

- Arrow 3 focuses on verifying the “fail-operational” [31] 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 [32], a dependability concept that is closely related to safety.

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

However, the MBSA literature is significantly unbalanced in terms of addressing the causal reasoning explicitly, which, as shown in Fig.2.1 determines whether the model is valid in the first place. More specifically, the automatic inference can only work with a set scope of specifications, while it relies on the causal reasoning to set the scope. [33] calls for a systematic approach to achieve confidence in the completeness of the analysis. As pointed out by [34], completeness of the causal factors may only be proven concerning those captured. “If a failure mode is not even part of the formal model, then it is impossible to reason about it. However, finding a complete set of failure modes for a given component is not an easy task.” In fact, most of the MBSA innovation focuses on model specification notations and algorithms for possible manipulations of the models [35]. 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 discipline. “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 [36]". This unbalanced development between the automatic inference and the explicit causal reasoning prevents applying MBSA to the safety assurance in the actual industry practice. More explicit causal reasoning techniques need to be developed.

Chapter 3

Method 1: Deriving the prescriptive constraints from the hazard and the functional goal

3.1 Preliminary

First of all, by “prescriptive constraints”, we mean the constraints that are defined without considering the implementation details of the system.

Second, like STPA, STPA+ is designed also from the systems-theoretic perspective: any system can be modelled with a (hierarchical) control structure. When the system is modeled as a control structure, both the function that the system seeks to achieve and the hazard that the system seeks to avoid are defined on the output variable of the controlled process (called “output variable” for short hereafter), because the output variable is the goal of the control structure. For example, for the hazard of “a power plant being too hot”, the output variable to be controlled is the temperature of the power plant, and the safety requirement is that the temperature has to be lower than a certain level; for the hazard of “an airplane flying too low to the terrain”, the output variable to control

is the altitude of the airplane, and the safety requirement can be defined as the altitude has to be above a certain altitude.

In the context of the control structure, we make the following definitions:

- Output behavior: The evolution of the output variable along the time axis[1] $y(t)$ where $t \in [st, sp]$.
  - $y(t)$ is called the *dynamic trajectory* of the output behavior.
  - st is the *start time* of the output behavior.
  - sp is the *stop time* of the output behavior.
- Functional constraints: The prescriptive constraints that are derived from the functional goal and defined for the output behavior. If satisfied by the output behavior, the functional goal will be achieved.
- Safety constraints: The prescriptive constraints that are derived from the hazard and defined for the output behavior. If satisfied by the output behavior, the hazard will be avoided.

Translating the functional goal into the functional constraints is straight forward. We focus on how to derive the safety constraints from the hazard in this chapter.

3.2 From the Unsafe Control Action to the safety constraints

To reiterate, the safety constraints are defined for the output behavior $y(t)$ where $t \in [st, sp]$. Therefore, the question is how to derive constraints for $\{y(t), st, sp\}$ from the hazard. STPA has four types of Unsafe Control Action (UCA), which are basically four ways that a control action can lead to a hazard. The idea is that,

[1] Here, we assume the output variable is continuous. A discrete variable is just a special case of a continuous one

if a control action can be constrained in a way so that all the four types of UCA can be avoided, then the resulting constraints are the safety constraints that the control structure should satisfy. [37] uses a simliar to derive safety constraints from the UCAs.

3.2.1 The implications of the UCA

STPA has four types of UCA that are “provably complete” [16].

UCA1: Not providing the control action leads to a hazard.

Cause 1-1: The necessary control action to avoid the hazard is not defined.

Implication 1-1: There must be at least one control action defined for each hazard.

Cause 1-2: The necessary control action to avoid the hazard is defined, but the necessary “must-start” condition is not defined or defined incorrectly.

Implication 1-2: The “must-start” condition (if applicable) must be correctly defined for each control action.

UCA2: Providing the control action leads to a hazard.

Cause 2-1: A defined control action has a “must-not-start” condition to avoid a hazard, which however is not defined or defined incorrectly.

Implication 2-1: The “must-start” condition (if applicable) must be correctly defined for each control action.

Cause 2-2: The performance (or value) of the control action is defined incorrectly.

Implication 2-2: The performance constraints of the control action must be correctly defined.

UCA3: Providing a potentially safe control action but too early, or too late.[2].

Cause 3: "A potentially safe control action" means the control action is not in "must-not-start" condition, and is hence either in the "must-start" condition or "can-start" condition (a condition that is neither must-start nor must-not-start). This UCA happens because the safe time interval for the control action to start is not defined or defined incorrectly.

Implication 3: The acceptable time intervals for "must-start" and "can-start" condition must be correctly defined.

UCA4: The control action lasts too long or is stopped too soon (for continuous control actions, not discrete ones).

Cause 4-1: Similar to the start scenarios, the "must-stop" or "can-stop" or "must-not-stop" condition to avoid the hazard is not defined or defined incorrectly.

Implication 4-1: The the "must-stop" or "can-stop" or "must-not-stop" condition (if applicable) must be correctly defined.

Cause 4-2: Similar to the start scenarios, the acceptable time intervals associated with the "must-stop" condition and the "can-stop" condition are not defined or defined incorrectly.

Implication 4-2: The acceptable time intervals for "must-stop" and "can-stop" conditions must be correctly defined.

3.2.2 The safety constraints

The implications of UCA are in fact the constraints that if is satisfied, a control action will be safe. Therefore, we define the safety constraints based on these implications.

[2]The "wrong order" in the original STPA definition will be addressed in the future work

- (mst, mst_T): A set of conditions when the output behavior must start and a time window within which the output behavior must start.

  For example, a car sitting in the intersection at green light waiting for the left turn *must start* once the light turns yellow and before it turns red.

- (nst, nst_T): A set of conditions when the output behavior must not start and a duration of such conditions.

  For example, a car must not start the left turn when the light is the red, until the light turns green.

- (cst, cst_T): A set of conditions when the output behavior can start; if the output behavior is to start, a time window within which the output behavior must start.

  For example, a car sits at the intersection by itself waiting for the left turn. It can make the left turn when the light is green, but it does not have to. But if it choose to turn left, it has to do it before the light turn red.

- (msp, msp_T): A set of conditions where the output behavior must stop and a time window within which the output behavior must stop.

  For example, a car (just initiated the left turn) must stop the left turn when it spots another car coming in with the right of way, before it gets in the way of the other car.

- (nsp, nsp_T): A set of conditions where the output behavior must not stop and a duration of such conditions.

  For example, a car must not stop the left turn when it is in the lanes for straight driving, until it is completely out of the intersection.

- (csp, csp_T): A set of conditions when the output behavior can stop; if the output behavior is to stop, a time window within which the output behavior must stop.

For example, a car can both make a left turn or take a U-turn. It initiates with the left turn but it can stop the left turn (and make a U turn) before it exits the intersection.

- pc: A set of constraints on the intended performance of the output behavior to avoid a hazard.

  For example, a car initiates the left turn at the green light, but driving too low to exit the intersection.

As a result, the output behavior $y(t)$ where $t \in [st, sp]$ must satisfy the safety constraints:

$$\begin{cases} y(t) \in pc \\ st \in mst\_T \vee st \in cst\_T \wedge st \notin nst\_T \\ sp \in msp\_T \vee sp \in csp\_T \wedge sp \notin nsp\_T \end{cases}$$

3.3 Deriving the safety constraints from the hazard

Although we have explained what the safety constraints for the output behavior should look like, how the safety constrains can be derived from the hazard is still unanswered. We answer this question in this section by first explaining how violating the safety constraints may lead to the hazard, and then reasoning backward about how the safety constraints can be derived from the hazard.

3.3.1 From safety constraints to hazard

We take a close look at how violating the safety constraints can lead to hazard.

- If the performance constraints pc are violated by the dynamic trajectory $y(t)$, the hazard will happen.

Implication: After the intended output behavior starts and before it stops, hazard happens if $y(t) \notin pc$.

- If the intended output behavior does not start before the must-start time window mst_T expires, the hazard will happen.
  Implication: Since the intended output behavior has not started yet, the hazard happens because the dynamic trajectory of the output variable before the intended output behavior violates its corresponding performance constraints. We call the evolution of the output variable before the intended output behavior the "*in-behavior*" in this dissertation. Therefore, the must-start condition and time window is defined to prevent the in-behavior from violating its own performance constraints.

- If the intended output behavior starts during the must-not-start time window nst_T, the hazard will happen.
  Implication: Since the hazard happens after the intended output starts, it is caused by the intended output behavior violating its own performance constraints. Therefore, the must-not-start condition and time window is defined to prevent the intended output behavior from violating its own performance constraints at the very beginning.

- If the intended output behavior does not stop before the must-stop time window msp_T expires, the hazard will happen.
  Implication: Since the hazard happens when the intended output behavior is still ongoing, it is caused by the intended output behavior violating its own performance constraints. Therefore, the must-stop condition and time window is defined to prevent the intended output behavior from violating its own performance constraints.

- If the intended output behavior stops during the must-not-stop time window nsp_T, the hazard will happen.
  Implication: Since the hazard happens after the intended output behav-

ior stops, it is caused by the dynamic trajectory of the output variable after the intended output behavior violating its corresponding performance constraints. We call the evolution of the output variable after the intended output behavior the "*out-behavior*" in this dissertation. Therefore, the must-not-stop condition and time window is defined to prevent the out-behavior from violating its own performance constraints.

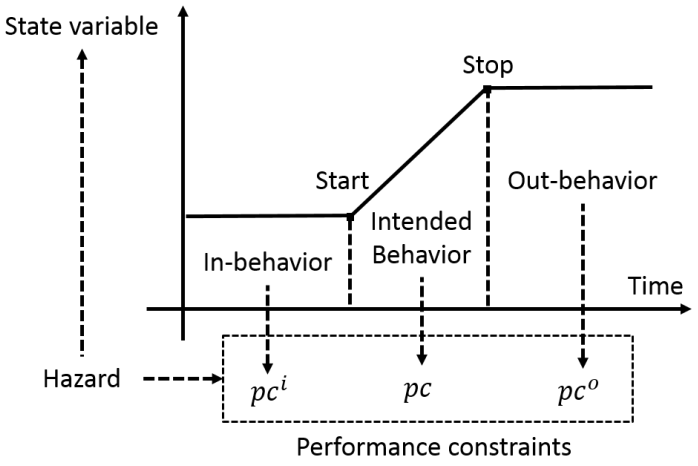


Figure 3.1: The intended output behavior can cause the hazard by unsafe start time, unsafe stop time and unsafe dynamic trajectory between the start time and the stop. The safety constraints on the start time and stop time of the intended output behavior are affected the performance constraints on the in-behavior and out-behavior respectively. The dotted arrow means "derive".

As shown in Fig.3.1, the performance constraints for the intended output behavior, the in-behavior and the out-behavior are pc, pc^i and pc^o. Hazard happens when pc, pc^i or pc^o is violated. Because the start time is also the stop time for the in-behavior, the safety constraints on the start time of the intended output behavior is determined by the both pc and pc^i. Similarly, the safety constraints on the stop time of the intended output behavior is determined by both pc and pc^o. For example, "too close to the air traffic" has different minimal distances (i.e. the performance constraint) for "approach", "descent" and "landing". Obviously, the airplane can only be undergoing one of the three behaviors at one specific time, and the applicable performance constraint (i.e. the minimal distance) depends on which behavior is ongoing. Furthermore, these three behaviors are

temporally adjacent. The start time of the descent must make sure not only the constraints on the “descent” are satisfied, but also the constraints on the “approach” cannot be violated by the time the airplane stops the “approach” and starts the “descent”. The stop time of the “descent” must make sure not only the constraints on the “descent” are satisfied, but also the constraints on the “landing” cannot be violated when the airplane starts the “landing”.

In summary, the safety constraints are defined to prescribe the start time, the stop time and the dynamic trajectory of the intended output behavior. Violating the safety constraints leads to the hazard in the following ways:

- The safety constraints on the start time is violated, causing the violation of pc^i or/and pc.
- The safety constraints on the stop time is violated, causing the violation of pc^o or/and pc.
- The safety constraints on the dynamic trajectory is violated, causing the violation of pc.

3.3.2 From hazard to safety constraints

The previous section explained how violating safety constraints may lead to hazard. We reason backward in this section about how to derive the safety constraints from the hazard.

In general, the safety constraints of the intended behavior are derived from the hazard as shown in Fig.3.2. First, given the hazard under study, the performance constraints (pc^i, pc and pc^o) are defined for in-behavior, the intended behavior and the out-behavior. Then the must-start condition and the associated time window (mst and mst_T) are determined based on pc^i; the must-not-start condition, the must-stop condition, and the associated time windows

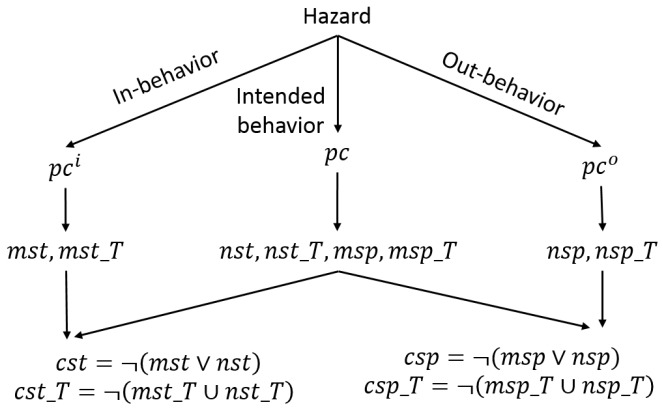


Figure 3.2: The process to derive the safety constraints from the hazard.

(nst, nst_T, msp and msp_T) are determined based on pc; the must-stop condition and the associated time window (msp and msp_T) are determined based on pc^o. Finally, the can-start condition, can-stop condition and the associated time windows (cst, cst_T, csp and csp_T) are determined based on the must-start condition, must-not-start condition, must-stop condition, must-not-stop condition and the associated time windows.

Must-start. The must-start condition mst and the time window mst_T are defined to prevent the dynamic trajectory of the in-behavior from violating its own performance constraints pc^i. If the prediction of the in-behavior shows that the dynamic trajectory of the in-behavior will violate pc^i at some time in the future, the intended behavior must start before that time to stop the in-behavior in order to avoid the violation. In some cases, the violation is projected in the far future where many uncertain factors may change the prediction of the in-behavior. It is possible to have "look-ahead time", meaning only violation within the "look-ahead time" will be counted as a violation and used to define the mst and mst_T.

Therefore, a must-start condition mst is true if the in-behavior will *eventually* violates its performance constraints pc^i; mst_T is how much time left before such violation happens, which can be calculated based on the prediction of the dynamic trajectory of the in-behavior.

Must-not-start. The must-not-start condition nst and the time window nst_T are defined to prevent the intended behavior from starting when the performance constraints pc are not satisfied at the time the intended behavior starts. If the prediction of the in-behavior shows that the dynamic trajectory of the in-behavior will violate pc at some time in the future, the intended behavior must not start as long as pc is not satisfied.

Therefore, a must-not-start condition nst is true if the in-behavior will *immediately* violate the performance constraints pc, if the intended behavior starts; nst_T is the time window(s) when such unsatisfiability lasts, which can be calculated based on the prediction of the dynamic trajectory of the in-behavior.

Must-stop. The must-stop condition msp and the time window msp_T are defined to prevent the intended behavior from violating its own performance constraints pc. If the planned dynamic trajectory of the intended behavior is going to violate pc at some time point in the future, then the intended behavior must stop before that time in order to avoid the violation. Similar to the "must-start" condition, a "look-ahead time" is also possible here to avoid looking too far into the future.

Therefore, a must-stop condition msp is true if the intended behavior will *eventually* violates its performance constraints pc; msp_T is how much time left before such violation happens, which can be calculated based on the planned dynamic trajectory of the intended behavior.

Must-not-stop. The must-not-stop condition nsp and the time window nsp_T are defined to prevent intended behavior stops at a point where the dynamic trajectory of the out-behavior will violate its own performance constraints pc^o. If the planned dynamic trajectory of the intended behavior is going to violate pc^o at some time point in the future, then the intended behavior must not stop as

long as pc^o is not satisfied.

Therefore, a must-not-stop condition nsp is true if the out-behavior will *immediately* violate the performance constraints pc^o, if the intended behavior stops; nsp_T is the time window(s) when such unsatisfiability lasts, and can be calculated based on the planned of the dynamic trajectory of the intended behavior.

Can-start & Can-stop. The can-start and can-stop condition and the associated time windows can calculated using the following expression:

$$\begin{cases} cst = \neg(mst \vee nst) \\ cst\_T = \neg(mst\_T \cup nst\_T) \\ cst = \neg(mst \vee nst) \\ cst\_T = \neg(mst\_T \cup nst\_T) \end{cases}$$

In summary, as shown in Fig.3.2, the safety constraints of the intended behavior are defined based on the in-behavior, the intended behavior and the out-behavior and the associated performance constraints. Table.3.1 is a summary what are needed to derive the proposed safety constraints.

Finally, to reiterate, the safety constraints are defined for the intended behavior, i.e. the behavior under study. The possible in-behavior and out-behavior must be identified before the safety constraints can be defined. The possible in-behaviors and out-behaviors can be derived from the design process. But this particular derivation process is out of the scope of this dissertation.

3.4 Defining the safety constraints

In general, the safety constraints can be derived directly following Table.3.1. However, there are two practical problems to be addressed before the proposed

Table 3.1: The derivation of the safety constraints.

Safety constraints		**Derivation**
Performance constrains	pc^i, pc and pc^o	Derived directly from the hazard with respect to the in-behavior, the intended behavior and the out-behavior.
Must-start	mst and mst_T	The prediction of the in-behavior, pc^i and the look-ahead time (optional)
Must-not-start	nst and nst_T	The prediction of the in-behavior and pc.
Can-start	cst and cst_T	$cst = \neg(mst \vee nst), \quad cst\_T = \neg(mst\_T \cup nst\_T)$
Must-stop	msp and msp_T	The planned intended behavior, pc, and the look-ahead time (optional).
Must-not-stop	nsp and nsp_T	The planned intended behavior and pc^o.
Can-stop	csp and csp_T	$csp = \neg(msp \vee nsp), \quad csp\_T = \neg(msp\_T \cup nsp\_T)$

safety constraints can be fully applied to real problems.

The first is mst and nst (same as msp and nsp) can both be true at the same time, because they are defined with respect to different behaviors (i.e. the in-behavior and the intended behavior) subject to potentially different performance constraints (i.e. pc^i and pc). Obviously, it is not acceptable to have such conflict in a set of safety constraints.

The second is that, in real practice there are usually sub-hazards under a general hazard. For example, the hazard of “inadequate altitude of an airplane” can have multiple sub-hazards such as inadequate altitude to the airspace boundary, to the terrain, to the weather, etc. As a result, the safety constraints with respect to the hazard of “inadequate altitude” has to integrate all the sub-requirements from the sub-hazards. However, the sub-requirements with respect to one sub-hazard may also pose conflicts to the sub-requirements from other sub-hazard. Therefore, there must be a way to integrate the (sub-)requirements so that their will not be any internal conflicts among them.

3.4.1 The single hazard situation

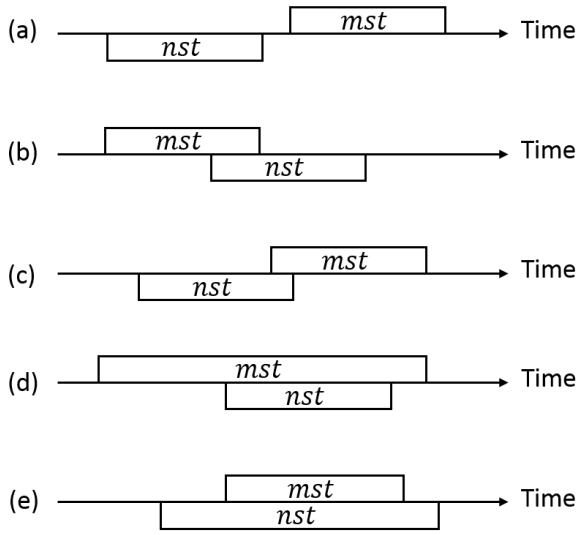


Figure 3.3: The scenarios of mst and nst for a single hazard. Note that there might be multiple sections of the nst, because for example a repeating in-behavior may enter the same must-not-start condition periodically and not start the intended behavior. But there can only be one section of mst, because by definition if the intended behavior does not start before the mst_T expires, there will be a hazard; but if the intended behavior starts within the mst_T, the whole when-to-start conditions are not applicable any more because the intended behavior already started, at which time it is the when-to-stop conditions that are applicable.

For a single hazard situation, there are five types of scenarios in terms of the relationship of mst_T and nst_T (Fig.3.3).

- Scenario (a): the mst and nst have no overlapping between each other. In this case, the intended output can start/not start accordingly to the defined time windows.
- Scenario (b): the nst overlaps the later part of the mst. In this case, the intended behavior must start before it enters the nst.
- Scenario (c): the nst overlaps the earlier part of the mst. In this case, the intended behavior must wait till the nst expires and start before the remaining mst_T expires.

- Scenario (d): the nst is included in the mst. In this case, the intended behavior must start within time interval *either before* the included nst *or after*. Note that by excluding the nst_T from the mst_T, we create multiple sections of the mst_T. But it is different because the relationship between the resulting sections are "OR", rather than the "AND" as for the nst sections.
- Scenario (e): the mst is included in the nst. The hazard is guaranteed in this case because there is no viable time stamp for the intended behavior to start before the mst_T expires.

In fact, we can succinctly summarize the scenarios above mathematically:

$$\text{If } mst\_T \neq \emptyset, mst\_T := \neg nst\_T \cap mst\_T$$

Finally, by definition if the intended behavior does not start before the mst_T expires, the hazard will happen, which will render any "safety constraints" after that time point meaningless. In other words, the time frame of interest (denoted as TF) is from the current time to the right bound of the mst_T. Therefore, we introduce the *frame operation*, denoted as $TF := frame(mst\_T)$ to acquire TF based on the mst_T, within which all the time windows mst_T, nst_T and cst_T are defined.

As a result, for the single hazard situation, the time windows can be integrated as below:

$$\text{If } mst\_T = \emptyset, cst\_T := \neg nst\_T;$$

$$\text{If } mst\_T \neq \emptyset, \begin{cases} TF := frame(mst\_T) \\ mst\_T := TF \cap \neg nst\_T \cap mst\_T \\ cst\_T := TF \cap \neg(mst\_T \cup nst\_T) \end{cases}$$

The "stop" conditions can be addressed in the same way as the "start" conditions.

If $msp\_T = \emptyset$, $csp\_T := \neg nsp\_T$;

If $msp\_T \neq \emptyset$, $\begin{cases} TF := frame(msp\_T) \\ msp\_T := TF \cup \neg nsp\_T \cap msp\_T \\ csp\_T := TF \cup \neg(msp\_T \cup nsp\_T) \end{cases}$.

3.4.2 The multiple sub-hazards situation

When there are multiple sub-hazards (assuming from 1 to N), all the safety constraints with respect to each sub-hazard must be integrated into the safety constraints for the hazard under study. The basic idea is similar with the single hazard situation, where the mst_T and nst_T are determined first and then the cst_T can be simply derived from them.

First, for the "must-not-start" condition, as long as there is one sub-hazard requiring "must-not-start", then the intended behavior must not start. Therefore, the nst as a whole is a disjunction of the nst_i of all the sub-hazards ($i = 1, ..., N$). Therefore, nst_T can be defined as below mathematically:

$$nst\_T = nst_1\_T \cup ... \cup nst_N\_T$$

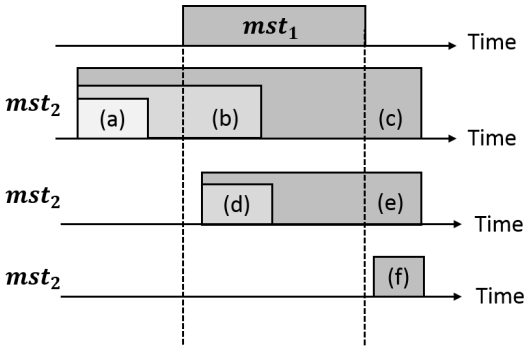


Figure 3.4: The scenarios of mst for the multiple sub-hazards situation.

Second, the integration of the "must-start" conditions is much more complicated. We assume two sub-hazards for the sake of explanation, and derive exhaustively six scenarios for the integration (Fig.3.4).

- Scenario (a): the mst_2 ends both before the mst_1 starts. In this case, the intended behavior must start within the mst_2_T before it expires and causes Sub-hazard 2. After the intended behavior starts, the original mst_1_T is not applicable any more and hence does not need to be satisfied.

- Scenario (b): the mst_2 starts before the mst_1 starts, and ends after the mst_1 starts but before it ends. In this case, the intended behavior must start within the mst_2_T, but with two different interpretations depending on which section it starts. If the intended behavior starts before mst_1 starts, it is similar to Scenario (a) where the start of the intended behavior renders mst_1 inapplicable. However, if the intended behavior starts after mst_1_T starts, it satisfies the "must-start" conditions for both sub-hazards at the same time.

- Scenario (c): the mst_2 starts before the mst_1 starts but ends after mst_1 ends. In this case, the intended behavior must start before the mst_1 ends. As for it starts before mst_1 starts or not, it is the same as Scenario (b).

- Scenario (d): the mst_2 starts after the mst_1 starts and ends before the mst_1 ends. In this case, the intended behavior must start after mst_1_T starts and before the mst_2_T ends. If it starts before the mst_2_T starts, it satisfies mst_1 and makes mst_2 inapplicable, which is acceptable; if it starts within mst_2_T, it satisfies both mst_1 and mst_2 at the same time.

- Scenario (e): the mst_2 starts after the mst_1 starts and ends after the mst_1 ends. In this case, the intended behavior must start within mst_1_T. If it starts before the mst_2_T starts, it satisfies mst_1 and makes mst_2 inapplicable, which is acceptable; if it starts after mst_2 starts and before mst_1 ends, it satisfies both mst_1 and mst_2 at the same time.

- Scenario (f): the mst_2 starts after mst_1 ends. In this case, the intended behavior must start within mst_1_T, which will satisfy mst_1 and render mst_2 inapplicable.

In summary, the intended behavior must start between the earliest left bound (denoted as $\underline{mst_i}$) and the earliest right bound (denoted as $\overline{mst_i}$) of all mst_i_T. If it starts at a time before some of the mst_i_T starts, it just renders them inapplicable, which is acceptable; if it starts at a time which is an element of the conjunction of all the mst_i_T, the intended behavior satisfies all the mst_i at the same time. Therefore, mathematically, the mst_T after the integration can be written as below:

$$mst\_T = [earliest(\underline{mst_i}), earliest(\overline{mst_i})], \text{ where } i = 1, ...N.$$

After the nst and mst are integrated from the sub-hazards, they can be further integrated in the same way as the single hazard situation. Furthermore, the "stop" conditions can be addressed in the same way. Finally, for the multiple sub-hazards situation, the time windows can be integrated as below:

$$\text{Let } \begin{cases} mst\_T = [earliest(\underline{mst_i}), earliest(\overline{mst_i})] \\ nst\_T = nst_1\_T \cup ... \cup nst_N\_T \\ msp\_T = [earliest(\underline{msp_i}), earliest(\overline{msp_i})] \\ nsp\_T = nsp_1\_T \cup ... \cup nsp_N\_T \end{cases}$$

$$\text{If } mst\_T = \emptyset, cst\_T = \neg nst\_T;$$

$$\text{If } mst\_T \neq \emptyset, \begin{cases} frame(mst\_T) \\ mst\_T := \neg nst\_T \cap mst\_T \\ cst\_T = \neg(mst\_T \cup nst\_T) \end{cases};$$

$$\text{If } msp\_T = \emptyset, csp\_T = \neg nsp\_T;$$

$$\text{If } msp\_T \neq \emptyset, \begin{cases} frame(msp\_T) \\ msp\_T := \neg nsp\_T \cap msp\_T \\ csp\_T = \neg(msp\_T \cup nsp\_T) \end{cases}.$$

3.5 Contribution

Thanks to the UCAs defined by STPA that are “provably complete”, Method 1 provides structured guidance to systematically identify hazardous scenarios. The resulting prescriptive constraints are solutions to address these scenarios and are also desired properties that the system must be designed for as a whole. Imagine if there is one scenario missing, the associated constraints which should have been designed for are also likely to be missed. As a result, the missing properties will never be examined explicitly by the verification program, which will eventually lead to “false assurance”.

Chapter 4

Method 2: Deriving the descriptive constraints for the controlled process.

The descriptive constraints for the control structure are manifested as the process model, which represents what the controlled process in the system (to be built) can possibly do under what condition. Although the process model varies from case to case, the methodology to define it can be derived by generalizing this process based on a general structure of the process model. This section introduces the general structure based on General Systems Theory and then explains how the process model can be defined based on the general structure.

4.1 Preliminary

First, We adopt from General Systems Theory [38] the general system representation as a general model structure of the process model.

$$(u(t), x(t), p) \xrightarrow{f} (\dot{x}, y(t))^T \tag{4.1}$$

- y is the set of output variables of the process under control.
- u is the set of control input variables, from the controller into the controlled process.
- x is the set of internal state variables.
- p is the set of parameters. A parameter can be varying among operations and even during a single operation, such as the weight of an airplane. It is different from the internal state variables in that it cannot be changed by the control input.
- f is not only the transformation between $(u(t), x(t), p)$ and $(\dot{x}, y(t))^T$, but also represents the mechanism that is going to be used directly from the physical laws or be implemented by the underlying devices.

Second, given the model structure above, the range of each variable/parameter has to be defined to complete the definition of the process model. They are the domain/co-domain of f as a mathematical function, see (4.2) below. Note that parameter may also change within a bound, and constant parameter is just a special case.

$$(u(t), x(t), p, \dot{x}, y(t)) \in (U, X, P, \dot{X}, Y) \tag{4.2}$$

In most analytical works, the definition of the process model stops here. They take (4.1) and (4.2) as the process model (or system dynamics) and start the design, for example designing a control algorithm, from here. Safety in these works is hence translated into a Constraint Satisfaction Problem [39], i.e., guaranteeing the constraints can never be violated under the respective design. However, this is insufficient. More precisely, satisfying (4.2) is *necessary* for safety, but not *sufficient*, because every mathematical model has assumptions. In the model-based world, we are good at computing properties based on the given constraints, but not so much when it comes to identifying the associated

assumptions, which is usually how a “surprise” happens and causes accidents. Before these assumptions are explicitly identified and specifically managed in the design, no safety assurance can be claimed from such design.

Method 2 is aimed to address this inadequacy by asking the following two questions:

- Where does the constraints in (4.2) come from?
- Are there assumptions that (4.1) and (4.2) are predicated on?

Finally, a process model is comprised of the following three parts. The rest of this section is to answer the two questions above.

Process model = {Model Structure, Constraints, Assumptions}

4.2 Constraining the controlled process

The two questions raised in the previous section are to make sure the process model will be a representation of the controlled process to be built. The process model cannot be valid on its own. It has to be consistent with how the controlled process will work in the real operation. Therefore, we study how the controlled process may be constrained in this section.

First, the controlled process is a desired mechanism(s) that accomplishes specific tasks. It operates on actual components. For the components that are out of the design scope (i.e., the designer has no control), we call them “environment”. Components within the design scope are the “system” to be built or further refined, or “human” to be trained. Therefore, the operation of the desired mechanism(s) can be constrained by the state of the environment and the states of the underlying components. For example, the airplane is not supposed

to take off in strong wind and heavy rain; the airplane is not supposed to land with the auto-brake when one of the hydraulic systems is down.

Furthermore, the mechanism can be viewed as an input-output transformation (4.1), which is obviously comprised of three parts: the input ports, the output ports and the transformation. The“input port” means the independent variable of the mathematical transformation and “output port” means the dependent variable of the mathematical transformation. The environment and the system/human may have constrains on all the three parts of the mechanism. For example, a human pilot can only exert a certain of force on the yoke to manipulate the control surface of an airplane in the early ages, a case of the underlying component constraining the input ports; a car has to run at a speed below the limit imposed by the government, a case of the environment constraining the output ports.

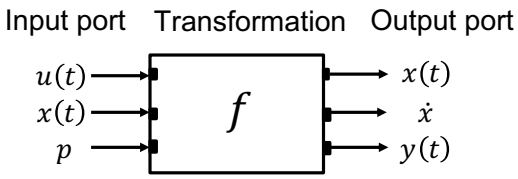


Figure 4.1: The model structure is abstracted into input ports, transformation and output ports. The reason that the internal state $x(t)$ is both input port and output port is that its value at the current step affects its value at the next step through $\dot{x}$. Therefore, it has to be treated differently depending on the situation.

Finally, the transformation may constrain the input ports and output ports. For example, when using Bernoulli’s law to compute the airspeed, the input airspeed cannot be too high, a case of the transformation constraining the input ports.

4.3 Constraining the process model

The process model is just a representation of the controlled process, and hence it is constrained in the same way as the controlled process. According to the

way that the controlled process is constrained, we develop the following map in Fig.4.2 to derive the constraints and the associated assumptions of the process model.

First, f represents concrete mechanisms (engineered or natural) of the actual process. It may only be able to process a finite set of the inputs and hence have constraints on the inputs (Arrow 1).

Second, the controlled process operates on actual components. For those that are out of the design scope (i.e., the designer has no control), we call them "environment". Components within the design scope are the "system" to be built or further refined, or "human" to be trained. For the environment, it may have a set that includes all the possible inputs (Arrow 2) and another set in which all the possible outputs must be included (Arrow 3). For the system/human, it is always subject to finite design/manufacture/natural capacities that may yield constraints on both inputs and outputs (Arrow 4).

Third, $(u, x, p, \dot{x}, y)$ is internally constrained by f. Therefore, the constraints on $(u, x, p, \dot{x}, y)$ also need to satisfy f mathematically (Arrow 5).

Finally, each constraint may have assumptions on the environment and the system/human for the constraint to be valid in the first place. Constraints and assumptions should always appear in pairs, and any isolated constraint or assumption must be justified.

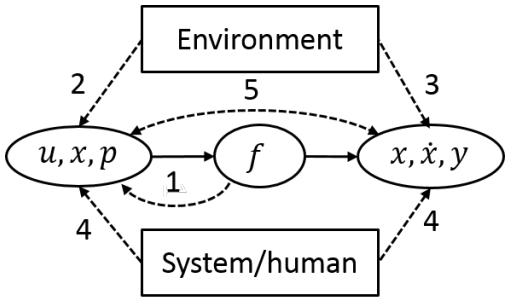


Figure 4.2: We identified eight arrows of constraints, a map to derive constraints for the process model. Each arrow reads as "the source constrains the end".

Therefore, all the **constraint-assumption pairs** need to be identified based on

Fig.4.2 to properly constrain the model of the controlled process. We are now explaining each of them.

- **Arrow 1: The transformation constrains the input ports**: the transformation is only valid when the values of the input ports are within a certain range. For example, the linear relationship between the AoA and the lift co-fficient requires the AoA to be within a certain range.
  - Constraint: The range of AoA to maintain the linear relationship.
  - Assumption: The range of AoA may vary over the wing configuration, for example icing condition. The wing configuration is hence the assumption.
- **Arrow 2: The environment constrains the input ports**: The environment (natural environment or other system) has a set that includes all the possible input values. For example, the airplane internal environment control system takes the air temperature as an input. Say, the aircraft is intended to fly no higher than 10,000 feet, then all the possible air temperatures from the ground level to 10,000 feet form a set constraints on the input of air temperature.
  - Constraint: The range of the possible environment temperature from the ground level to 10,000 feet.
  - Assumption: The temperature range may vary over the locations, such as close to equator vs. the north pole. The location is hence the assumption.
- **Arrow 3: The environment constrains the output ports**: The environment restricts the range of the allowed output values, such as a noise limit of the aircraft at certain altitude.
  - Constraint: The acceptable range of noise level by regulation.

  - Assumption: The acceptable range of noise may vary over the ground habitat, for example dessert or city. The ground habitat is hence the assumption.

- **Arrow 4: The system/human constrains the input/output ports**: The system/human can only accept a finite range of the input values and produce a finite range of output values due to limits of the capacities. For example, the electrical aircraft can fly with x KG payload at its maximal capacity at a maximal altitude Y.
  - Constraint: The acceptable range of payload X is a constraint on the input, and the maximal altitude Y is a constraint on the output.
  - Assumption: These constraints all depend on the battery level. Lower batter level means less X and Y. The battery level is hence the assumption.

- **Arrow 5: The transformation constrains the input and output ports**: Mathematically, the transformation is a mapping, which by definition is a constraint on the input ports and the output ports.
  - Constraint: The transformation itself, for example the Pascal's law.
  - Assumption: Pascal's law only works when the environment temperature is above the freezing point of the hydraulic fluid. The environment temperature is hence the assumption.

4.4 Contribution

Method 2 contributes to assuring the validity of the process model, which is fundamental to avoid "false assurance". Specifically, Method 2 develops a map to properly identify all the constraints and associated assumptions for a given model structure. As a result, Method 2 can help define a valid process model,

even when there is no existing system to model from, by systematically extracting all the relevant information of the elements involved in the model structure. Furthermore, when the process model is derived from an existing system through scientific modeling, Method 2 can still be helpful to cross-check the validity (including identifying assumptions) of the resulting process model. Scientific modeling starts from the existing system to find a model to represent the system. Method 2 is just the opposite, starting from a given model (structure) to find out what is implied about the real system. These two are dual processes that can be used to complement each other.

Caveat: Method 2 is a methodology to define the process model but cannot replace (aerospace/electrical/mechanical/etc.) engineer's scientific theory about the real process. In other words, Method 2 is not the theory, rather the engineering application of the theory. If there is something wrong with the engineer's understanding of the science, Method 2 cannot necessarily identify those mistakes.

Chapter 5

Method 3: Reference architecture for the controller design

5.1 Preliminary

5.1.1 The problem

So far, Method 1 has defined the prescriptive constraints:

$$\begin{cases} y(t) \in pc \\ st \in mst\_T \vee st \in cst\_T \wedge st \notin nst\_T \\ sp \in msp\_T \vee sp \in csp\_T \wedge sp \notin nsp\_T \end{cases}$$

Method 2 has defined the descriptive constraints:

$$(u(t), x(t), p) \xrightarrow{f} (\dot{x}, y(t))^T \text{ and } (u(t), x(t), p, \dot{x}, y(t)) \in (U, X, P, \dot{X}, Y)$$

Mathematically, the next task is to design a controller to issue the right control inputs u at the right time so that the prescriptive constraints can be satisfied subject to the descriptive constraints that the controlled process should satisfy for all time.

However, a *safe* controller in the actual system is more than a mathematical problem solver. It is a decision maker that constantly monitors the environment and the controlled process, watches out for hazards, reacts to changes and adjusts the decisions in real time. To design such a safe controller is to identify the safety-critical scenarios and design mechanisms into the controller to address all the safety-critical scenarios.

In this chapter, we will first explain in detail how the safety-critical scenarios can be identified, and then briefly propose a reference architecture for the controller to address all the safety-critical scenarios. Details of how the reference architecture is developed can be found in Appendix C.

5.1.2 Scope of the safe controller

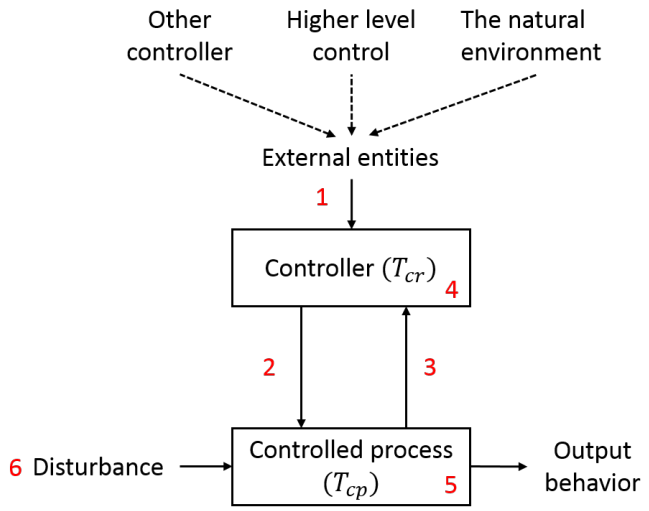


Figure 5.1: The scenarios that are not addressed by the reference architecture. But all of these scenarios are covered by MBSA.

Ideally, a *safe* controller must be able to handle all the safety-critical scenarios regardless of whether failure is involved. However, as explained in Chapter 1, we only consider the "hazard without failure" in this work, because failures have been extensively addressed in MBSA. Therefore, in this section we explain those failure-related safety-critical scenarios that are not addressed in this work.

Fig.5.1 is a description of the control structure and the external entities that interact with the control structure. There are 6 causal factors that are not addressed by the controller. In other words, the hazardous scenarios associated with these causal factors cannot be avoided or mitigated by the proposed reference architecture. However, all the 6 causal factors are usually covered by MBSA.

1. What information needs to be taken in from the external entities is covered by the proposed reference architecture, but we assume that the input information is correct and always received in time. It is possible in reality that the input information is incorrect from the source, is corrupted during transmission or outdated due to delay.

2. The correctness of the control action, both in terms of its value and timing is covered by the reference architecture, but we assume perfect actuator. It is possible that control action might not be successfully exerted on the controlled process or is exerted with a delay.

3. What information needs to be taken in from the controlled process is covered by the proposed reference architecture, but we assume perfect sensor at both value and timing aspect. It is possible that the information received is incorrect, corrupted, or delayed.

4. The actions that the controller must take are covered by the reference architecture, but we assume that the controller will perform all these actions correctly. It is possible that the controller fails, just like human controller make errors, software controller crashes, and mechanical controller brakes.

5. How to adjust the model of the controlled process according to the change of the implicit constraints is covered by the reference architecture, but we assume the controlled process actually follows the defined transforma-

tion. It is possible that the controlled process may not behave as defined by the process model due to real-time failures or implementation problems.

6. The disturbance can change the parameter and/or the states of the controlled process. The proposed reference architecture only consider the scenarios where the parameter and/or states have a “sudden jump” due to the disturbance. Other possible effect of the disturbance such as a value change for an extended time or a permanent structural change of the controlled process are not considered.

5.1.3 Functional architecture

Finally, the proposed reference architecture is functional architecture rather than physical architecture. In other words, the proposed architecture specifies what to do instead of who and how to do it, i.e. design intent vs. the implementation. The decision to stay at the functional level is because the goal of the reference architecture is to avoid hazardous scenarios due to inadequate functional design (assuming perfect implementation) rather the failure to fulfill the defined functions.

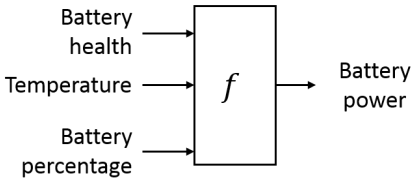


Figure 5.2: The action to calculate the the total electrical power that a battery contains.

For example, the controller of an eVTOL needs to calculate the total electrical power that a battery contains in real time. Assuming the battery power can be calculated with the the battery health, the current environment temperature and the current battery percentage level through a mapping f (Fig.5.2). The action

at the functional level only requires the latest information of the battery health, but does not specify where to get this information and how to get it. It is possible that the battery health information is input by the maintenance people at the last maintenance time; it is also possible the information provided to the flight manager and is input at the pre-flight phase; it is also possible that this information is received in real time from the battery health tracking system of the battery manufacture. Different ways of implementation are vulnerable to different failures. However, at the functional level, an adequate design only asks for the latest battery health without specifying where the information comes from and how it is received.

5.2 Three general tasks for a controller

In the most general sense, the controller as a decision maker needs to accomplish three tasks.

First, the prescriptive constraints are independent from the control structure, and hence can only be derived by observing the information from the environment, which implies that the first task of a controller is to decide the prescriptive constraints by observing the environment. We name the task “perceive prescriptive constarints” in the rest of the paper.

Second, the controller only affects the controlled process through the control actions, which implies that the last task of a controller is to decide the control action. We name the task “generate control action” hereafter.

Third, the prescriptive constraints are not always suitable to generate the control action. In fact, the controller must have at least one control reference to decide a control action, which implies that the control reference must be planned as a stepping stone from the prescriptive constraints to the control action. We name the task “plan control reference”.

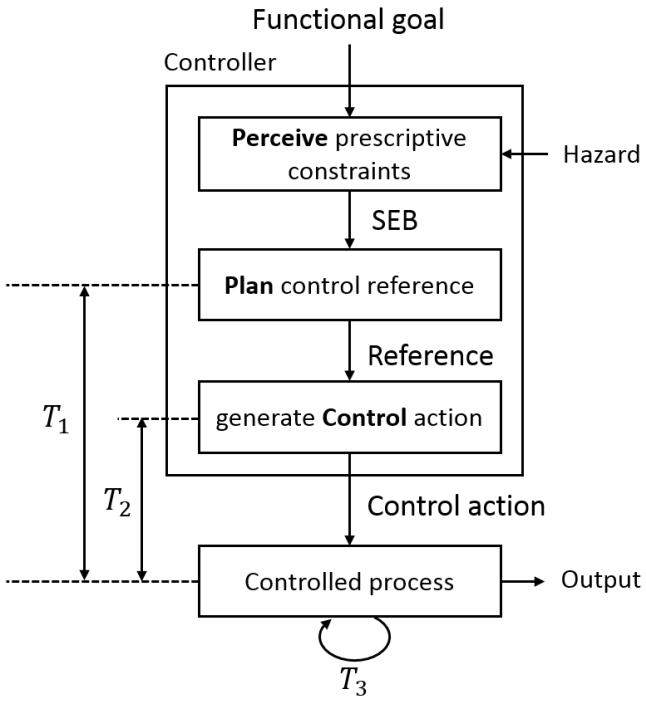


Figure 5.3: A controller must accomplish three general tasks, which forms a functional structure of the controller.

Combining the three tasks, we obtain a general functional structure of a controller in Fig.5.3, which is a generic characterization of all controllers. Another way to reason about this structure is from the bottom to top. Because the controlled process is manipulated by the control action, there must be a task in the controller to "generate the control action"; because at least one control reference must exist so that the control action can be generated, there must be task in the controller to "plan the control reference"; because the control reference always serves a control goal (i.e. the prescriptive constraints defined the output variable of the controlled process), there must be a task in the controller to generate such constraints by interpreting the higher level control command in a hierarchical structure or collecting information from the environment for its own goal, i.e. "perceive the prescriptive constraints".

Because this structure is "functional", it can be tailored in the specific design applications based on the implementation choices. For example, Task 1 can be unnecessary if the prescriptive constraints are directly passed from the

higher level, or they are constant and baked in the controller at the design time; Task 2 can be unnecessary if the prescriptive constraints happen to be fit as a control reference, for example the temperature of the power plant must cool down below x degree from y degree by z seconds; Task 3 can be unnecessary if the control reference is the same as the control action such as in the waypoint-based air traffic flow management [40]. Note that although the structure can be tailored in the specific application, it does not mean the structure is not general. Each task plays a unique role in the decision making process of the controller, and tailoring happens only because one set of actions can serve multiple roles but does not eliminates the roles in the first place.

Furthermore, the output of the task "perceive prescriptive constraints" is the "safety enforcing behavior (**SEB**)" instead of the "prescriptive constraints". As explained in Method 1, the prescriptive constraints are for the output behavior (i.e. the dynamic trajectory of the output variable of the controlled process), which contain the constraints on the start time, the stop time and the trajectory between the start time and the stop time. The SEB is a set of output behaviors whose start time, stop time and the trajectory in-between satisfy the respective constraints from the prescriptive constraints. As long as the eventual output behavior satisfy the SEB, the prescriptive constraints are automatically satisfied. Therefore, the SEB is just a proxy of the prescriptive constraints with a more direct connection with the output behavior. In addition, the SEB can be a refinement (i.e. a subset) of the prescriptive constraints. Such a refinement relationship provides flexibility to define the control goals in accordance with the uncertainty level of the controlled process. The more exact the control goals are, the more precise the prediction of the dynamic trajectory can be. On one hand, when there is no uncertainty in the controlled process, the SEB can be refined into an exact dynamic trajectory, which if achieved, will be exactly the same as the eventual output behavior. On the other hand, if the controlled process is subject to a high level of uncertainty, SEB can be the same as the prescriptive con-

straints (i.e. no refinement) in order to maximize the likelihood to find a control action. In summary, SEB is a flexible refinement of the prescriptive constraints, and hence fits in the general structure of Fig.5.3.

Finally, as shown in Fig.5.3, we define that the expected time duration of Task 2, Task 3 and the controlled process is T_1, the expected time duration of Task 3 and the controlled process is T_2, and the expected time delay of the controlled process is T_3. We will explain the safety-critical scenarios due to these time duration later.

5.3 Four types of safety-critical scenarios

Each of the three tasks that a controller has to make is a decision-making process. Regardless of the specific decision being made, each decision-making process may lead up to the six possible results in Fig.5.4.

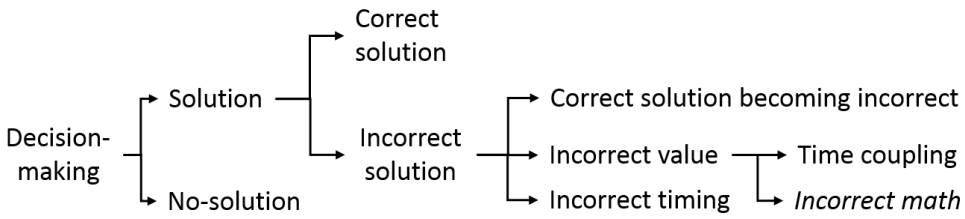


Figure 5.4: The six possible results that a decision-making process can lead up to.

First, a decision making process can either finds a solution or (1) no solution can be found. Second, if a solution is found, the found solution can either be (2) correct or incorrect. Third, for the incorrect solution, it can be caused by the value of the solution is incorrect, (3) the time that the solution is made is too late or (4) a solution that is previously correct becomes incorrect due to change. Finally, a solution with incorrect value be caused by (5) incorrect math or (6) the coupling relationship between the value aspect and the time aspect is

not considered.

Five of the six possible results are unsafe. Among the five unsafe results, “incorrect math” is addressed extensively by theoretical control or formal methods. Therefore, we *do not* consider this scenario in this work. The rest of four unsafe results are the four types of scenarios that must be addressed properly to avoid, which by definition are the **safety-critical scenarios** that a controller must be designed for.

Therefore, we consider the following four types of safety-critical scenarios for all the three tasks that a controller must accomplish.

- **No-solution**: It refers to the scenarios where no solution can be found.
- **Previously safe**: It refers to the scenarios where an already generated decision becomes unsafe due to change.
- **Unsafe timing**: It refers to the scenarios where the decision is made too late, so much so that the timing constraints in the prescriptive constraints cannot be met as the operation of the rest of the system also takes time.
- **Time coupling**: It refers to the scenarios where the result of a decision is affected by the time delay of the system, i.e. time and value are coupled, such as the phenomenon of Pilot-induced Oscillations.

5.4 Safety-critical scenarios of Task 1

5.4.1 The task

We summarize Task 1 in this subsection. The detailed derivation process can be found in Appendix C.

Task 1 is to generate the prescriptive constraints based on Method 1. The

prescriptive constraints of the output behavior $y(t)$ where $t \in [st, sp]$ can be defined in (5.1), and $(Y(t), ST, SP)$ are the final prescriptive constraints.

$$\begin{cases} y(t) \in Y(t) \subseteq pc, \text{ where } t \in [st, sp] \\ st \in ST \subseteq mst\_T \cup cst\_T \cap \neg nst\_T \\ sp \in SP \subseteq msp\_T \cup csp\_T \cap \neg nsp\_T \end{cases} \tag{5.1}$$

Before the desired output behavior starts, Task 1 takes in the functional goal FG and the prediction of the in-behavior, and observes a set of information from the environment E and the controlled process CP, to generate the prescriptive constraints and make sure the desired output behavior starts before mst_T expires.

$$(E, FG, In\text{-}B, CP) \rightarrow (pc, mst\_T, cst\_T, nst\_T) \rightarrow (ST, SP, Y(t)) \tag{5.2}$$

After the desired output behavior starts, the current prescriptive constraints might need an update. Besides E, FG and CP explained above, the prescriptive constraints to be updated, $Y'(t)$ and SP', may also be needed to calculate the deadline msp_T' to stop seeking the current prescriptive constraints and starts seeking the new prescriptive constraints $(ST, SP, Y(t))$.

$$(E, FG, CP, Y'(t), SP') \rightarrow (msp\_T', pc, msp\_T, csp\_T, nsp\_T) \rightarrow (ST, SP, Y(t)) \tag{5.3}$$

5.4.2 The safety-critical scenarios

Based on the description of Task 1, we examine the possible safety-critical scenarios that Task 1 must address.

No-solution. For (5.2), the entire must-start time window is included by the

must-not-start time window (i.e. $mst\_T \subseteq nst\_T$). For (5.3), the entire must-stop time window is included by the must-not-stop time window (i.e. $msp\_T \subseteq nsp\_T$).

Previously safe. This scenario can be caused by the change of the prescriptive constraints, or the deviated output behavior due to uncertainty. The former case can be caused by the changes of $\{E, FG, In\text{-}B, CP\}$ that make the current $(ST, SP, Y(t))$ invalid or unsafe. The latter case can be caused by the actual output behavior $y(t)$ does not start/ stop within ST/SP, or violates $Y(t)$ due to uncertainty.

Unsafe timing. This scenario is caused by the prescriptive constraints are decided too late. For (5.2), the prescriptive constraints must be decided before $\overline{mst\_T} - T_1$ so that there will be enough time for the execution of the rest of the system, so that the output behavior can eventually start before mst_T expires. For (5.3), the prescriptive constraints must be decided before $\overline{msp\_T'} - T_1$ for the execution of the rest of the system, so that the output behavior can eventually switch to the new prescriptive constraints before msp_T' expires.

Time coupling. This scenario is not applicable. Coupling between time and value is a phenomenon where the current decision is made based on a future condition that is predicted based on the current condition. The decision that this task makes is only predicting the safe condition in the future based on the current condition. In other words, coupling is decision made on prediction, where the correctness of the decision relies on the correctness of the prediction. Hence they are coupled. But Task 1 is only making a prediction and thus no coupling.

5.5 Safety-critical scenarios of Task 2

5.5.1 The task

Task 2 is to generate the control reference r so that the resulting output behavior $y(t)$ will satisfy the prescriptive constraints $Y(t)$ during $[st, sp]$. We briefly demonstrate how r can be derived in the subsection mathematically.

Based on the General Systems Theory, the process model and the control algorithm can be represented in (5.4).

$$\begin{cases} \text{Process model: } (u(t), x(t), p) \xrightarrow{f} (\dot{x}, y(t))^T \\ \text{Control algorithm: } (r, x(t), p) \xrightarrow{g} u(t) \end{cases} \tag{5.4}$$

Plug g into f, we get:

$$(r, x(t), p) \xrightarrow{(f,g)} (\dot{x}(t), y(t))^T \tag{5.5}$$

For an extended period of time $t \in T = [st, sp]$, (5.5) can be written as below:

$$(r, x(st), p) \xrightarrow{(f,g)} (x(T), y(T))^T \tag{5.6}$$

Focusing on the output, we can get (5.7) by inverse function $(f \cdot g)$.

$$(y(T), x(st), p) \xrightarrow{(f \cdot g)^{-1}} r \tag{5.7}$$

Note that (5.7) is not necessarily valid mathematically because $(f \cdot g)^{-1}$ might not exist. But it reveals the fact that the desired output behavior, f, g, the initial condition and the parameter all together determine the control reference. As a result, a more general depiction of such relationship is (5.8), where $Y(T) = \{y(t) \in Y(t) | T = [st, sp]\}$, and the arrow simply means a mapping (not neces-

sarily a function) exists between the two sides.

$$(Y(T), x(st), p, f, g) \rightarrow r \tag{5.8}$$

Finally, based on 5.6, once r is selected, the trajectory evolution of the controlled process is also determined given the initial condition $x(st)$ and the parameter p. As a result, functionally speaking, the r has to be picked not only to satisfy the prescriptive constraints $Y(t)$, but also the descriptive constraints $(U, X, P, \dot{X}, Y)$. Therefore, Task 2 is to generate the control reference by executing (5.8) subject to $(u(t), x(t), p, \dot{x}, y(t)) \in (U, X, P, \dot{X}, Y)$.

5.5.2 The safety-critical scenarios

Based on the description of Task 2, we examine the possible safety-critical scenarios that Task 2 must address.

No-solution. The controller cannot find a control reference to satisfy $Y(T)$ and the descriptive constraints $(U, X, P, \dot{X}, Y)$ at the same time, or $(f, U, X, P, \dot{X}, Y)$ is invalid because the implicit constraints are not satisfied.

Previously safe. The prescriptive and/or descriptive constraints can be violated due to the change of the constraints or the deviation of the actual controlled process from the planned trajectory evolution.

- The prescriptive constraints *are going to* be violated by the output behavior. This scenario can either be caused by the change of $Y(T)$ from Task 1, or the deviation of the trajectory evolution due to uncertainty.
- The descriptive constraints *are going to* be violated by the trajectory evolution. This scenario can either be caused by the change of $(f, U, X, P, \dot{X}, Y)$, or the deviation of the trajectory evolution due to uncertainty.

- The descriptive constraints *being* violated by the current states of the controlled process. This scenario can either be caused by the change of $(f, U, X, P, \dot{X}, Y)$, or the sudden change of the current states of the controlled process.
- The prescriptive constraints *being* violated by the current states of the controlled process. This scenario is addressed by the "previously safe" scenario of Task 1.

Unsafe timing. This scenario is caused by the control reference is issued too late, later than T_2 before it is supposed to take effect on the controlled process.

Time coupling. The value of the control reference can be affected by the time delay.

In the most general sense (Fig.5.5), the control reference starts to be generated at t_1 and the controlled process starts to seek the control reference at t_4. When such event sequence operates fast enough, the states observed at t_1 can be considered the same as the actual states at t_4. However, when the time delay between t_1 and t_4 is unneglectable, the initial states used to generate the control reference should not be the states observed at t_1, but rather the predicted states at t_4 based on the observation at t_1.

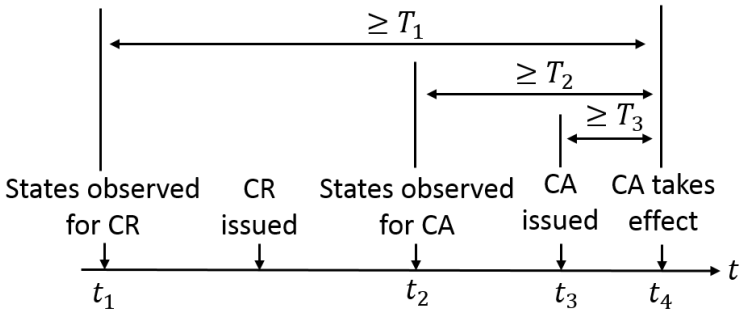


Figure 5.5: The event sequence from generating the control reference at t_1, to generating the control action at t_2, to the control action issued at t_3 and finally to the control action takes full effect at the controlled process at t_4. The time intervals of t_1, t_2 and t_3 with t_4 have to be greater than the time delay T_1, T_2 and T_3 defined in Fig.5.3. CR and CA stand for control reference and control action respectively.

Therefore, $x(st)$ in (5.8) in general should be predicted based on the observation of a state at least T_1 time before st, and updated when necessary. Therefore, the generation of r is coupled with the time delay through $x(st)$.

5.6 Safety-critical scenarios of Task 3

5.6.1 The task

Task 3 is the same as the "control algorithm" in theoretical control: generating the control actions based on the given control references (5.9).

$$\begin{aligned} (r, x(t), p) &\xrightarrow{g} u(t) \\ s.t.\ \{x(t), u(t)\}^T &\in \{X, U\}^T \end{aligned} \tag{5.9}$$

Note that the constraint $\{x(t), u(t)\}^T \in \{X, U\}^T$ is automatically satisfied because it has been addressed when generating r at Task 2, which is different from the regular theoretical control practice. The difference is because the standard formulation of a theoretical control problem is to generate the control actions that will satisfy both the prescriptive constraints and the descriptive constraints based on given control references, and (in some cases) to request new control references when no control action can be found. However, when given the initial condition, the control algorithm and the process model, whether a control action can be found has already been determined by the selection of the control references. In other words, a control action cannot be found is because of the wrong control reference.

In fact, $(u(t), x(t), p, \dot{x}, y(t)) \in (U, X, P, \dot{X}, Y)$ is a requirement that simply says that the trajectory evolution of the controlled process $(u(t), x(t), p, \dot{x}, y(t))$ cannot violate the descriptive constraints $(U, X, P, \dot{X}, Y)$. This requirement can be assigned to Task 2 or Task 3. In some applications, mathematical guarantees

can be made through reachability analysis at design time that the control action can always be found, i.e. $(u(t), x(t), p, \dot{x}, y(t)) \in (U, X, P, \dot{X}, Y)$ is always true, if the control references are selected from a specific set. In this case, such requirement is assigned to the design. Therefore, as a general task and under the principle of "making the right decision in the beginning", this requirement is assigned to Task 2.

5.6.2 The safety-critical scenario

Based on the description of Task 3, we examine the possible safety-critical scenarios that Task 3 must address.

No-solution. It has been explained in the previous section that as long as a control reference is generated, the control action can always be found. However, if the generated control reference needs to be updated, only the control reference after $t_c + t_4 - t_1$ can be updated (t_c is the current time) because of the time delay $t_4 - t_1$ in Fig.5.5. As a result, the control action within $[t_c, t_c + t_4 - t_1]$ must be updated in a way that the constraints (prescriptive and descriptive) can be satisfied at the same time. No-solution scenario is hence when such control action cannot be found.

Previously safe. For systems that operate at a slower pace, the control actions may be issued a certain time before being executed. As shown in Fig.5.5, the control action is issued over T_3 time delay before it is supposed to take effect at the controlled process. It is possible the issued control actions may lead to a violation of the constraints (prescriptive and descriptive) due to either the change of the constraints or the deviation of the controlled process from the predicted trajectory evolution.

Granted, some of the issued control actions will be updated because of the control references are updated due to this scenario. However, same as the no-

solution scenario, there is always a period of time in the immediate future that the control references will not be updated due to the time delay. Therefore, this scenario still needs to be addressed appropriately by Task 3.

Unsafe timing. This scenario is caused by the control action is issued too late, later than T_3 before it is supposed to take effect on the controlled process.

Time coupling. Similar to the time coupling scenario in Task 2, $x(t)$ in (5.9) in general should be predicted based on the observation of a state at least T_2 time before t, and updated when necessary. Therefore, the generation of u is coupled with the time delay through $x(t)$.

5.7 The reference architecture

The safety-critical scenarios identified above can already be used to examine whether an existing controller design is safe. However, we take one step further by defining "actions" within each task to address all the safety-critical scenarios. As a result, 33 main actions and 12 enabling action and the interactions among them are defined for the reference architecture. In this section, we give an overview of the reference architecture. Instead of explaining all the actions, we focus on the level of "theme" for each task. A theme is a group of actions that work together to achieve a common goal. In this way, we make a high level explanation about how the proposed reference architecture addresses the safety-critical scenarios for each task. The detailed process to derive the reference architecture can be found in Appendix C.

In general, three themes are defined for the reference architecture: the generate theme, the predict theme and the monitor theme. The generate theme addresses the no-solution and the unsafe timing scenario; the predict theme addresses the time coupling scenario; the monitor theme addresses the previously safe scenario. We now explain them for each task.

Task 1. Task 1 includes two themes: the generate theme and the monitor theme (Fig.5.6). The generate theme is to generate/update the prescriptive constraints, and to make sure the controller will response properly if no prescriptive constraint can be found (i.e. no-solution) or the prescriptive constraints are found too late (i.e. unsafe timing). The monitor theme is to make sure the generated prescriptive constraints stay safe, and request the generate theme to update the prescriptive constraints if they are not safe anymore (i.e. previously safe).

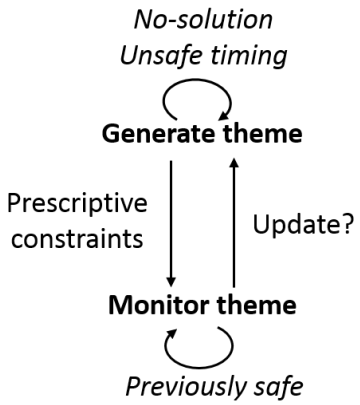


Figure 5.6: Task 1 is comprised of two themes: the generate theme and the monitor theme.

Task 2. Task 2 includes all three themes (Fig.5.7). The generate theme is to generate/update the control reference, and to make sure the controller will response properly if no control reference can be found (i.e. no-solution) or the control reference is found too late (i.e. unsafe timing). The predict theme is to predict the trajectory evolution of the controlled process based on the current control reference. On one hand, the trajectory evolution is sent to the generate theme for the $x(st)$ in (5.8) (i.e. time coupling). On the other hand, the trajectory evolution is sent to the monitor theme to make sure the trajectory evolution led by the current control reference will not violate the prescriptive constraints or the descriptive constraints (i.e. previously safe). Furthermore, the monitor theme also addresses the scenario when the current state of the controlled process violates the descriptive constraints.

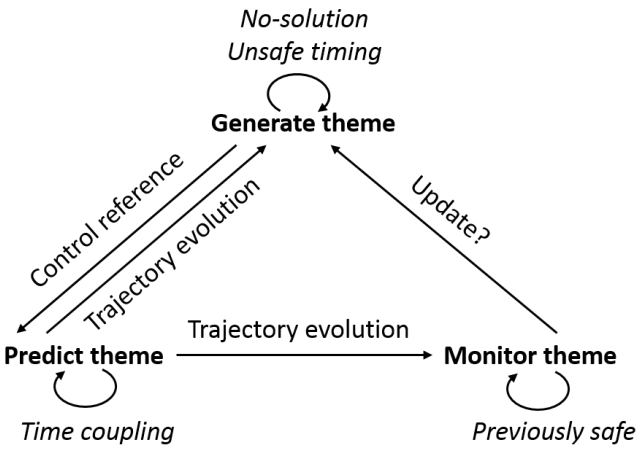


Figure 5.7: Task 2 is comprised of all three themes.

Task 3. Task 3 includes all three themes (Fig.5.8). The generate theme is to generate/update the control action, and to make sure the controller will response properly if no control action can be found (i.e. no-solution) or the control action is found too late (i.e. unsafe timing). The predict theme is to predict the trajectory evolution of the controlled process based on the control actions that are already generated. On one hand, the trajectory evolution is sent to the generate theme for the $x(t)$ in (5.9) (i.e. time coupling). On the other hand, the trajectory evolution is sent to the monitor theme to make sure the trajectory evolution led by the current control actions can achieve the current control references, and will not violate the prescriptive constraints or the descriptive constraints (i.e. previously safe).

As a result, the main actions that are included in each task can be summarized in (Table.5.1).

Table 5.1: The main actions that each theme includes. MA stands for main action.

	Generate theme	Predict theme	Monitor theme
Task 1	$MA_1, ..., MA_3, MA_9, ..., MA_{17}$	NA	$MA_4, ..., MA_{11}, MA_{18}, ..., MA_{20}$
Task 2	MA_{21}, MA_{22}	MA_{23}, MA_{24}	$MA_{25}, ..., MA_{29}$
Task 3	MA_{30}, MA_{31}	MA_{32}	MA_{33}

Finally, the reference architecture for the controller is defined in Fig.5.9 to

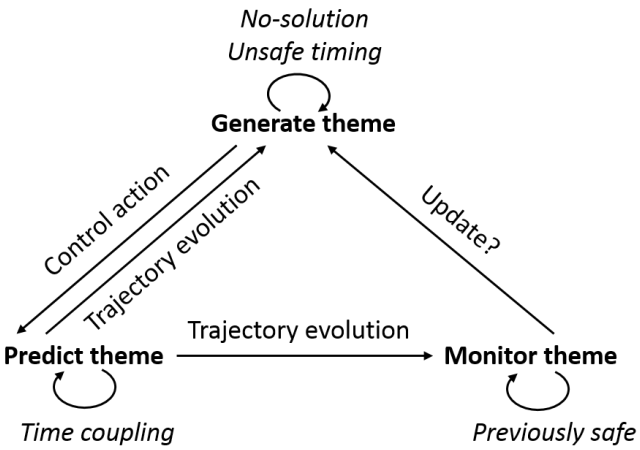


Figure 5.8: Task 3 is comprised of all three themes.

address all the identified safety-critical scenarios. The interactions of the main actions with other main actions, the enabling actions, the controlled process and the external entities are depicted in the colored cells. Particularly, the interaction of the controller with the controlled process and the external entities are presented in Fig.5.10. The reference architecture works like a *class* (as in Object Oriented Programming). In the specific design applications, engineers only need to instantiate the reference architecture and tailor it for its own problem, which will yield a controller that has all the safety-critical scenarios automatically addressed. The full readable reference architecture in N^2 diagram can be found in Appendix D.

5.8 Contribution

Method 3 first identifies the safety-critical scenarios (without failure at the component level) that a controller must address, and then provides a reference architecture that addresses the safety-critical scenarios in the beginning. Any controller that is designed according to the reference architecture will have these safety-critical scenarios automatically addressed, resulting in a safe-by-construction controller.

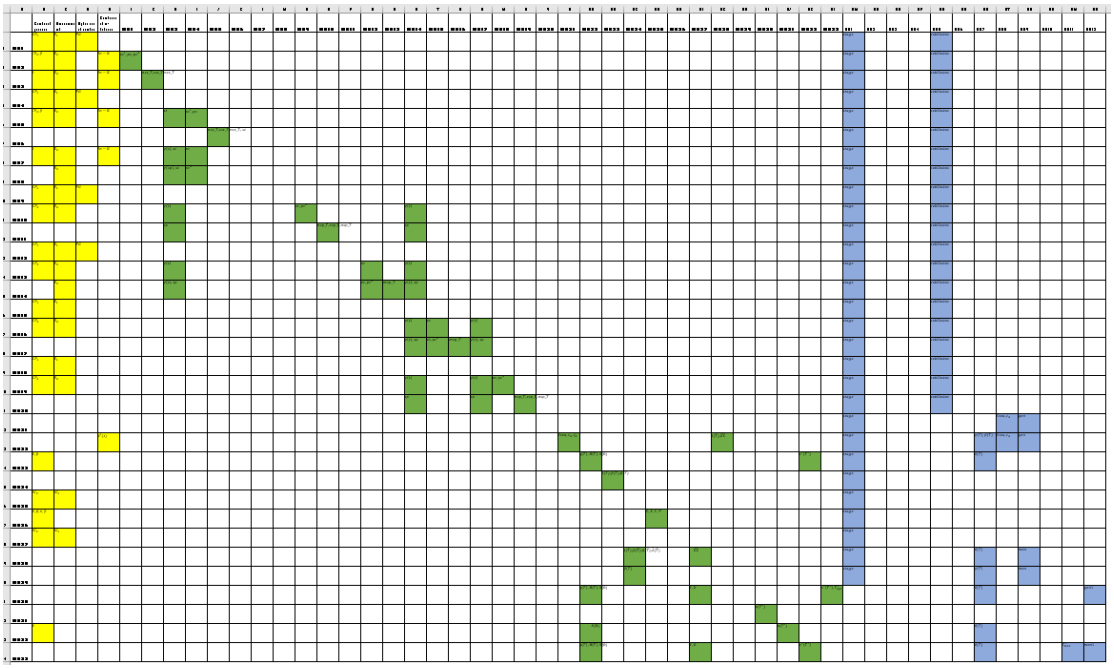


Figure 5.9: The reference architecture in an N^2 diagram. The yellow, green and blue cells are the interactions between the main actions and the environment, within the main actions, and between the main actions and the enabling actions respectively.

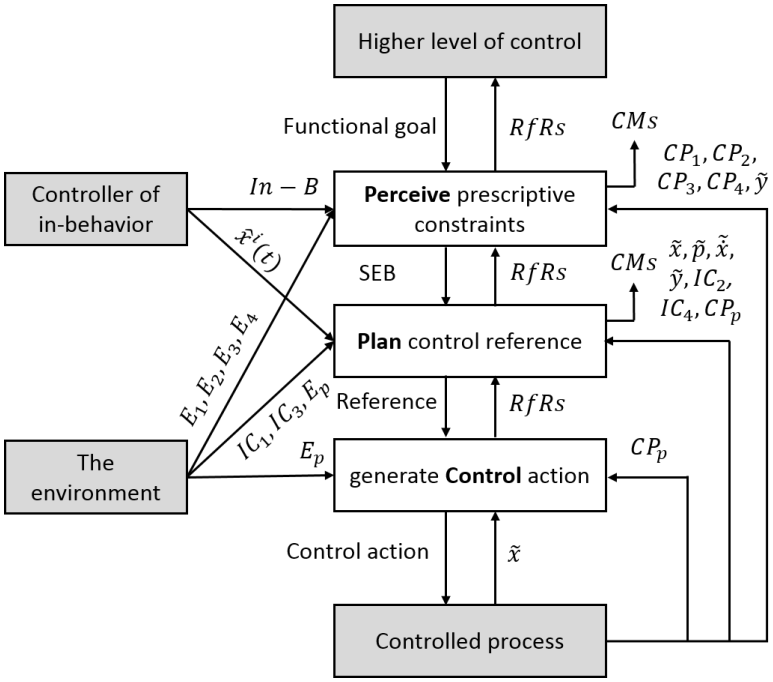


Figure 5.10: The interaction of the controller with the controlled process and the external entities. The three white boxes are the three tasks within a controller, and all the grey boxes are the entities external to the controller. The detailed actions within the controller and the interactions between them can be found in Appendix D.

Chapter 6

A case study on Urban Air Mobility

We apply all the three methods to an early design for Urban Air Mobility (UAM). Specifically, we focus the hazard of “inadequate altitude” associated with the functional goal of “descent”. Through this case study, we show how the three methods can help design a system that can avoid the hazard and achieve the functional goal at the same time.

6.1 The problem setting

A UAM typical flight includes five phases (Fig.6.1): take-off, climb to enroute, enroute, descent and landing. The case study focuses on the **descent** phase, specifically to design a guidance system to direct the UAM vehicle to descend from the cruise trajectory to the hover point above a landing pad, which is in fact the functional goal to achieve. Furthermore, we select the hazard of **inadequate altitude** as an example in this case study to demonstrate how the UAM is designed to avoid hazards. Although there might be other hazards, the same process that is demonstrated in the case study can also be applied to those other hazards.

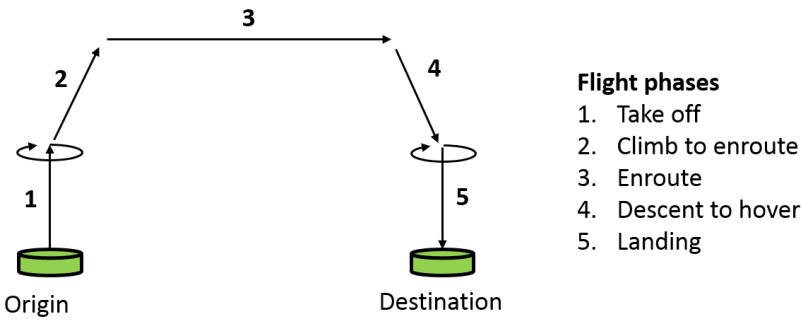


Figure 6.1: The case study focuses on the descent phase of a typical UAM flight.

Specifically, as shown in Fig.6.2, the functional goal under study is to guide a UAM vehicle to descend and arrive at Point B at rta from the cruise phase. The hazard of inadequate altitude is refined into four sub-hazards: **inadequate altitude with respect to weather/traffic/terrain/airspace**.

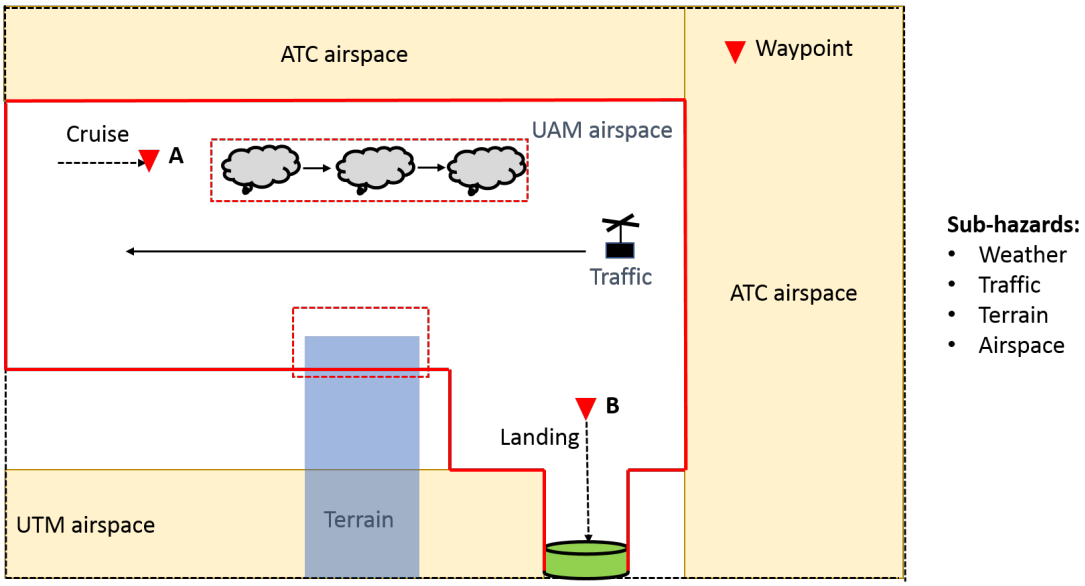


Figure 6.2: The UAM case study of avoiding the hazard of *inadequate altitude*

Finally, **we assume** that the descent *can always stop immediately once Point B is arrived at* rta. In this way, the out-behavior will not be considered when deciding the constraints on the stop time of the descent. Because deriving the constraints on the stop time is similar to deriving the constraints on the start time, this assumption as a result will reduce the size of this case study without sacrificing too much coverage of the methodology.

6.2 Method 1: deriving the prescriptive constraints

In this section, we show how the individual prescriptive constraints are identified with respect the functional goal and the (sub-)hazard.

The in-behavior, the intend behavior and the out-behavior are respectively **cruise**, **descent** and **go-around** when descent becomes unsafe or infeasible.

6.2.1 The functional constraints

With respect to the functional goal, the functional constraints comes from the environment and the feasibility of the control structure. **We assume**, rta might change after initially issued, but Point B does not change once issued.

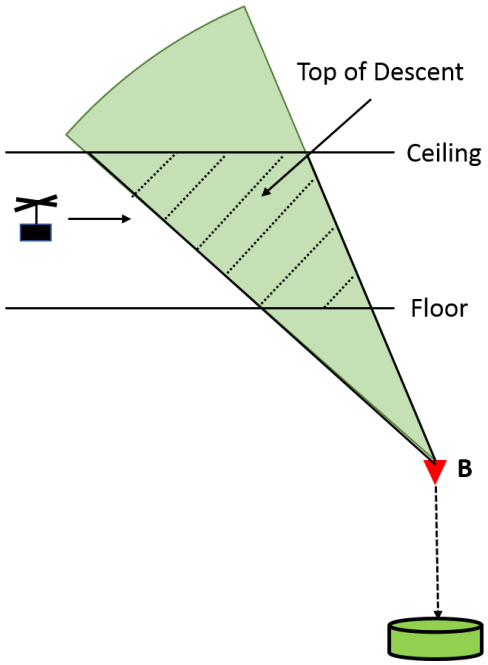


Figure 6.3: Deriving the prescriptive constraints w.r.t. the functional goal.

From the *environment*, we identify three requirements as below , specifically the “procedure” in this case study. Of note, there might be more requirements from the environment, but identifying them rely on the domain knowledge and does not affect the demonstration of the method. Therefore, we only use the

following three requirements for example.

- The top of descent can only be a point between the ceiling and the floor of Fig.6.3.
- The vehicle can [1] only start the descent if it is cleared.
- The vehicle has to stop the descent if the clearance is revoked.

From the perspective of the *feasibility* of the functional goal, the following two performance constraints are identified.

- The descent trajectory must be bounded by the green cone of Fig.6.3 formed by the lower upper bound and the upper bound of descent angle of the vehicle.
- The vehicle must arrive at Point B at rta, neither before or after.

All of three requirements from the environment and two performance constraints from the perspective of feasibility are then translated into the prescriptive constraints below. Note that multiple prescriptive constraints may be derived from one requirement due to the translation from the natural languages to pure logic.

1. Performance constraint (pc_0^1): The descent trajectory must be bounded within the green cone before Point B is reached.

   Assumption: The determination of the green cone depends on attributes from both the environment and the vehicle. The shape of the green cone, for example, may depends on the headwind (w) and the weight of the payload ($weight$).

2. Performance constraint (pc_0^2): The vehicle must arrive at Point B at rta.

[1] The "can" actually means "must-not-start". The "can" condition is neither the "must" nor "must-not".

3. Must-start condition (mst_0^1): The vehicle must start the descent once it level flies into the shaded area before it exits the shaded area.

    - The associated time window $mst\_T_0^1$ can be calculated based on the predicted vehicle speed and the length of the horizontal cross section.

    Assumption: The ceiling and the floor are decided by the procedure for descent. They can be changed by factors such as the cloud and visibility, which are the assumption for this constraint.

4. Must-start condition (mst_0^2): The vehicle must start the descent when $t_c \in [rta - \delta_1, rta - \delta_2]$.

    - The associated time window $mst\_T_0^2$ is $[max(t_c, rta - \delta_1), rta - \delta_2]$.

    Assumption: This constraint is based on the assumption that the vehicle is in its nominal operation mode; δ_1 and δ_2 can be adjusted based on the headwind.

5. Must-not-start condition (nst_0^1): The vehicle must not start the descent if it is outside the shaded area.

    - The associated time window $nst\_T_0^1$ is from the current time t_c to the time that the vehicle is projected to enter the shaded area, or all the time after the time the vehicle is projected to leave the shaded area.

6. Must-not-start condition (nst_0^2): The vehicle must not start the descent if $t_c \notin [rta - \delta_1, rta - \delta_2]$.

    - The associated time window:

$$nst\_T_0^2 = \begin{cases} [t_c, rta - \delta_1], \text{when } t_c < rta - \delta_1 \\ [rta - \delta_2, +\infty], \text{when } t_c \in [rta - \delta_1, rta - \delta_2] \\ [t_c, +\infty], \text{when } t_c > rta - \delta_2. \end{cases}$$

7. Must-not-start condition (nst_0^3): The vehicle must not start the descent if it is not cleared.

8. Can-start condition (cst_0): $False$, because $cst_0 = \neg(mst_0^1 \vee nst_0^1 \vee mst_0^2 \vee nst_0^2 \vee nst_0^3)$ is always at any time before the descent start. No associated time window is defined because $cst_0 = false$.

9. Must-stop condition (msp_0^1): The descent must stop if the planned trajectory is going to exit the green cone before rta (for example because of the change of the green cone).

    - The associated time window $msp\_T_0^1$ is from the current time t_c to the time that the vehicle is projected to exit the green cone. Note that the determination of the time window depends on what is considered safe (or feasible in this case). Therefore, it may vary based on the preference on the safety margin.

10. Must-stop condition (msp_0^2): The descent must stop if the planned trajectory cannot guide the vehicle to arrive at Point B on time due to the change of rta.

    - The associated time window $msp\_T_0^2$ is from the current time t_c to the current stop time or the new rta, whichever is earlier. Note that if the current SEB stops too late, the controller may not be able to find a new SEB, which is an example that satisfying constraints in this task does not necessarily leads to feasibility.

11. Must-stop condition (msp_0^3): The descent must stop *immediately* if the clearance is revoked before rta.

12. Must-not-stop condition (nsp_0): The current time is before rta and the vehicle is not at Point B. Intuitively, this means the descent is not finished and it is not impossible to reach Point B on time.

    - The associated time window $nsp\_T_0 = [t_c, rta]$.

13. Can-stop condition (csp_0): The descent can stop if the vehicle arrives at Point B at rta.

- The associated time window $csp\_T_0 = [rta, +\infty]$.

6.2.2 Inadequate altitude with respect to the traffic

The respective performance constraints are defined in below (Fig.6.4):

- pc_1^i: During cruise, the minimal clearance with other vehicle is x_1 radius.
- pc_1: During descent, and the minimal clearance with other vehicle is y_1 radius.

Assumption: The minimal clearance is for example based on the vehicle type of the traffic and the battery level of the vehicle.

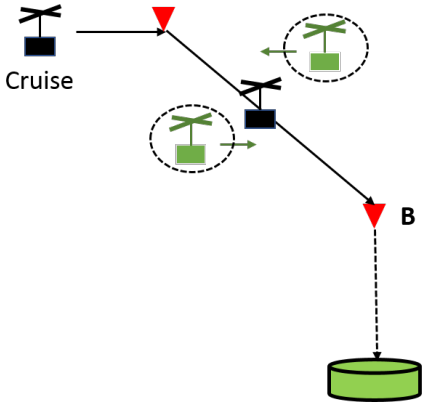


Figure 6.4: Deriving the prescriptive constraints w.r.t. the traffic.

Accordingly, the prescriptive constraints with respect to the traffic can be defined as following:

1. The must-start condition (mst_1): The descent must start when the traffic is projected to enter the x_1 feet radius of the vehicle. Note that if the look-ahead time is defined, then mst_1 is $true$ only when such entrance is projected to happen from the current time less than the look-ahead time.
    - The associated time window mst_T_1 is from the current time t_c to the

time that the traffic is projected to enter the x_1 feet radius of the vehicle.

2. The must-not-start condition (nst_1): The descent must not start when the traffic is within y_1 radius of the vehicle.
    - The associated time window nst_T_1 is from the current time t_c to the time that the traffic is projected to exit the y_1 feet radius of the vehicle.
3. The can-start condition (cst_1): $cst_1 = \neg(mst_1 \vee nst_1)$ and $cst\_T_1 = \neg(mst\_T_1 \cup nst\_T_1)$.
4. The must-stop condition (msp_1^1): The descent must stop when the traffic is projected to enter the y_1 feet radius of the vehicle.
    - The associated time window $msp\_T_1^1$ is from the current time t_c to the time that the traffic is projected to enter the y_1 feet radius of the vehicle.
5. The can-stop condition (csp_1): $csp_1 = \neg msp_1$ and $csp\_T_1 = \neg msp\_T_1$.

6.2.3 Inadequate altitude with respect to the weather

The performance constraint for the cruise (pc_2^i) and the descent (pc_2) is the same, which is the vehicle must stay out of the area that is impacted by the weather.

Accordingly, the prescriptive constraints with respect to the weather (Fig.6.5) can be defined as following:

1. The must-start condition (mst_2): The predicted trajectory of the level flight is going to cross a weather impacted area.
    - The associated time window mst_T_2 is from the current time t_c to the time that the traffic is projected to enter the weather impacted area.

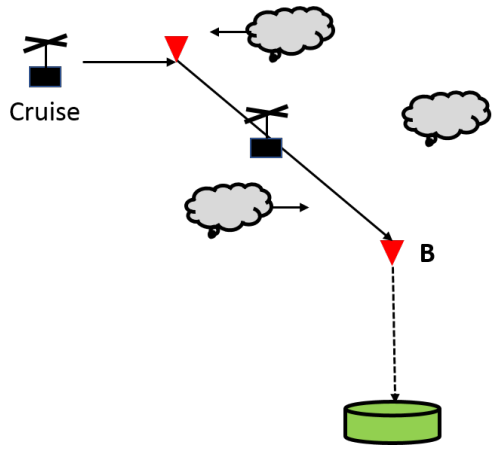


Figure 6.5: Deriving the prescriptive constraints w.r.t. the weather.

2. The must-not-start condition (nst_2): The vehicle is in the weather impacted area.

   - The associated time window nst_T_2 is from the current time t_c to the time that the vehicle is projected to exit the weather area.

3. The can-start condition (cst_2): $cst_2 = \neg(mst_2 \vee nst_2)$ and $cst\_T_2 = \neg(mst\_T_2 \cup nst\_T_2)$. For example, the can-start condition can be when there is no weather in the way of the predicted level flight, or the vehicle has exited the weather area.

4. The must-stop condition (msp_2^1): The planned trajectory must stop when the vehicle is projected to enter the weather area (because the weather can form unexpectedly).

   - The associated time window $msp\_T_2^1$ is from the current time t_c to the time that the vehicle is projected to enter the weather area.

5. The can-stop condition (csp_2): $csp_2 = \neg msp_2$ and $csp\_T_2 = \neg msp\_T_2$.

6.2.4 Inadequate altitude with respect to the terrain

The hazard with respect to the terrain is the altitude of the vehicle is less than a minimal distance with the terrain (Fig.6.4). The corresponding performance

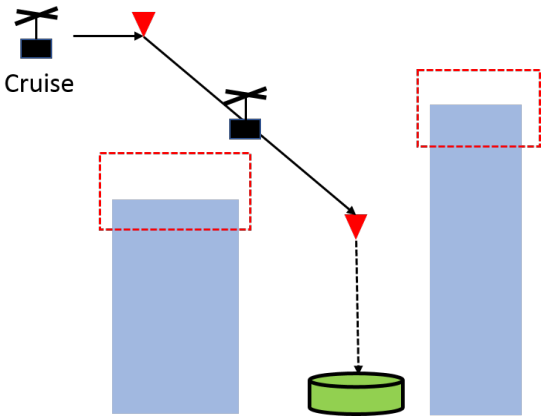


Figure 6.6: Deriving the prescriptive constraints w.r.t. the terrain.

constraints with respect to the cruise and the descent are $pc_3^i = x_3$ and $pc_3 = y_3$ respectively.

Assumption: The minimal distance from the terrain is for example based on the ground habitat, such as residence, power grid and natural area.

Accordingly, the prescriptive constraints with respect to the terrain (Fig.6.6) can be defined as following:

1. The must-start condition (mst_3): There is terrain that is higher or less than x_3 lower than the the predicted trajectory of the level flight.
    - The associated time window mst_T_3 is from the current time t_c to the time that the vehicle is projected to enter the conflicting terrain area.
2. The must-not-start condition (nst_3): The vehicle is less than y_3 above the terrain.
    - The associated time window nst_T_3 is from the current time t_c to the time that the vehicle is projected to be more than y_3 above the terrain.
3. The can-start condition (cst_3): $cst_3 = \neg(mst_3 \vee nst_3)$ and $cst\_T_3 = \neg(mst\_T_3 \cup nst\_T_3)$. For example, the can-start condition can be when the level flight altitude is way higher than x_3 and y_3 above the terrain.
4. The can-stop condition (csp_3): $csp_3 = \neg msp_3$ and $csp\_T_3 = \neg msp\_T_3$.

Note that we assume the terrain does not change in real time. Therefore, the planned trajectory can guarantee the pc_3 is always satisfied. Although the real output behavior can still deviate from the planned one, it does not affect the definition of the prescriptive constraints.

6.2.5 Inadequate altitude with respect to the airspace

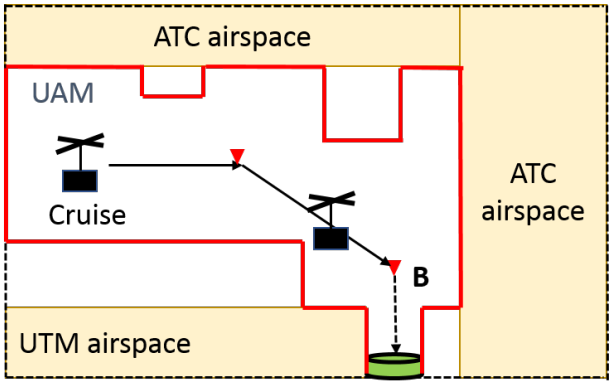


Figure 6.7: Deriving the prescriptive constraints w.r.t. the airspace.

The hazard with respect to the terrain is the altitude of the vehicle is less than a minimal distance with the ceiling of the airspace (Fig.6.7). The corresponding performance constraints with respect to the cruise and the descent are $pc_4^i = x_4$ and $pc_4 = y_4$ respectively.

Assumption: The minimal distance from the ceiling is for example based on the category of the adjacent airspace.

Accordingly, the prescriptive constraints with respect to the terrain (Fig.6.7) can be defined as following:

1. The must-start condition (mst_4): The current level flight altitude is above or less than x_4 below the airspace ceiling some points at the projected trajectory.

    - The associated time window mst_T_4 is from the current time t_c to the time that the vehicle is projected to be higher than or less than x_4

below the airspace.

2. The must-not-start condition (nst_4): The vehicle is above or less than y_4 below the ceiling.

    - The associated time window nst_T_3 is from the current time t_c to the time that the vehicle is projected to be more than y_4 below the ceiling.

3. The can-start condition (cst_4): $cst_4 = \neg(mst_4 \vee nst_4)$ and $cst\_T_4 = \neg(mst\_T_4 \cup nst\_T_4)$.

4. The must-stop condition (msp_4^1): Some section of the planned trajectory is higher or less than y_4 lower than the ceiling.

    - The associated time window $nst\_T_4^1$ is from the current time t_c to the time that the vehicle is projected to be higher than or less than y_4 below the ceiling.

5. The can-stop condition (csp_4): $csp_4 = \neg msp_4$ and $csp\_T_4 = \neg msp\_T_4$.

6.3 Method 2: deriving the descriptive constraints

6.3.1 The model structure

The descent process rather than the vehicle is the process under control (Fig.6.8). The controller is to eventually manipulate two output variables of the descent process: the real descent angle γ and the real longitudinal velocity v. The control input of the process is the instructed descent angle (Des) and the instructed longitudinal velocity (Vel). The head wind, denoted as w, is blowing horizontally right to left. Mathematically, the process model can be written as below:

$$\begin{cases} \gamma = Des \\ v = Vel - w \cdot cos\gamma \\ s(t) = s_0 + vt \end{cases}$$

where

- Control input: $(Vel, Des)^T$.
- Parameter: w.
- System states: $(\gamma, s)^T$, where $s = (x, h)$.
- Derivatives: v is the derivative of $s(t)$.
- Output variable: $s(t)$.

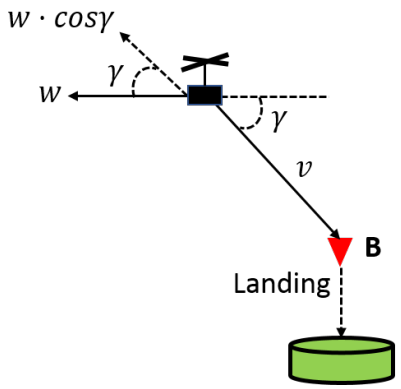


Figure 6.8: The process model for the descent.

Graphically, the input ports and the output ports can be represented in Fig.6.9.

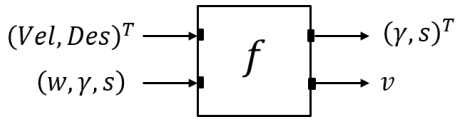


Figure 6.9: The structure of the process model.

6.3.2 Deriving the constraint-assumption pair

Each constraint-assumption is derived for a reason. We call that reason "**concern**" in this work. "**Constraint**" is defined to address the concern and can only successfully address the concern when the "**assumptions**" are satisfied in the actual operation.

First, because the highest order derivatives of the system states must (be considered) change instantaneously, the process model above can only be applied in the scenarios where the longitudinal velocity v and the descent angle γ can be considered change instantaneously. Various constraints can be derived from this concern, such as the congestion level of the airspace, the type of the vehicle, the expected maneuver, the terrain, the weather etc. The derivation of the constraints highly relies on the domain knowledge and may vary from case to case. For example:

- Concern 1: The longitudinal velocity must be considered to change instantaneously.
  - Constraint: the gap between the instructed velocity and the current velocity must be less than 20 knots, i.e. $|Vel - v| \leq 20$ knots.
  - Assumption: The traffic congestion level must be low (however the congestion level is measured), i.e. $congestion = low$ so that the airspace is sparse enough that such approximation is proper.
- Concern 2: The descent angle must be considered to change instantaneously.
  - Constraint: the gap between the instructed descent angle and the current descent angle must be less than $15°$, i.e. $|Des - \gamma| \leq 15°$.
  - Assumption: The headwind must be less than 50 knots, i.e. $w < 50$ knots as stronger headwind takes longer time to maneuver.

Second, we derive the constraint-assumption pair based on the map provided by Method 2.

Arrow 1: The transformation constrains the input ports.

- Concern 3: $\{v, \gamma, w\}^T$ must be within a specific envelop, so that the vehicle can maintain a *constant* descent angle and a *constant* longitudinal speed.
  - Constraint: The envelop can characterized by $h(v, \gamma, w) = 0$ where $v < 200$ knots, $\gamma \in [35°, 55°]$ and $w \in [25, 45]$ knots
  - Assumption: The crosswind must be less than 30 knots, i.e. $cw \in [0, 30]$ knots, so that the envelop is valid.

Arrow 2: The environment constrains the input ports. Environment here means the elements that are external to the controlled process. All three variables in the input ports may have concerns with respect to the environment. For example:

- Concern 4: The air traffic center may prescribe that instructed descent velocity Vel cannot be too high due to the limitation of the surveillance infrastructure.
  - Constraint: The instructed velocity must be less than 200 knots, i.e. $Vel \leq 200$ knots.
  - Assumption: $Vel \leq 200$ may be applicable to a specific weather condition, for example sunny days vs. days with heavy rain. Therefore, the weather condition may be an assumption of the process model here, and hence $Vel \leq 200$ must be updated accordingly if necessary.
- Concern 5: The descent procedure may prescribe that the descent angle γ must be bounded by a specific range to maintain a "flow" pattern of the descent traffic.

- Constraint: The descent angle Des must be bounded between $30°$ and $60°$, i.e. $Des \in [30°, 60°]$.
- Assumption: $Des \in [30°, 60°]$ is procedure-specific. If a new procedure is applied for example due to emergency, the bound for Des must be updated accordingly.

- Concern 6 The weather service provider prescribe that the headwind around the vertiport cannot be too high in order to guarantee the quality of their service for the vertiport operation (not necessarily for the descent process).
  - Constraint: The headwind must be less than 50 knots i.e. $w < 50$ knots.
  - Assumption: $w < 50$ knots may only reflect the normal capability of the weather service provider. If the provider is in a degraded service mode due to some of its internal issues (e.g. outage), $w < 50$ must be updated accordingly.

Arrow 3: The environment constrains the output ports. All three variables in the output ports may have concerns with respect to the environment. For example:

- Concern 7: The descent procedure may prescribe that real descent velocity v cannot be too low in order to maintain a minimal traffic throughput.
  - Constraint: The real velocity must be greater than 50 knots, i.e. $v > 50$ knots.
  - Assumption: $v > 50$ may based on an estimation of the regular traffic. If the traffic load is extraordinarily high due to for example temporary airspace restriction, the minimal velocity may be increased to accommodate the heavier traffic, and hence this constraint of the process model must be updated accordingly.

- Concern 8: The vertiport operation may prescribe that the descent angle γ cannot be too small in order to avoid interference between operation of the adjacent landing pad.
  - Constraint: The descent angle γ must be greater than $45°$, i.e. $\gamma > 45°$.
  - Assumption: $\gamma > 45°$ is conditioned on a specific layout of the vertiport. If new landing pads are added or back-up landing pads are activated, the range of γ must change accordingly.
- Concern 9: The vehicle must maintain a minimal vertical distance from the ground to minimize the disturbance to the public.
  - Constraint: The vertical distance from the ground elevation (denoted as GE) must be greater than H, i.e. $h > H + GE$.
  - Assumption: The minimal distance H is affected by the usage of the ground area, such as residence, natural habitat and high-voltage power line. Therefore, H must be updated accordingly in real time if necessary.

Arrow 4: The system/human constrains the input ports and the output ports The device/human must be able to accept all the values within the range of the input ports. Vel and Des are the instruction from the controller, which depending on the specific implementation can be vocal or digital. However, they are just numbers, and hence no constraints are derived for them. Only the headwind w is constrained.

- Concern 10: The headwind is observed by the on-board sensor, and hence is constrained by the capacity of the sensor.
  - Constraint: The headwind must be less than 60 knots in order to guarantee the accuracy of the on-board sensor, i.e. $w < 60$ knots.

- Assumption: The sensor data is only accurate for example when the environment temperature is between $[-10, 50]$°C. Therefore, the assumption of the process model for $w < 60$ is $tem \in [-10, 50]$°C.

- Concern 11: The vehicle has a capacity limit for the descent angle.
  - Constraint: The real descent angle is bounded between 30° and 75°, i.e. $\gamma \in [30^\circ, 75^\circ]$
  - Assumption: The range of the descent angle may vary over many factors. For example, the vehicle that transport freight can have a steeper descent angle than those commuting human passengers; the heavier of the payload is, the narrower of the descent angle bound; the stronger the headwind is, the less steeper the maximal descent angle is. There might be more factors, but as an example we define the assumptions of the process model for $\gamma \in [30^\circ, 75^\circ]$ is the type of payload is *passenger commute*, the payload weight is between $weight \in [1000, 1500]$ lbs and the headwind is less than 60 knots.

- Concern 12: The vehicle has a capacity limit for the descend altitude.
  - Constraint: The allowable descent altitdue is between 2500 feet to 1000 feet, i.e. $h \in [1000, 2500]$ feet ASL.
  - Assumption: $h \in [1000, 2500]$ feet is measured in ASL. Such limit varies over the local ground elevation (denoted as GE). We define the assumption of the process model for $h \in [1000, 2500]$ feet ASL is that $GE \in [200, 500]$ feet ASL.

- Concern 13: The vehicle has a capacity limit for the longitudinal velocity of descent.
  - Constraint: The longitudinal velocity of descent is bounded within $[150, 250]$ knots, i.e. $v \in [150, 250]$ knots.

- Assumption: $v \in [150, 250]$ knots varies over the descent angle γ and the payload weight: the steeper the descent angle is and the heavier the payload is, the slower the longitudinal velocity is. We define the assumption of the process model for $v \in [150, 250]$ is $\gamma \in [35°, 55°]$ and $weight \in [950, 1450]$ lbs.

Arrow 5: The transformation constrains the input ports and the output ports.

- Concern 14: The crosswind cannot be too strong so that the vehicle can maintain a *constant* descent angle and a *constant* longitudinal speed.
  - Constraint: f.
  - Assumption: The crosswind must be less than 40 knots, i.e. $cw \in [0, 40]$ knots.
- Concern 15: The battery level cannot be too low, so that the vehicle can maintain a *constant* descent angle and a *constant* longitudinal speed.
  - Constraint: f.
  - Assumption: The battery level must be above 40%.

In summary, all the constraint-assumption pairs can be summarized in Table.6.1.

Therefore, the process model is:

$$\begin{cases} \gamma = Des \\ v = Vel - w \cdot cos\gamma \\ s(t) = s_0 + vt \end{cases}$$

subject to the **constraints** below:

Table 6.1: The constraints of the process model and assumptions of the process model derived from each concern

Concern	Constraint	Assumption
1	$\|Vel - v\| \leq 20$ knots	Traffic congestion level is low
2	$\|Des - \gamma\| \leq 15°$	$w < 50$ knots
3	$h(v, \gamma, w) = 0$ where $v \leq 200$ knots $\gamma \in [35°, 55°]$ and $w \in [25, 45]$ knots	$cw \leq 30$ knots.
4	$Vel \leq 200$ knots	Weather is sunny
5	$Des \in [30°, 60°]$	A specific descent procedure
6	$w < 50$ knots	Operation status of weather provider is normal
7	$v > 50$ knots	Expected traffic throughput is normal
8	$\gamma > 45°$	Vertiport layout
9	$h > H + GE$	Type of ground usage
10	$w < 60$ knots	$tem \in [-10, 50]°$C
11	$\gamma \in [30°, 75°]$	Payload type; $weight \in [1000, 1500]$ lbs; $w \leq 60$ knots
12	$h \in [1000, 2500]$ feet ASL	$GE \in [200, 500]$ feet ASL
13	$v \in [150, 250]$ knots	$\gamma \in [35°, 55°]$ and $weight \in [950, 1450]$ lbs
14	f	$cw \leq 40$ knots
15	f	Battery Level is greater than 40%

$$EC = \begin{cases} h(v, \gamma, w) = 0 \\ |Vel - v| \leq 20 \text{ knots} \\ |Des - \gamma| \leq 15° \\ Vel \leq 200 \text{ knots} \\ Des \in [35°, 55°] \\ w \in [25, 45] \text{ knots} \\ v \in [150, 250] \text{ knots} \\ \gamma \in [45°, 55°] \\ h \in [max(H + GE, 1000), 2500] \text{ feet ASL} \end{cases}$$

subject to the **assumptions** below:

$$
IC = \begin{cases}
\text{Traffic congestion level is low;} \\
\text{The weather is sunny;} \\
\text{The specific descent procedure;} \\
\text{The operation status of the weather provider is normal;} \\
\text{The expected traffic throughput is normal;} \\
\text{The vertiport layout;} \\
\text{Type of the ground habitat;} \\
tem \in [-10, 50]^{\circ}\text{C} \\
\text{Payload type is passenger commute;} \\
weight \in [1000, 1450] \text{ lbs;} \\
GE \in [200, 500] \text{ feet ASL;} \\
cw \leq 30 \text{ knots;} \\
\text{The battery level is greater than 40\%}
\end{cases}
$$

6.4 Method 3: Architecting a safe controller

The reference architecture forces the specific design application to consider a wide variety of safety-critical scenarios of the environment, the controller and the controlled process, in order to identify and address the safety-critical scenarios (without the presence of the component level failure) at the same time when the controller is being defined.

In this case study, we will show how the actions defined in the reference architecture can help the descent example to identify the safety-critical scenarios and define an architecture of the controller that addresses them in the beginning.

6.4.1 The overall workflow of the controller

As shown in Fig.6.10, from left to right is the expected logic for the vehicle to descend to Point B on time and avoid the hazards at the same time. The vehicle flies at the instructed speed to catch the designated waypoints on time, so that the vehicle flies a desired 4d-trajectory that can achieve the functional goal and avoid the hazard at the same time. The workflow of controller is in an exact opposite order. **Task 1** is to define a desired 4d-trajectory based on the prescriptive constraints, that if achieved the vehicle an reach Point B on time and avoid the hazards at the same time; **Task 2** is to select 4d-waypoints on the desired 4d trajectory so that if the 4d-waypoints are respected, the desired 4d trajectory is achieved; **Task 3** is to issue the instructions on the longitudinal speed Vel and the descent angle Des to arrive at the 4d-waypoint on time.

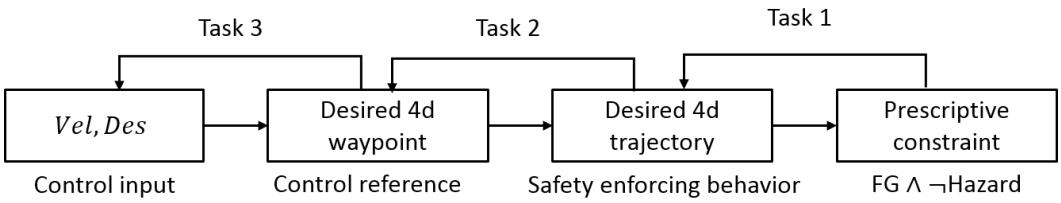


Figure 6.10: The overall workflow of the controller. FG stands for functional goal.

Finally, action takes time. Fig.6.11 is a summary of the expected time intervals ($T1, T2$ and $T3$) between the key time points of the decision making process of the controller. The key time points are at the 4d descent trajectory is decided, the 4d waypoints are decided, the instruction on velocity and descent angle is issued and the vehicle position is actually affected. Obviously, $T1 - T2$ is the expected time duration for Task 2; $T2 - T3$ is the expected time duration Task 3; $T3$ is the expected time duration of one execution of the controlled process. We are now explain each task one by one.

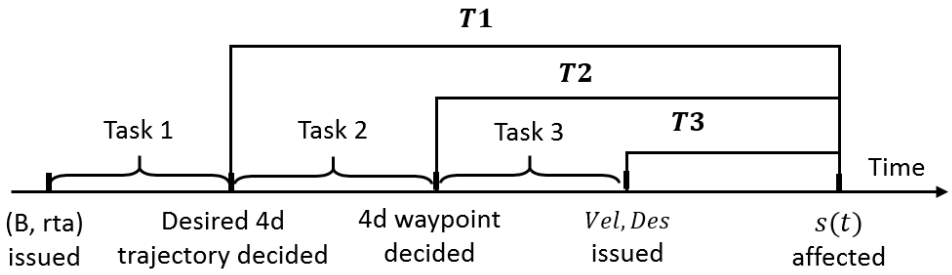


Figure 6.11: The expected time intervals between key time points of the controller's decision-making process.

6.4.2 Task 1: Perceive the prescriptive constraints to generate a safe 4d trajectory.

6.4.2.1 The overall description

Overall, this task is to perceive the prescriptive constraints to determine the safety enforcing behavior (SEB). According to RA, three stages are defined for the case study (Fig.6.12). However, because Stage 2 and Stage 3 are similar as they are subject to the same hazards, **we assume** that the descent *can always stop* immediately once Point B is arrived at rta to avoid repetition. Therefore, only the first two stages are applicable in this case study, which is why Stage 3 in Fig.6.12 is in dotted line.

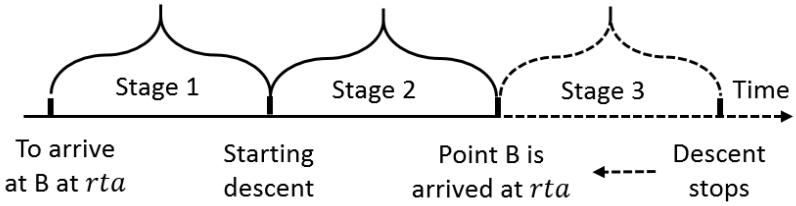


Figure 6.12: The three stages of Task 1. But because we assume the descent *can stop and thus will stop* immediately once Point B is arrived at rta, only the first two stages are applicable in this case study.

As a result, the two stages divides the timeline of Task 1 into 4 segments below (Fig.6.13), and each stage is defined in Fig.6.14 with two variables $fgStatus$ and OB defined as below:

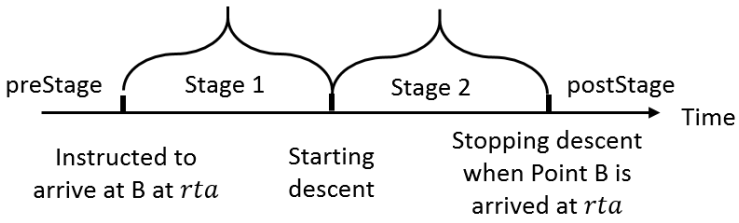


Figure 6.13: The timeline of Task 1 is divided into four segments by the two stages.

fgStatus	False	Requested		Achieved
OB	off	off	on	off
Stage	preStage	Stage1	Stage2	postStage

Figure 6.14: The definition of the stages of Task 1.

- $fgStatus$ represents the status of the descent instruction, i.e. "arrive at Point B at rta".
  - $false$: The instruction for descent has not been issued, or the instruction is cancelled, or the descent is accomplished.
  - $Requested$: The descent instruction is issued but the descent has not been accomplished.
  - $Achieved$: The descent instruction is issued and the descent is accomplished. Note that this is only transient state because it is also a $postStage$.
- OB represents whether the descent has started yet.
  - on: The descent has started and still on.
  - off: The descent has not started or already stops.

6.4.2.2 The enabling action

$\boldsymbol{EA_1}$**: Deciding the stages.** Based on the definition of the stages in Fig.6.14, 6 possible transitions are identified (Fig.6.15), each of which corresponds to one sub-action defined in the reference architecture.

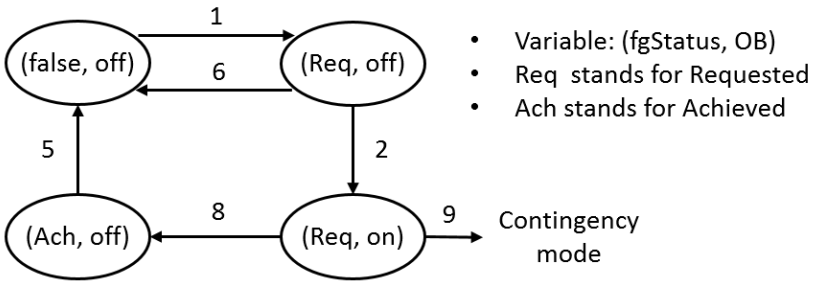


Figure 6.15: 6 possible (not considering failure) transitions between the stages are identified for this case study. The number associated with each arrow represent the corresponding sub-action in the reference architecture.

Transitioning from $preStage$ to $Stage1$.

- Trigger event: $fgStatus : false \rightarrow Requested$ [2].
- Guard condition: $OB = off$.
- Input: None.
- Output: The $stage$.
- Transformation: $stage \leftarrow Stage1$ [3].

This action corresponds to Sub-action 1 of the reference architecture.

Transitioning from $Stage1$ to $Stage2$.

- Trigger event: $OB : off \rightarrow on$.
- Guard condition: $fgStatus = Requested$.
- Input: None.
- Output: The $stage$.
- Transformation: $stage \leftarrow Stage2$.

[2]The notation $x : a \rightarrow b$ means the value of variable x transitions from a to b.
[3]The notation $x \leftarrow a$ means assign value a to variable x.

This action corresponds to Sub-action 2 of the reference architecture.

Transitioning from $postStage$ to $preStage$.

- Trigger event: $fgStatus : Achieved \rightarrow false$.
- Guard condition: $OB = off$.
- Input: None.
- Output: The $stage$.
- Transformation: $stage \leftarrow preStage$.

This action corresponds to Sub-action 5 of the reference architecture.

Transitioning from $Stage1$ to $preStage$.

- Trigger event: $fgStatus : Requested \rightarrow false$.
- Guard condition: $OB = off$.
- Input: None.
- Output: The $stage$.
- Transformation: $stage \leftarrow preStage$.

This action corresponds to Sub-action 6 of the reference architecture.

Transitioning from $Stage2$ to $postStage$.

- Trigger event: $\{fgStatus : Requested \rightarrow Achieved\} \wedge \{OB : on \rightarrow off\}$.
- Guard condition: NA.
- Input: None.
- Output: The $stage$ and $fgStatus$.

- Transformation: $stage \leftarrow postStage$ and $fgStatus \leftarrow false$.

This action corresponds to Sub-action 8 of the reference architecture.

Transitioning from $Stage2$ to contingency mode.

- Trigger event: $fgStatus : Requested \rightarrow false$.
- Guard condition: $OB = on$.
- Input: None.
- Output: CM_{15}.
- Transformation: $CM_{15} \leftarrow true$.

This action corresponds to Sub-action 9 of the reference architecture.

As a result, EA_1 can be represented in Fig.6.16 graphically.

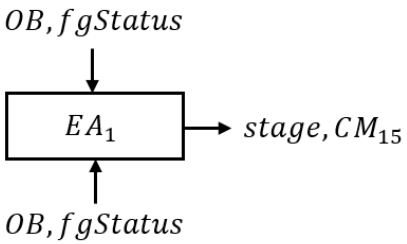


Figure 6.16: Graphical representation of EA_1 for the case study.

EA_2: Start/stop watcher. This action is to address the possibility that mst_T or msp_T is violated. Specifically, if the descent does not start before mst_T expires, or the descent does not stop before msp_T expires, the system enters contingency mode by sending out CM_4. Therefore, the following two sub-actions are defined.

Sub-action 1:

- Trigger event: mst_T expires.

- Guard condition: $stage = Stage1$.
- Input: OB.
- Output: CM_4.
- Transformation: If $OB = false$, then $CM_4 \leftarrow true$.

Sub-action 2:

- Trigger event: msp_T expires.
- Guard condition: $stage = Stage2$.
- Input: OB.
- Output: CM_4.
- Transformation: If $OB = true$, then $CM_4 \leftarrow true$.

Graphically, Action EA_2 can be represented in Fig.6.17.

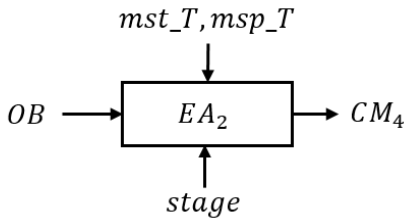


Figure 6.17: Graphical representation of EA_2.

$\boldsymbol{EA_3}$**: Deviation handler.** This action is to address the situation where the vehicle flies beyond the planned 4d-trajectory (bound). Although beyond the planned 4d-trajectory (bound) does not necessarily lead to inadequate altitude, the hazard is not actively controlled and hence safety cannot be proactively enforced by the controller.

- Trigger event: $\{\widetilde{s}(t) \notin s(t)\}$ transitions from $false$ to $true$.

- Guard condition: $stage = Stage2$.
- Input: NA.
- Output: CM_{17}.
- Transformation: $CM_{17} \leftarrow true$.

As a result, EA_3 can be represented in Fig.6.18 graphically.

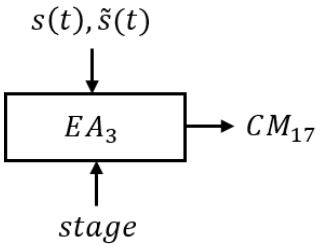


Figure 6.18: Graphical representation of EA_3 for the case study.

EA_5: SEB status monitor. This action is to record the satisfiability of the desired trajectory. It is assigned $false$ once the monitor themes detects unsatisfiability of the desired trajectory and is assigned $true$ once the generate theme find a new desired trajectory. In addition, if the unsatisfiability is not found at $Stage1$, then the SEB is set to $false$ to prevent the execution of an unsatisfactory SEB. As a result, we define the following two sub-actions for EA_5:

Sub-action 1:

- Trigger event: $sebStatus_1 \vee sebStatus_2 \vee sebStatus_3 \vee sebStatus_4 \vee sebStatus_5$: $true \rightarrow false$.
- Guard condition: $stage! = (preStage \vee postStage)$.
- Input: NA.
- Output: $sebStatus$.
- Transformation: $sebStatus \leftarrow false$.

Sub-action 2:

- Trigger event: $sebStatus_1 \vee sebStatus_2 \vee sebStatus_3 : true \rightarrow false$.
- Guard condition: $stage = Stage1$.
- Input: NA.
- Output: $(st, y(t), sp)$.
- Transformation: $(st, y(t), sp) \leftarrow false$.

Sub-action 3:

- Trigger event: $s(T)$ changes.
- Guard condition: $stage! = (preStage \vee postStage) \wedge s(T)! = false$
- Input: NA.
- Output: $sebStatus$.
- Transformation: $sebStatus \leftarrow true$.

Graphically, Action EA_5 can be represented in Fig.6.19.

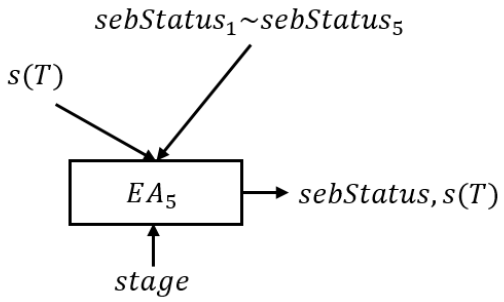


Figure 6.19: Graphical representation of EA_5.

6.4.2.3 *The main action*

We go through each main action of Task 1 for this case study. For each applicable main action, we first identify the safety-critical scenarios that are revealed by the reference architecture; then we specify the action based on the definition of the reference architecture with the inputs, the outputs, the guard condition, the trigger event and the intent of the transformation. After that, we obtain a set of connected actions (or black boxes) with well-defined ports, which by definition is the architecture of the controller for the descent example.

MA_1: Generate the performance constraints at Stage 1. This action is defined in the reference architecture because the performance constraints might change over the functional goal, the environment and the controlled process.

Safety-critical scenario. Defining this action according to the reference architecture for this case study reveals the following Safety-critical scenario (SCS):

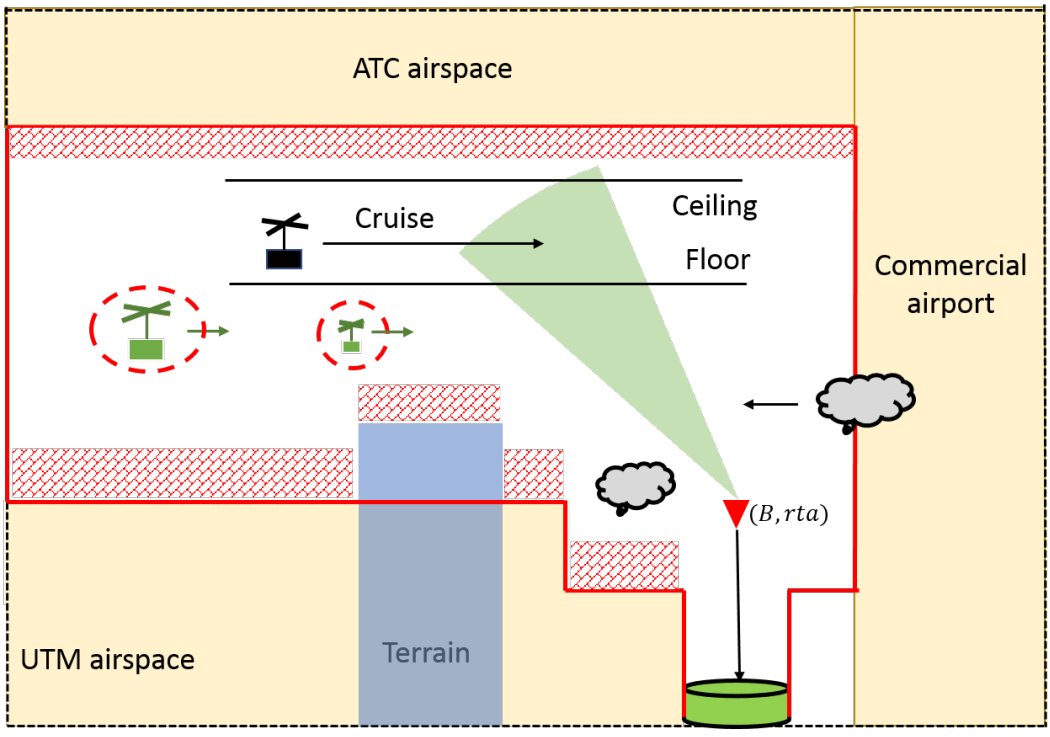


Figure 6.20: The factors to decide the performance constraints of the descent.

SCS1: The shape of the green cone of Fig.6.20 is affected the headwind and the weight of the payload. Therefore, the shape of the green cone must adjust accordingly if the headwind and the weight of the payload change.

SCS2: The minimal distance from the traffic (the dotted red circle in Fig.6.20) is affected by the type of the vehicle in the traffic and the battery level of the vehicle. Therefore, the red circle must adjust accordingly if the type of the vehicle in the traffic and the battery level of the vehicle change.

SCS3: The minimal distance from the terrain (the red shaded area above the terrain in Fig.6.20) is affected by the type of the ground habitat. Therefore, the minimal distance must adjust accordingly if the type of the ground habitat changes.

SCS4: The minimal distance from the airspace (the red shaded area below and above the airspace boundaries in Fig.6.20) is affected by the category of the adjacent airspace. Therefore, the minimal distance from the airspace boundaries must change if the category of the adjacent airspace changes.

SCS5: MA_1 may take too long to generate the performance constraint, so long that there is not enough time for MA_2 to generate mst_T before mst_T actually expires [4].

SCS1-SCS4 are the assumptions that are derived in Method 1 for pc_0^1, (pc_1^i, pc_1, pc_1^o), (pc_3^i, pc_3, pc_3^o) and (pc_4^i, pc_4, pc_4^o) respectively. Not considering the first four safety-critical scenarios SCS1-SCS4 can lead to incorrect performance constraints, which may further lead to hazard and is impossible to be identified by any verification program.

Furthermore, SCS5 implies the following hazardous scenario: because the deadline to start the descent is calculated based on the performance constraints,

[4] If mst_T is generated after mst_T expires, it is unsafe. But MA_1 takes too long so that there is not enough time to generate the desired descent trajectory is not necessarily unsafe. So, this scenario is relevant because it has safety implication. This is why the reference architecture is "safety-guided".

if it takes too long to calculate the performance constraints, it is possible that the constraints on the start time are already violated before the controller is even aware of the constraints in the first place. In this case, the vehicle enters the hazardous situation while it is still trying to figure out what is safe and unsafe.

Therefore, SCS5 requires an explicit requirement that the time duration of this action must be significantly shorter than the mst_T (for example, from the descent command is received to cruising out of the green cone) to rule out such possibility in the real system. If SCS5 is not considered in the design in the first place, it will require modeling a longer duration of MA_1 than mst_T in order to retrieve SCS5 through verification, which is unusual if the modeler does not have the hazardous scenario mentioned above in mind. As a result, the hazardous scenario can be easily omitted by a verification program.

Defining the action. According to the reference architecture, the input-output-transformation of this action are defined in Table.6.2. The inputs are all the factors identified in Method 1 that affect the performance constraints. **SCS1-SCS4** are addressed by Table.6.2.

Table.6.3 is the trigger event, the guard condition and the duration of this action. **SCS5** is addressed by the duration. The expected time duration e_1 must be short enough so that the performance constraints are not going to be generated after mst_T expires. However, there is always a chance that e_1 is too long that MA_2 eventually generates mst_T after mst_T expires. Therefore, a long enough look-ahead time must be selected to decide mst_T early to lower the likelihood of an expired mst_T to an acceptable level.

MA_2: Generate the constraints on the start time at Stage 1. This action is defined to identify the constraints on the start time of descent. These constraints are used not only to define the desired descent trajectory, but also to inform the controller the deadline to decide the start time and the desired trajectory.

Table 6.2: The input-output-transformation of MA_1.

Input source	**Input**	**output**	**Transformation**
Weather service provider	Headwind $\widehat{w}(t)$	pc_0^1	Generate/update the shape of the green cone.
The vehicle	The weight of payload pl.		
MA_1	pc_0^2		
Higher level of control	Point B and rta	pc_0^2	Generate/update Point B and rta.
The traffic surveillance	Vehicle type of the traffic vt	pc_1^i, pc_1	Generate/update the minimal distance from the traffic.
The vehicle	The battery level of the vehicle bl		
NA	NA	pc_2^i, pc_2	Pre-defined
Terrain map provider	The ground habitat gh	pc_3^i, pc_3	Generate/update the minimal distance from the terrain.
Airspace info provider	The category of the adjacent airspace cat	pc_4^i, pc_4	Generate/update the minimal distance from the airspace boundary.

Table 6.3: The trigger event, the guard condition and the duration of MA_1

	Content	**Info Source**	
Trigger	$\{stage : preStage \rightarrow Stage1\}$ $\vee\{sebStatus : true \rightarrow false\}$	$stage$	EA_1
		$sebStatus$	EA_5
Guard	$\{sebStatus = false\} \wedge \{stage = Stage1\}$	$stage$	EA_1
		$sebStatus$	EA_5
Duration	e_1 must be short enough, so that there will be enough time for MA_2 to generate mst_T before mst_T expires.	NA	

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS6: The acceptable altitude of top of descent is affected by the cloud and visibility.

SCS7: The viable time to start the descent is affected by the operational mode of the vehicle and the headwind.

SCS8: It is possible that there is conflict between mst_T and nst_T, in which case it is impossible to find a start time.

SCS9: It is possible that the vehicle deviates from the projected cruise trajectory or cruise speed.

SCS10: MA_2 may take too long to generate mst_T, so long that mst_T already expires by the time mst_T is generated.

SCS11: It is possible that MA_2 waits too long for the resolution of the conflict between mst_T and nst_T, so long that there is not enough time left to generate and start the desired descent trajectory before mst_T expires[5].

SCS6 and SCS7 are the assumptions that are derived in Method 1 for mst_0^1 and mst_0^2 respectively. Not considering them may lead to failure to achieve the functional goal regardless of the hazard, an omission that can never be retrieved by verification. SCS8 is caused by specific combinations of the sub-hazards that make starting the descent impossible. If the defined hazards happen to not include such combinations, this scenario can never be captured by verification.

SCS9 is about the possibility that projection of the cruise trajectory and speed can be wrong due to for example deviation caused by aleatoric uncertainty. If the projection is wrong, then the constraints on the start time can also be wrong. In a verification program, because such projection comes from outside the controller being designed, it is reasonably taken as given and out of the scope of verification. However, if such possibility is not explicitly designed for, the system will enter the hazardous condition by faithfully operating as designed.

SCS10 implies that MA_2 must be accomplished in a short time because the

[5]The reason that they are relevant (i.e. should not be abstracted away) is that they may lead to hazard, which is why this is safety-guided, not say efficiency-guided.

controller needs to know the deadline of mst_T to act accordingly. SCS11 implies when $\{mst\_T, nst\_T, cst\_T\}$ has internal conflicts, MA_2 can only wait for the resolution till some time before mst_T because after that point there will be no time left to generate a new descent trajectory and actually start it. In complete randomness, there is no guarantee that a verification program can capture the two possibilities above with model execution due to state explosion.

Defining the action. The input-output-transformation of MA_2 is defined in Table.6.4 to decide the constraints on the start time. **SCS6-SCS7** are addressed in $mst\_T_0^1$ and $mst\_T_0^2$. If **SCS8** becomes true, then RfR_1 is sent to higher level of control for a new descent command. In addition, if **SCS11** becomes true (i.e. $t_c > \overline{mst\_T} - T1 - e_3$), then CM_1 is sent for the system to enter contingency mode.

Table 6.4: The input-output-transformation of MA_2. Note that nst_0^3 is not included in the table because it is addressed as the guard condition of MA_3. The specific design application can deviate from the reference architecture as long as the deviation can be justified.

Input source	**Input**	**Output**	**Transformation**
MA_1	pc_0^1	$mst\_T_0^1$	Generate the time window for the must-start condition with respect to Point B.
Weather service provider	Cloud cd, visibility vb		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
MA_2	$mst\_T_0^1$	$nst\_T_0^1$	$nst\_T_0^1 = \neg mst\_T_0^1$
MA_1	pc_0^2	$mst\_T_0^2$	Generate the time window for the must-start condition w.r.t. rta.
The vehicle	Vehicle operation mode vm		
Weather service provider	Headwind $\widehat{w}(t)$		
MA_2	$mst\_T_0^2$	$nst\_T_0^2$	$nst\_T_0^2 = \neg mst\_T_0^2$

Input source	Input	Output	Transformation
MA_1	pc_1^i	mst_T_1	Generate the time window for the must-start condition w.r.t. the traffic.
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Traffic surveillance provider	The projected traffic trajectory $\widehat{s}_{tf}$, the projected traffic speed $\widehat{v}_{tf}$.		
MA_1	pc_1	nst_T_1	Generate the time window for the must-not-start condition w.r.t. the traffic.
The vehicle	The current vehicle position $\widetilde{s}$		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Traffic surveillance provider	The projected traffic trajectory $\widehat{s}_{tf}$, the projected traffic speed $\widehat{v}_{tf}$, the current traffic position $\widetilde{s}_{tf}$.		
MA_2	mst_T_1, nst_T_1	cst_T_1	$cst\_T_1 = \neg(mst\_T_1 \cup nst\_T_1)$
MA_1	pc_2^i	mst_T_2	Generate the time window for the must-start condition w.r.t. the weather.
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Weather service provider	The projected weather impacted area $\widehat{wa}$		

Input source	Input	Output	Transformation
MA_1	pc_2	nst_T_2	Generate the time window for the must-not-start condition w.r.t. the weather.
The vehicle	The current vehicle position $\widetilde{s}$		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Weather service provider	The projected weather impacted area $\widehat{wa}$, the current weather impacted area $\widetilde{wa}$		
MA_2	mst_T_2, nst_T_2	cst_T_2	$cst\_T_2 = \neg(mst\_T_2 \cup nst\_T_2)$
MA_1	pc_3^i	mst_T_3	Generate the time window for the must-start condition w.r.t. the terrain.
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Terrain map provider	The location of the terrain lt, the elevation of the terrain et.		
MA_1	pc_3	nst_T_3	Generate the time window for the must-not-start condition w.r.t. the terrain.
The vehicle	The current vehicle position $\widetilde{s}$		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Terrain map provider	The location of the terrain lt, the elevation of the terrain et.		

Input source	Input	Output	Transformation
MA_2	mst_T_3, nst_T_3	cst_T_3	$cst\_T_3 = \neg(mst\_T_3 \cup nst\_T_3)$
MA_1	pc_4^i	mst_T_4	Generate the time window for the must-start condition w.r.t. the airspace boundary.
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Airspace info provider	The projected airspace boundary $\widehat{ab}$.		
MA_1	pc_4	nst_T_4	Generate the time window for the must-not-start condition w.r.t. the airspace boundary.
The vehicle	The current vehicle position $\widetilde{s}$		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Airspace info provider	The current airspace boundary $\widetilde{ab}$, the projected airspace boundary $\widehat{ab}$.		
MA_2	mst_T_4, nst_T_4	cst_T_4	$cst\_T_4 = \neg(mst\_T_4 \cup nst\_T_4)$
MA_2	$mst\_T_0^1, nst\_T_0^1, mst\_T_0^2, nst\_T_0^2, mst\_T_1, nst\_T_1, cst\_T_1, mst\_T_2, nst\_T_2, cst\_T_2, mst\_T_3, nst\_T_3, cst\_T_3, mst\_T_4, nst\_T_4, cst\_T_4$	mst_T, nst_T, cst_T	Aggregate the individual constraints.
		RfR_1	If there is no viable st, then send RfR_1.
		CM_1	If $t_c > \overline{mst\_T} - T1 - e_3$

According to the reference architecture, the trigger event, the guard condition and the duration are summarized in Table.6.5. Intuitively, if the the projected

cruise trajectory is deviated, no desired descent trajectory can be made before the deviation is corrected. This is actually **SCS9**, which is addressed by the guard condition $\{\widehat{s}^i(t) = \widetilde{h}\} \wedge \{\widehat{v}^i(t) = \widetilde{v}\}$. Furthermore, e_2 must be short enough so that mst_T will not be generated after mst_T expires. However, there is always a chance that e_2 is too long that mst_T is generated after mst_T expires. Therefore, a long enough look-ahead time must be selected to decide mst_T early to lower the likelihood of an expired mst_T to an acceptable level. This addresses **SCS10**.

Table 6.5: The trigger event, the guard condition and the duration of MA_2

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{sebStatus = false\}$ $\wedge\{stage = Stage1\}$ $\wedge\{\widehat{s}^i(t) = \widetilde{s}\} \wedge \{\widehat{v}^i(t) = \widetilde{v}\}$	$stage$	EA_1
		$sebStatus$	EA_5
		$\widehat{s}^i(t), \widehat{v}^i(t)$	The controller for cruise
		$\widetilde{v}, \widetilde{s}$	The vehicle
Duration	e_2 must be short enough, so that mst_T can be generated before mst_T expires.	NA	

MA_3: Generate the desired descent 4d trajectory at Stage 1. This action is to generate the desired descent trajectory that satisfy all the prescriptive constraints, so that Point B can be reached on time and in the meantime hazards can be avoided.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS12: It can be impossible to find the desired descent trajectory due to the internal conflicts among the sub-hazards.

SCS13: It is impossible to decide the descent trajectory if the projected cruise trajectory is deviated.

SCS14: MA_3 may take too long to generate the desired descent trajectory, so long that there is not enough time left to actually start the desired descent trajectory before mst_T expires.

SCS15: It is possible that MA_3 waits too long for the resolution of the conflict among the sub-hazards, so long that there is not enough time left to start the desired descent trajectory before mst_T expires.

SCS12, SCS13, SCS14 and SCS15 are similar to the SCS8, SCS9 SCS10 and SCS11 respectively. Consequently, they can be missed by a verification program for the same reasons.

Defining the action. The input-output-transformation of MA_3 is defined in Table.6.6. If **SCS12** happens, RfR_2 is sent to the higher level of control for a new descent instruction. If **SCS14** or **SCS15** happens, CM_2 is sent for the system to enter the contingency mode.

According to the reference architecture, there is no trigger event for this action, and the guard condition is summarized in Table.6.7. The guard condition $\wedge\{\widehat{s}^i(t) = \widetilde{h}\} \wedge \{\widehat{v}^i(t) = \widetilde{v}\}$ is to address **SCS13**; $\{dc = true\}$ is the implementation of nst_0^3 that is specific to this case study.

The resulting desired descent trajectory $s(T)$ is shown in Fig.6.21. Note that $s(T)$ does not have to be a single trajectory. In fact, it is more likely to be *a set of trajectories* due to the unavoidable aleatoric uncertainty and the lack of knowledge of the operational details of controlled process. As shown in Fig.6.21, $s(T)$ is any trajectory within the area confined by the red line, as long as the vehicle can descend to altitude h_1 between $[t_1, t_2]$ and h_2 between $[t_3, t_4]$.

Note that an improper selection of $[t_1, t_2]$ and $[t_3, t_4]$ can make the desired trajectory infeasible. But this is not captured in the prescriptive constraints, which is aligned with the observation that the prescriptive constraints only rules out some of "the infeasible" but does not guarantee feasibility.

Table 6.6: The input-output-transformation of MA_3.

Input source	**Input**	**Output**	**Transformation**
MA_2	mst_T, nst_T, cst_T	Start time of the descent st	Decide the start time of the descent.
MA_1	pc_0^1, pc_0^2, pc_1, pc_2, pc_3, pc_4	• $s(T) = \{s(t) \| t \in [st, rta]\}$ where $s(st)$ must be consistent with the projected cruise trajectory; • RfR_2 if $s(T)$ cannot be found; • CM_2 if $t_c > \overline{mst\_T} - T1$	Decide the desired descent trajectory $s(T)$, where $T = [st, rta]$
The controller for cruise	The projected cruise trajectory $\widehat{s^i}(t)$, the projected cruise speed $\widehat{v^i}(t)$		
Traffic surveillance provider	The projected traffic trajectory $\widehat{s_{tf}}$, the projected traffic speed $\widehat{v_{tf}}$.		
Weather service provider	The projected weather impacted area $\widehat{wa}$.		
Terrain map provider	The location of the terrain lt, and elevation of the terrain et.		
Airspace info provider	The projected airspace boundary $\widehat{ab}$		

Table 6.7: The trigger event, the guard condition and the duration of MA_3

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{sebStatus = false\}$ $\wedge\{stage = Stage1\} \wedge \{\widehat{s^i}(t) = \widetilde{s}\}$ $\wedge\{\widehat{v^i}(t) = \widetilde{v}\} \wedge \{dc = true\}$	$stage$	EA_1
		$sebStatus$	EA_5
		$\widehat{s^i}(t), \widehat{v^i}(t)$	The controller for cruise
		$\widetilde{v}, \widetilde{s}$	The vehicle
		dc	Vertiport
Duration	NA	NA	

MA_4: Update the performance constraints at Stage 1. This is the first action of the monitor theme of Stage 1. Once a desired descent trajectory is created,

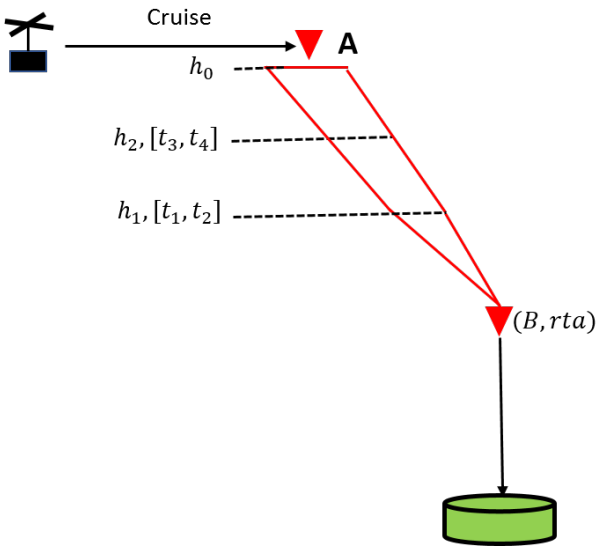


Figure 6.21: The resulting desired descent trajectory $s(T)$.

this action updates the performance constraints if any of the inputs changes.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios. Note that SCS1-SCS4 also applies to MA_4. But because they have been explained in MA_1 and will be addressed by MA_4 in the same way as MA_1, they are not mentioned here to avoid repetition.

SCS16: The change in the descent instruction, the environment or the vehicle may render the original performance constraints invalid, which may make the start time or the desired descent trajectory unsatisfactory before the descent even starts.

SCS17: MA_4 may take too long to update the performance constraints, so long that there will be no enough time for MA_5 to generate mst_T and nst_T before the selected st passes or before mst_T expires.

SCS16 implies that the desired descent trajectory can become unsatisfactory due to input changes before the descent even starts. If such possibility is not built into the model, a verification program cannot find it. SCS17 is similar to SCS5, and hard to capture if it is not explicitly built in the model.

Defining the action. The activities to *update* the performance constraints are the same as those to *generate* the performance constraints. As a result, MA_4 has a similar input-output-transformation with MA_1 in Table.6.8. Because of the assumption that an action always respond to the latest input values (see the "assumption" section of the reference architecture), if any of the inputs changes, MA_4 will be activated to recalculate the performance constraints. This addresses **SCS16**.

The trigger event, the guard condition and the duration of MA_4 (Table.6.9) are different from MA_1. According to the reference architecture, MA_4 is only triggered when a new desired descent trajectory is generated, which is characterized by $sebStatus$ transitioning from $false$ to $true$. Once triggered, MA_4 keeps monitoring the change of the inputs as long as there is a defined descent trajectory and the $stage$ is still at $Stage1$, i.e. $\{sebStatus = true\} \wedge \{stage = Stage1\}$. $\{dc = true\}$ in the guard condition is the implementation of nst_0^3 that is specific to this case study. Furthermore, the e_4 must be short enough so that there will be enough time for MA_5 to generate mst_T and nst_T before the selected st passes and before mst_T expires. However, there is always a chance that e_4 is too long that MA_5 eventually generates mst_T and nst_T after st passes or mst_T expires. Therefore, a long enough look-ahead time must be selected to decide mst_T early to lower the likelihood of an expired mst_T or a passed st to an acceptable level. This addresses **SCS17**.

MA_5: Update the constraints on the start time at Stage 1. This action is to update the constraints on the start time before the descent even starts.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios. Note that SCS6-SCS9 also applies to MA_5. But because they have been explained in MA_2 and will be addressed by MA_5 in the same way as MA_2, they are not mentioned here to

Table 6.8: The input-output-transformation of MA_4.

Input source	**Input**	**output**	**Transformation**
Weather service provider	Headwind $\widehat{w}(t)$	pc_0^1	Generate/update the shape of the green cone.
The vehicle	The weight of payload pl.		
MA_4	pc_0^2		
Higher level of control	Point B and rta	pc_0^2	Generate/update Point B and rta.
The traffic surveillance	Vehicle type of the traffic vt	pc_1^i, pc_1	Generate/update the minimal distance from the traffic.
The vehicle	The battery level of the vehicle bl		
NA	NA	pc_2^i, pc_2	Pre-defined
Terrain map provider	The ground habitat gh	pc_3^i, pc_3	Generate/update the minimal distance from the terrain.
Airspace info provider	The category of the adjacent airspace cat	pc_4^i, pc_4	Generate/update the minimal distance from the airspace boundary.

Table 6.9: The trigger event, the guard condition and the duration of MA_4

	Content	**Info Source**	
Trigger	$\{sebStatus : false \rightarrow true\}$	$sebStatus$	EA_5
Guard	$\{sebStatus = true\}$ $\wedge\{stage = Stage1\} \wedge \{dc = true\}$	$stage$	EA_1
		$sebStatus$	EA_5
		dc	Vertiport
Duration	e_4 must be short enough so that MA_5 can generate mst_T and nst_T before st passes and mst_T expires.	NA	

avoid repetition.

SCS18: The change of the performance constraints, the projected cruise trajectory, the environment or the vehicle may render the original constraints on the start time invalid, which may make the desired descent trajectory

unsatisfactory before the descent even starts.

SCS19: MA_5 may take too long to update the constraints on the start time, so long that mst_T and nst_T are generated after the st passes or mst_T expires.

SCS20: It is possible that MA_5 waits too long for the resolution of the conflict between mst_T and nst_T, so long that it is after the descent starts at st.

SCS21: It is possible that MA_5 waits too long for the resolution of the conflict between mst_T and nst_T, so long that there is no enough time to generate a new descent trajectory and actually starts it.

SCS18 and SCS19 are similar to SCS16 and SCS17. SCS20 implies the scenario where the previously decided st is within the updated nst_T. SCS21 implies the scenario where the descent does not start after the updated mst_T expires.

Defining the action. The input-output-transformation as MA_5 in summarized in Table.6.10. Because the activities to *update* the constraints on the start time are the same as those to *generate* the constraints on the start time, Table.6.10 is very similar with Table.6.4. The only major difference is that MA_5 responds differently to the conflict between mst_T and nst_T from MA_2, i.e. the last row of both tables. Because of the assumption that an action always respond to the latest input values (see the "assumption" section of the reference architecture), if any of the inputs changes, MA_5 will be activated to recalculate the performance constraints. This addresses **SCS18**. When there is conflict between mst_T and nst_T, RfR_3 is sent to higher level of control for resolution. In the meantime, if st is passed while there is conflict between mst_T and nst_T, then the defined descent trajectory is set to $false$ and hence will not be executed. This addresses **SCS20**. If mst_T expires then the controller enters the contingency mode by sending out CM_3 to EA_4. This addresses **SCS21**.

The guard condition of MA_5 (Table.6.11) is different from MA_2. Once any

of the inputs changes, MA_5 recalculates the constraints on the start time as long as there is a defined descent trajectory, the $stage$ is still at $Stage1$, and the projected cruise trajectory is not deviated, i.e. $\{sebStatus = true\} \wedge \{stage = Stage1\} \wedge \{\widehat{s}^i(t) = \widetilde{h}\} \wedge \{\widehat{v}^i(t) = \widetilde{v}\}$. In additoin, e_5 to generate mst_T and nst_T must be short enough so that there will be enough time to generate mst_T and nst_T before the selected st starts and mst_T expires. However, there is always a chance that e_5 is too long that mst_T and nst_T are generated after st passes or mst_T expires. Therefore, a long enough look-ahead time must be selected to decide mst_T early to lower the likelihood of an expired mst_T or a passed st to an acceptable level. This addresses **SCS19**.

Table 6.10: The input-output-transformation of MA_5. Note that nst_0^3 is not included in the table because it has been addressed as the guard condition of MA_4. This table is very similar to Table.6.4. The only major difference is the last row, where ϵ is an infinitesimal number.

Input source	**Input**	**Output**	**Transformation**
MA_4	pc_0^1	$mst\_T_0^1$	Generate the time window for the must-start condition w.r.t. Point B.
Weather service provider	Cloud cd, visibility vb		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$.		
MA_5	$mst\_T_0^1$	$nst\_T_0^1$	$nst\_T_0^1 = \neg mst\_T_0^1$
MA_4	pc_0^2	$mst\_T_0^2$	Generate the time window for the must-start condition w.r.t. rta.
The vehicle	Vehicle operation mode vm		
Weather service provider	Headwind $\widehat{w}(t)$		
MA_5	$mst\_T_0^2$	$nst\_T_0^2$	$nst\_T_0^2 = \neg mst\_T_0^2$
MA_4	pc_1^i		

Input source	Input	Output	Transformation
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$.	mst_T_1	Generate the time window for the must-start condition w.r.t. the traffic.
Traffic surveillance provider	The projected traffic trajectory $\widehat{s}_{tf}$, the projected traffic speed $\widehat{v}_{tf}$.		
MA_4	pc_1	nst_T_1	Generate the time window for the must-not-start condition w.r.t. the traffic.
The vehicle	The current vehicle position $\widetilde{s}$		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$.		
Traffic surveillance provider	The projected traffic trajectory $\widehat{s}_{tf}$, the projected traffic speed $\widehat{v}_{tf}$, the current traffic position $\widetilde{s}_{tf}$.		
MA_5	mst_T_1, nst_T_1	cst_T_1	$cst\_T_1 = \neg(mst\_T_1 \cup nst\_T_1)$
MA_4	pc_2^i	mst_T_2	Generate the time window for the must-start condition w.r.t. the weather.
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Weather service provider	The projected weather impacted area $\widehat{wa}$		
MA_4	pc_2		

Input source	Input	Output	Transformation
The vehicle	The current vehicle position $\widetilde{s}$	nst_T_2	Generate the time window for the must-not-start condition w.r.t. the weather.
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$.		
Weather service provider	The projected weather impacted area $\widehat{wa}$, the current weather impacted area $\widetilde{wa}$		
MA_5	mst_T_2, nst_T_2	cst_T_2	$cst\_T_2 = \neg(mst\_T_2 \cup nst\_T_2)$
MA_4	pc_3^i	mst_T_3	Generate the time window for the must-start condition w.r.t. the terrain.
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$.		
Terrain map provider	The location of the terrain lt, the elevation of the terrain et.		
MA_4	pc_3	nst_T_3	Generate the time window for the must-not-start condition w.r.t. the terrain.
The vehicle	The current vehicle position $\widetilde{s}$		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$.		
Terrain map provider	The location of the terrain lt, the elevation of the terrain et.		

Input source	Input	Output	Transformation
MA_5	mst_T_3, nst_T_3	cst_T_3	$cst\_T_3 = \neg(mst\_T_3 \cup nst\_T_3)$
MA_4	pc_4^i	mst_T_4	Generate the time window for the must-start condition w.r.t. the airspace boundary.
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$.		
Airspace info provider	The projected airspace boundary $\widehat{ab}$.		
MA_4	pc_4	nst_T_4	Generate the time window for the must-not-start condition w.r.t. the airspace boundary.
The vehicle	The current vehicle position $\widetilde{s}$		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$.		
Airspace info provider	The current airspace boundary $\widetilde{ab}$, the projected airspace boundary $\widehat{ab}$.		
MA_5	mst_T_4, nst_T_4	cst_T_4	$cst\_T_4 = \neg(mst\_T_4 \cup nst\_T_4)$
MA_5	$mst\_T_0^1, nst\_T_0^1, mst\_T_0^2$, $nst\_T_0^2, mst\_T_1, nst\_T_1$, $cst\_T_1, mst\_T_2, nst\_T_2$, $cst\_T_2, mst\_T_3, nst\_T_3$, $cst\_T_3, mst\_T_4, nst\_T_4$, cst_T_4	mst_T, nst_T, cst_T	Assign $false$ if there is no viable st and $t_c = st - \epsilon$.
		RfR_1	If there is no viable st, then send RfR_1.
		CM_1	If $t_c > \overline{mst\_T} - T1 - e_6 - e_1 - e_2 - e_3$

Table 6.11: The trigger event, the guard condition and the duration of MA_5

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{sebStatus = true\}$ $\wedge\{stage = Stage1\}$ $\wedge\{\widehat{s}^i(t) = \widetilde{h}\} \wedge \{\widehat{v}^i(t) = \widetilde{v}\}$	$stage$	EA_1
		$sebStatus$	EA_5
		$\widehat{s}^i(t), \widehat{v}^i(t)$	The controller for cruise
		$\widetilde{v}, \widetilde{h}$	The vehicle
Duration	e_5 must be short enough so that mst_T and nst_T can be generated before st passes and mst_T expires.	NA	

MA_6: Monitor the satisfiability of the desired start time at Stage 1. This action is to decide whether the previously defined start time of the descent is still satisfactory when mst_T and nst_T are updated.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the safety-critical scenarios below.

SCS22: The change of mst_T and nst_T may make the previously defined st unsatisfactory.

SCS23: MA_6 may take too long, so long that st passes before MA_6 finishes.

SCS22 is similar to SCS18. SCS23 implies the scenario where MA_6 takes too long to calculate whether the defined start time is satisfactory under the new constraints, so long that the decision cannot be made before the descent is supposed to start. If that is the case, the descent will start without knowing whether the start time will lead to a hazard. There is no guarantee that a verification program can capture this scenario if it is not specifically designed in the verification program.

Defining the action. The input-output-transformation of MA_6 is defined in Table.6.12, which addresses **SCS22** and **SCS23**. The trigger event, guard condition and the duration are defined in Table.6.13.

Table 6.12: The input-output-transformation of MA_6 .

Input source	**Input**	**Output**	**Transformation**
MA_3	The start time st	$sebStatus_1$	If $st \subseteq nst\_T, sebStatus_1 \leftarrow false$; If $st \subseteq mst\_T \cup cst\_T, sebStatus_1 \leftarrow true$; If the decision cannot made before st, $sebStatus_1 \leftarrow false$.
MA_5	mst_T, nst_T,cst_T		

Table 6.13: The trigger event, the guard condition and the duration of MA_6

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{sebStatus = true\} \wedge \{stage = Stage1\}$	$stage$	EA_1
		$sebStatus$	EA_5
Duration	NA	NA	

MA_7: Monitor the satisfiability of the descent trajectory at Stage 1. This action is to decide whether a defined descent trajectory is satisfactory before the descent starts.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the two safety-critical scenarios below.

SCS24: The previously decided descent trajectory can become unsatisfactory before it even starts due to the change of the performance constraints, the projected cruise trajectory and the environment.

SCS25: MA_7 may take too long to check the satisfiability of $s(T)$, so long that the descent starts at st without knowledge of the satisfiability of $s(T)$.

SCS26: The defined descent trajectory may become impossible if the projected cruise trajectory is deviated.

A verification program may find that the change of the environment may lead to unsatisfiability of a planned descent trajectory by the occurrence of an accident during the descent. Measures such as monitoring the change of the environment may be taken to rule out this scenario *during the descent*. But the real

problem is that, if the controller finds the planned descent trajectory is unsatisfactory *before the descent*, then the descent should not have started in the first place unless a new satisfactory descent trajectory can be found. SCS24 reflects such a basic safety-guide design philosophy, which cannot be instilled through verification.

SCS25 and R26 is similar to SCS23 and SCS13 respectively.

Defining the action. The input-output-transformation of MA_7 is defined in Table.6.14 to address **SCS24**. The transformation is to examine the satisfiability of performance constraints $\{pc_0^1 \wedge pc_0^2 \wedge pc_1 \wedge pc_2 \wedge pc_3 \wedge pc_4\}$ by examining $s(T)$ with respect to $\{\widehat{s}_{tf} \wedge \widehat{v}_{tf} \wedge \widehat{wa} \wedge lt \wedge ct \wedge \widehat{ab}\}$. If any of the performance constraints is violated, then $sebStatus_2$ is assigned $false$, otherwise is assigned $true$. If such a decision cannot be made before st, then $sebStatus_2$ is assigned $false$. This addresses **SCS25**.

The trigger event, guard condition and the duration are defined in Table.6.15. $\{\widehat{s}^i(t) = \widetilde{h}\} \wedge \{\widehat{v}^i(t) = \widetilde{v}\}$ is to make sure that the projected cruise trajectory is not deviated, which addresses **SCS26**.

MA_9: Update the performance constraints at Stage 2. This is the first action of the monitor theme of Stage 2. It updates the performance constraints on the defined descent trajectory in real time. Because the assumption that the descent *can always stop* immediately once Point B is arrived at rta, there is no constraint from pc^o on the stop time. Therefore, only pc is considered here.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios. Note that SCS1-SCS4 also applies to MA_9. But because they have been explained in MA_1 and will be addressed by MA_9 in the same way as MA_1, they are not mentioned here to avoid repetition.

Table 6.14: The input-output-transformation of MA_7.

Input source	**Input**	**Output**	**Transformation**
MA_3	The desired descent trajectory $s(T)$	$sebStatus_2$	• Compare $s(T)$ with $\{pc_0^1 \wedge pc_0^2 \wedge pc_1 \wedge pc_2 \wedge pc_3 \wedge pc_4\}$ in the context of $\{\widehat{s}_{tf} \wedge \widehat{v}_{tf} \wedge \widehat{wa} \wedge lt \wedge ct \wedge \widehat{ab}\}$ • If the decision cannot made before st, $sebStatus_2 \leftarrow false$.
MA_4	pc_0^1, pc_0^2, pc_1, pc_2, pc_3, pc_4		
MA_3	st		
The controller for cruise	The projected cruise trajectory $\widehat{s}^i(t)$, the projected cruise speed $\widehat{v}^i(t)$		
Traffic surveillance provider	The projected traffic trajectory $\widehat{s}_{tf}$, the projected traffic speed $\widehat{v}_{tf}$.		
Weather service provider	The projected weather impacted area $\widehat{wa}$.		
Terrain map provider	The location of the terrain lt and elevation of the terrain et.		
Airspace info provider	The projected airspace boundary $\widehat{ab}$		

Table 6.15: The trigger event and the guard condition of MA_7

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{sebStatus = true\} \wedge \{stage = Stage1\}$ $\wedge\{\widehat{s}^i(t) = \widetilde{h}\} \wedge \{\widehat{v}^i(t) = \widetilde{v}\}$	$stage$	EA_1
		$sebStatus$	EA_5
		$\widehat{s}^i(t), \widehat{v}^i(t)$	The controller for cruise
		$\widetilde{v}, \widetilde{h}$	The vehicle
Duration	NA	NA	

SCS27: The change in the descent instruction, the environment or the vehicle may lead to the change of the performance constraints, which may potentially make the descent trajectory that has not been executed unsatisfactory.

SCS28: MA_9 may take too long to update the performance constraints, so long that there will be no enough time for MA_{10} to generate msp_T and nsp_T before the rta passes and msp_T expires.

SCS27 and SCS28 are similar with SCS16 and SCS17 respectively.

Defining the action. The activities to *update* the performance constraints are summarized in Table.6.16. Because of the assumption that an action always respond to the latest input values (see the "assumption" section of the reference architecture), if any of the inputs changes, MA_9 will be activated to recalculate the performance constraints. This addresses **SCS27**.

According to the reference architecture, the trigger event, the guard condition and the duration of MA_9 are summarized in Table.6.17. $\{dc = true\}$ in the guard condition is the implementation of msp_0^3 that is specific to this case study. Furthermore, the e_9 must be short enough so that there will be enough time for MA_{10} to generate msp_T and nsp_T before the rta passes and before msp_T expires. However, there is always a chance that e_9 is too long that MA_{10} eventually generates msp_T and nsp_T after rta passes or msp_T expires. Therefore, a long enough look-ahead time must be selected to decide msp_T early to lower the likelihood of an expired msp_T or a passed rta to an acceptable level. This addresses **SCS28**.

MA_{10} : Update the constraints on the stop time at Stage 2. This action is to update the constraints on the stop time of the descent in real time.

Safety-critical scenarios. Defining this action according to the reference ar-

Table 6.16: The input-output-transformation of MA_9.

Input source	**Input**	**output**	**Transformation**
Weather service provider	Headwind $\widehat{w}(t)$	pc_0^1	Generate/update the shape of the green cone.
The vehicle	The weight of payload pl.		
MA_9	pc_0^2		
Higher level of control	Point B and rta	pc_0^2	Generate/update Point B and rta.
The traffic surveillance	Vehicle type of the traffic vt	pc_1	Generate/update the minimal distance from the traffic.
The vehicle	The battery level of the vehicle bl		
NA	NA	pc_2	Pre-defined
Terrain map provider	The ground habitat gh	pc_3	Generate/update the minimal distance from the terrain.
Airspace info provider	The category of the adjacent airspace cat	pc_4	Generate/update the minimal distance from the airspace boundary.

Table 6.17: The trigger event, the guard condition and the duration of MA_9

	Content	**Info Source**	
Trigger	$\{stage : Stage1 \rightarrow Stage2\} \vee \{sebStatus : false \rightarrow true\}$	$sebStatus$	EA_5
Guard	$\{stage = Stage2\} \wedge \{sebStatus = true\} \wedge \{dc = true\}$	$stage$	EA_1
		$sebStatus$	EA_5
		dc	Vertiport
Duration	e_9 must be short enough so that MA_{10} can generate msp_T and nsp_T before rta passes and msp_T expires.	NA	

chitecture reveals the following safety-critical scenarios.

SCS29: The change in the performance constraints, the environment or the ve-

hicle may lead to the change of the constraints on the stop time, which may potentially make the descent trajectory that has not been executed unsatisfactory.

SCS30: MA_{10} may take too long to update the constraints on the stop time, so long that msp_T and nsp_T are generated after the rta passes or msp_T expires.

SCS31: It is possible there is conflict between msp_T and nsp_T.

SCS29, SCS30 and SCS31 are similar with SCS27, SCS28 and SCS8.

Defining the action. The input-output-transformation as MA_{10} in summarized in Table.6.18. Because of the assumption that an action always respond to the latest input values (see the "assumption" section of the reference architecture), if any of the inputs changes, MA_{10} will be activated to recalculate the constraints on the stop time. This addresses **SCS29**. Moreover, when there is conflict between msp_T and nsp_T, a new descent trajectory is generated to resolve such conflict by assigning the outputs to $false$. This addresses **SCS31**.

The guard condition of MA_{10} in summarized in Table.6.19. e_{10} must be short enough so that msp_T and nsp_T are generated before rta passes and msp_T expires. However, there is always a chance that e_{10} is too long that msp_T and nsp_T are generated after rta passes or msp_T expires. Therefore, a long enough look-ahead time must be selected to decide msp_T early to lower the likelihood of an expired msp_T or a passed rta to an acceptable level. This addresses **SCS30**.

Table 6.18: The input-output-transformation of MA_{10}. Note that msp_0^3 is not included in the table because it has been addressed as the guard condition of MA_9.

Input source	**Input**	**Output**	**Transformation**
MA_9	pc_0^1	$msp\_T_0^1$	Generate the time window for the must-stop condition w.r.t. the green cone.
MA_3/MA_{14}	$s(T)$		
MA_9	pc_0^2	$msp\_T_0^2$	Generate the time window for the must-stop condition w.r.t. rta.
MA_3/MA_{14}	$s(T)$		
MA_9	pc_0^2	$nsp\_T_0^1$	Generate the time window for the must-not-stop condition with respect to rta.
MA_{10}	$msp\_T_0^1$, $msp\_T_0^2$, $nsp\_T_0^1$	csp_T_0	Generate the time window for the can-stop condition w.r.t. the green cone and rta.
MA_9	pc_1	$msp\_T_1^1$	Generate the time window for the must-stop condition w.r.t. the traffic.
MA_3/MA_{14}	$s(T)$		
Traffic surveillance provider	The projected traffic trajectory $\widehat{s}_{tf}$, the projected traffic speed $\widehat{v}_{tf}$.		
MA_{10}	$msp\_T_2^1$	$csp\_T_2^1$	Generate the time window for the can-stop condition w.r.t. the traffic.
MA_9	pc_2		
MA_3/MA_{14}	$s(T)$		

Input source	**Input**	**Output**	**Transformation**
The weather service provider	The projected weather impacted area $\widehat{wa}$	$msp\_T_2^1$	Generate the time window for the must-stop condition w.r.t. the weather.
MA_{10}	$msp\_T_2^1$	$csp\_T_2^1$	Generate the time window for the can-stop condition w.r.t. the weather.
MA_9	pc_4	$msp\_T_4^1$	Generate the time window for the must-stop condition w.r.t. the airspace boundary.
MA_3/MA_{14}	$s(T)$		
Airspace info provider	The projected airspace boundary $\widehat{ab}$		
MA_{10}	$msp\_T_4^1$	$csp\_T_4^1$	Generate the time window for the can-stop condition with respect to the airspace boundary.
MA_{10}	$msp\_T_0^1, msp\_T_0^2$, $nsp\_T_0^1, csp\_T_0$, $msp\_T_1^1, csp\_T_2^1$, $msp\_T_2^1, csp\_T_2^1$, $msp\_T_4^1, csp\_T_4^1$	msp_T, nsp_T, csp_T	Assign $false$ if there is conflict between msp_T and nsp_T.

MA_{11}: Monitor the satisfiability of the desired stop time at Stage 2. This action is to decide whether the previously defined stop time of the descent is still satisfactory when msp_T and nsp_T are updated.

Table 6.19: The trigger event, the guard condition and the duration of MA_{10}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{stage = Stage2\} \wedge \{sebStatus = true\}$	$stage$	EA_1
		$sebStatus$	EA_5
Duration	e_{10} must be short enough so that so that msp_T and nsp_T are generated before rta passes and msp_T expires.	NA	

Safety-critical scenarios. Defining this action according to the reference architecture reveals the safety-critical scenarios below.

SCS32: The update of msp_T and nsp_T may make the previously defined rta unsatisfactory.

SCS33: MA_{11} may take too long, so long that the descent stops before this action determines the satisfiability of rta.

SCS32 and SCS33 are similar to SCS22 and SCS23 respectively.

Defining the action. The input-output-transformation of MA_{11} is defined in Table.6.20, which addresses **SCS32** and **R33**. The trigger event, guard condition and the duration are defined in Table.6.21.

Table 6.20: The input-output-transformation of MA_{11} .

Input source	**Input**	**Output**	**Transformation**
MA_3/MA_{14}	The stop time rta	$sebStatus_4$	If $rta \subseteq nsp\_T, sebStatus_4 \leftarrow false$; If $rta \subseteq msp\_T \cup csp\_T$, $sebStatus_4 \leftarrow true$; If the decision cannot made before rta, $sebStatus_4 \leftarrow false$.
MA_{10}	msp_T, nsp_T, csp_T		

MA_{12}: Generate the performance constraints at Stage 2. This action updates the performance constraints on the defined descent trajectory in real time.

Table 6.21: The trigger event, the guard condition and the duration of MA_{11}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{sebStatus = true\} \wedge \{stage = Stage2\}$	$stage$	EA_1
		$sebStatus$	EA_5
Duration	NA	NA	

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS34: MA_{12} may take too long to update the performance constraints, so long that there will be no enough time for MA_{13} to generate msp_T before the rta passes and msp_T expires.

SCS34 is similar to SCS30.

Defining the action. MA_{12} is the similar to MA_9. The input-output-transformation is summarized in Table.6.22. According to the reference architecture, the trigger event, the guard condition and the duration of MA_{12} are summarized in Table.6.23. $\{dc = true\}$ in the guard condition is the implementation of msp_0^3 that is specific to this case study. Furthermore, the e_{12} must be short enough so that there will be enough time for MA_{13} to generate msp_T before the rta passes and before msp_T expires. This addresses **SCS34**. However, there is always a chance that e_{12} is too long that MA_{13} eventually generates msp_T after rta passes or msp_T expires. Therefore, a long enough look-ahead time must be selected to decide msp_T early to lower the likelihood of an expired msp_T or a passed rta to an acceptable level.

MA_{13}: Generate the must-stop time window at Stage 2. This action is to generate the must-stop time window as a potential deadline for MA_{14} to generate the new descent trajectory.

Safety-critical scenarios. Defining this action according to the reference ar-

Table 6.22: The input-output-transformation of MA_{12}.

Input source	**Input**	**output**	**Transformation**
Weather service provider	Headwind $\widehat{w}(t)$	pc_0^1	Generate/update the shape of the green cone.
The vehicle	The weight of payload pl.		
MA_{12}	pc_0^2		
Higher level of control	Point B and rta	pc_0^2	Generate/update Point B and rta.
The traffic surveillance	Vehicle type of the traffic vt	pc_1	Generate/update the minimal distance from the traffic.
The vehicle	The battery level of the vehicle bl		
NA	NA	pc_2	Pre-defined
Terrain map provider	The ground habitat gh	pc_3	Generate/update the minimal distance from the terrain.
Airspace info provider	The category of the adjacent airspace cat	pc_4	Generate/update the minimal distance from the airspace boundary.

Table 6.23: The trigger event, the guard condition and the duration of MA_{12}

	Content	**Info Source**	
Trigger	$\{sebStatus : true \rightarrow false\}$	$sebStatus$	EA_5
Guard	$\{stage = Stage2\} \wedge \{sebStatus = false\}$ $\wedge\{dc = true\}$	$stage$	EA_1
		$sebStatus$	EA_5
		dc	Vertiport
Duration	e_{12} must be short enough so that MA_{13} can generate msp_T before rta passes and msp_T expires.	NA	

chitecture reveals the following safety-critical scenarios.

SCS35: MA_{13} may take too long to update the performance constraints, so long that msp_T is generated after the rta passes and msp_T expires.

SCS35 is similar to SCS34.

Defining the action. The input-output-transformation is summarized in Ta-

ble.6.24, and the trigger event, the guard condition and the duration are summarized in Table.6.25. Note that the e_{13} must be short enough so that msp_T will be generated before the rta passes and before msp_T expires. This addresses **SCS35**. However, there is always a chance that e_{13} is too long that msp_T is generated eventually after rta passes or msp_T expires. Therefore, a long enough look-ahead time must be selected to decide msp_T early to lower the likelihood of an expired msp_T or a passed rta to an acceptable level.

Table 6.24: The input-output-transformation of MA_{13}. Note that msp_0^3 is not included in the table because it has been addressed as the guard condition of MA_9.

Input source	**Input**	**Output**	**Transformation**
MA_{12}	pc_0^1	$msp\_T_0^1$	Generate the time window for the must-stop condition w.r.t. the green cone.
MA_3/MA_{14}	$s(T)$		
MA_{12}	pc_0^2	$msp\_T_0^2$	Generate the time window for the must-stop condition w.r.t. rta.
MA_3/MA_{14}	$s(T)$		
MA_{12}	pc_1	$msp\_T_1^1$	Generate the time window for the must-stop condition w.r.t. the traffic.
MA_3/MA_{14}	$s(T)$		
Traffic surveillance provider	The projected traffic trajectory $\widehat{s}_{tf}$, the projected traffic speed $\widehat{v}_{tf}$.		
MA_{12}	pc_2	$msp\_T_2^1$	Generate the time window for the must-stop condition w.r.t. the weather.
MA_3/MA_{14}	$s(T)$		
The weather service provider	The projected weather impacted area $\widehat{wa}$		
MA_{12}	pc_4		

Input source	Input	Output	Transformation
MA_3/MA_{14}	$s(T)$	$msp\_T_4^1$	Generate the time window for the must-stop condition w.r.t. airspace boundary.
Airspace info provider	The projected airspace boundary $\widehat{ab}$		
MA_{13}	$msp\_T_0^1, msp\_T_0^2$, $msp\_T_1^1, msp\_T_2^1$, $msp\_T_4^1$	msp_T	Generate msp_T

Table 6.25: The trigger event, the guard condition and the duration of MA_{13}

	Content	Info Source	
Trigger	NA	NA	
Guard	$\{stage = Stage2\} \wedge \{sebStatus = false\}$	$stage$	EA_1
		$sebStatus$	EA_5
Duration	e_{13} must be short enough so that msp_T is generated before rta passes and msp_T expires.	NA	

MA_{14}: Generate the new descent trajectory at Stage 2. This action is to generate a new satisfactory descent trajectory before it is too late.

*Safety-critical scenarios.*Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS36: It can be impossible to find a new satisfactory descent trajectory due to the conflicts among the sub-hazards.

SCS37: MA_{14} may take too long to generate the desired descent trajectory, so long that there is not enough time left to actually start the new descent trajectory before msp_T expires and rta passes.

SCS38: It is possible that MA_{14} waits too long for the resolution of the conflict among the sub-hazards, so long that there is not enough time left to start the new descent trajectory before msp_T expires and rta passes.

SCS36, SCS37 and SCS38 are similar to SCS12, SCS14 and SCS15.

Defining the action. The input-output-transformation of MA_{14} is defined in Table.6.26, and the trigger event, the guard condition and the duration are summarized in Table.6.27. If **SCS36** happens, RfR_5 is sent to the higher level of control for a new descent instruction. If **SCS37** or **SCS38** happens, CM_5 is sent for the system to enter the contingency mode.

Table 6.26: The input-output-transformation of MA_{14}.

Input source	**Input**	**Output**	**Transformation**
MA_{13}	msp_T	$\{st < rta\}$ $\wedge\{st < \overline{msp\_T}\}$	Decide the start time st of the new descent trajectory.
MA_{14}	rta		
MA_{12}	pc_0^1, pc_0^2, pc_1, pc_2, pc_3, pc_4	$s(T) = \{s(t)\|t \in [st, rta]\}$ where $s(st)$ must be consistent with the old descent trajectory at st; Or RfR_5 if $s(T)$ cannot be found; Or CM_5 if $s(T)$ has not been found, while $\{t_c > \overline{mst\_T} - T1\}$ $\vee\{t_c > rta - T1\}$	Decide the new descent trajectory $s(T)$, where $T = [st, rta]$.
MA_{14}	$s(T)$		
Traffic surveillance provider	The projected traffic trajectory $\widehat{s_{tf}}$, the projected traffic speed $\widehat{v_{tf}}$.		
Weather service provider	The projected weather impacted area $\widehat{wa}$.		
Terrain map provider	The location of the terrain lt, and elevation of the terrain et.		
Airspace info provider	The projected airspace boundary $\widehat{ab}$		

Table 6.27: The trigger event, the guard condition and the duration of MA_{14}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{stage = Stage2\} \wedge \{sebStatus = false\}$	$stage$	EA_1
		$sebStatus$	EA_5
Duration	NA	NA	

6.4.3 Task 2: Derive the 4d waypoints from the desired 4d trajectory

6.4.3.1 The overall description

The desired descent trajectory $s(T)$ decided in Task 1 is shown in Fig.6.21. Any trajectory confined by the red boundaries is satisfactory, as long as the vehicle can descend to altitude h_1 between $[t_1, t_2]$ and to h_2 between $[t_3, t_4]$. Furthermore, the process model defined in Method 2 is denoted as $pm = \{Des, Vel, \gamma, v, w, s(t)\}$, where Des and Vel are the control inputs, w is the parameter, γ and v are the system state, and $s(t)$ is both the output and the system state.

The generate theme. According to the reference architecture, there are three themes for this task. The generate theme is to generate the control references for the controlled process. In this example, the control references are a set of 4d waypoints. To generate the control reference, the control algorithm needs to be defined first. It is possible that multiple control algorithms are defined for the same process model, however we only define one in this example as below:

$$\begin{cases} Des = \gamma = \arctan \frac{\widetilde{h} - h_r}{x_r - \widetilde{x}} \\ Vel = v = \frac{\sqrt{(x_r - \widetilde{x})^2 + (h_r - \widetilde{h})^2}}{t_r - t_c} + w \cdot cos\gamma \end{cases} \tag{6.1}$$

where $(\widetilde{x}, \widetilde{h}, t_c)$ is the current 4dt waypoint, and (x_r, h_r, t_r) is the target 4dt waypoint. Using this control algorithm, the vehicle flies straight to the target 4d waypoint with a constant speed.

Furthermore, according to the reference architecture, the control references can be generated in different ways, i.e. “single”, “all at once” and ‘one (batch) by one (batch)”, with the last one as the most general way to generate the control references. In this case study, we choose the most general way to generate the target 4dt waypoints one segment by one segment. Each segment is guided by one target 4dt waypoint.

Finally, the target waypoints (x_r, h_r, t_r) are to be achieved by Task 3, i.e. the control structure in Fig.6.22. We assume the total time delay introduced by the feedback loop is $T2$. According to the reference architecture, the time interval between two 4dt waypoints must be greater than $T1 - T2$, which is the time to generate the target waypoint based on the desired descent trajectory.

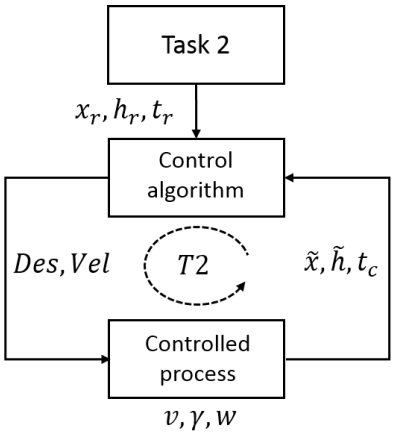


Figure 6.22: The control structure to achieve the target waypoints. The total time delay introduced by the feedback loop is assumed $T2$.

The predict theme The *predict theme* is to predict the evolutionary trajectory of the process model based on the selected waypoints. In this example, the dynamic trajectory of the process model is denoted as $\widehat{pm} = \{\widehat{Des}, \widehat{Vel}, \widehat{\gamma}, \widehat{v}, \widehat{w}(t), \widehat{s}(t)\}$.

The headwind as the parameter is first predicted $\widehat{w}(t)$. Second, given the target waypoint (x_r, h_r, t_r), the dynamic trajectory of the process can be predicted as below:

$$\begin{cases} \widehat{Des} = \widehat{\gamma} = \arctan \frac{\widetilde{h}-h_r}{x_r-\widetilde{\widetilde{x}}} \\ \widehat{Vel} = \widehat{v} = \frac{\sqrt{(x_r-\widetilde{x})^2+(h_r-\widetilde{h})^2}}{t_r-t_c} + \widehat{w}(t) \cdot cos\widehat{\gamma} \\ \widehat{s}(t) = (\widetilde{x} + \widehat{v} \cdot t \cdot cos(\widehat{\gamma}), \widetilde{h} - \widehat{v} \cdot t \cdot sin(\widehat{\gamma})) \end{cases}$$

The monitor theme According to the reference architecture, a process model has a set of constraints of the process model (EC) and a set of assumptions of the process model (IC). As long as IC is satisfied (i.e. $ic \in IC$), the process model is subject to EC. Furthermore, there can be a mapping between IC and EC, meaning different IC may lead to different EC. However, as previously defined, we only consider a specific set of IC and EC. In other words, if IC is not satisfied ($ic \notin IC$), then EC is invalid, which in turn makes the process model invalid.

The *monitor theme* checks three "ilities":

- Validity: whether the process model is in real time applicable to the controlled process, i.e. $pm \in EC$. To make sure EC is valid, $ic \in IC$ needs to be monitored in real time.
- Feasibility: whether the evolution of the controlled process will make the process model inapplicable, i.e. $\widehat{pm} \in EC$. To make sure EC is valid, $\widehat{ic} \in IC$ must be predicted for the time span under study.
- Satisfiability: whether the resulting descent trajectory will be bounded by the red boundaries, i.e. $\widehat{s}(T) \in s(T)$.

We use Fig.6.23 to explain these three concepts. The red boundary is the intended descent trajectory derived in Method 1 with respect to the hazard of "inadequate altitude". Furthermore, in Method 2, it is defined that the vehicle must maintain a distance from the ground in order to minimize the disturbance

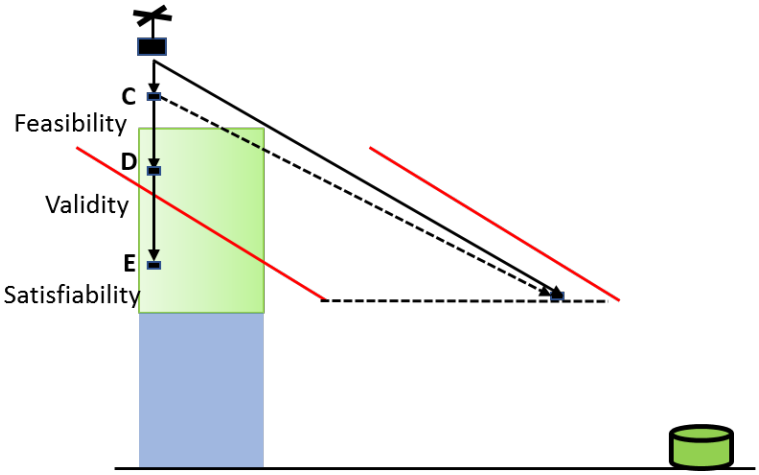


Figure 6.23: An example to explain validity, feasibility and satisfiability.

to the ground habitat. In the process model, such distance is defined as the constraint of the process model of the process model. In the figure, such distance is represented by the green box above the blue box (which represents the terrain). Originally, the vehicle is descending following the solid arrow within the red boundaries. However, for some reason, the vehicle suddenly drops altitude. If it drops to Point C and still tries to reach the original waypoint, because the dotted trajectory overlaps with the green box, such trajectory is *infeasible* as it will violate the constraint of the process model of the process model. As a solution, the controller must select a new waypoint before the vehicle enters the green box. If the vehicle drops to point D, because it is already within the green box, meaning the constraint of the process model is already violated, therefore the process model is *invalid*. If the vehicle drops to point E, because it is outside the red boundary, the output behavior is *unsatisfactory*. As a result, safety cannot be enforced.

6.4.3.2 The enabling action

EA_6**: Predicting the parameter.** This action is to predict the headwind during the desired descent time period. Such information is needed for all three themes.

- Trigger event: None.
- Guard condition: $stage = Stage1 \vee Stage2$.
- Input: The headwind prediction from the a service provider $\widehat{w}(t)$, and the desired descent trajectory $s(T)$.
- Output: $\widehat{w}(t)$ is the predicted headwind during the desired descent time period.
- Transformation: $\widehat{w}(t) \leftarrow \{\widehat{w}(t)|t \in T\}$.

As a result, EA_6 can be represented in Fig.6.24 graphically.

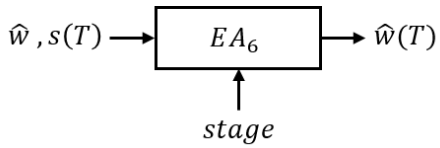


Figure 6.24: Graphical representation of EA_6 for the case study.

$\boldsymbol{EA_7}$**: Segmenting the SEB.** This action is to divide the desired descent trajectory into segments. The waypoints are then generated segment by segment.

- Trigger event: None.
- Guard condition: $stage = Stage1 \vee Stage2$.
- Input: $\widehat{w}(t)$ from EA_6 and $s(T)$.
- Output: The segmented prediction of the headwind $\widehat{w}(t)(\overline{T})$ and the segmented desired descent trajectory $s(\overline{T})$.
- Transformation: Diving the desired descent period into segments so that (1) the parameter in each segment is constant and (2) the time span of each segment is longer than $T1 - T2$.

Because the desired descent trajectory in Fig.6.21 already has three segments, descending from h_0 to h_1, from h_1 to h_2, and from h_2 to Point B. We divide the desired segment into the three segments so that (1) the time span of each segment is longer than $T1 - T2$ and (2) the headwind is not predicted to change in each segment.

As a result, EA_7 can be represented in Fig.6.25 graphically.

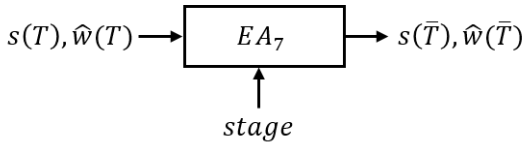


Figure 6.25: Graphical representation of EA_7 for the case study.

EA_8: Determining the scenario. This action reflects the scenario and t_d for the generate theme.

- Trigger event: NA.
- Guard condition: NA.
- Input: Sat and t_{d1} are the value and the associated deviation point of satisfiability; Fea and t_{d2} are the value and the associated deviation point of the feasibility. They are all outputs of the monitor theme.
- Output: $Scea$ is the specific scenario of the generate theme based on $\{Sat, Fea\}$; t_d is the associated deviation point that will go into the generate theme.
- Transformation:

$$\begin{cases} Scea = 1 \text{ and } t_d = false, if\ Sat = true \wedge Fea = true \\ Scea = 2 \text{ and } t_d = t_{d2}, if\ Sat = false \wedge Fea = true \\ Scea = 3 \text{ and } t_d = t_{d1}, if\ Sat = true \wedge Fea = false \\ Scea = 4 \text{ and } t_d = min(t_{d1}, t_{d2}), if\ Sat = false \wedge Fea = false \end{cases}$$

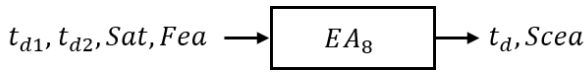


Figure 6.26: Graphical representation of EA_8 for the case study.

Graphically, Action EA_8 can be represented in Fig.6.26.

EA_9: Determining the priority. This action is to set the priority for the generate theme and the monitor theme when the inputs of predict theme or the monitor theme change.

- Trigger event: NA.
- Guard condition: NA.
- Input: $\{\{\widehat{\gamma}^*, \widehat{v}^*, \widehat{s}^*\}, \widehat{w}(t)(\overline{T}), \{\widetilde{\gamma}, \widetilde{v}, \widetilde{s}\}, \widetilde{w}\}$, and $\{s(T), IC\}$.
- Output: gen and mon.
- Transformation: If $\{\{\widehat{\gamma}^*, \widehat{v}^*, \widehat{s}^*\}, \widehat{w}(t)(\overline{T}), \{\widetilde{\gamma}, \widetilde{v}, \widetilde{s}\}, \widetilde{w}\}$ changes, $gen \leftarrow false$ and $mon \leftarrow false$; after the predict theme is updated, $mon \leftarrow true$; after the monitor theme is updated, $gen \leftarrow true$. If $\{s(T), IC\}$ changes, $gen \leftarrow false$; after the monitor theme is updated, $gen \leftarrow true$.

Graphically, Action EA_9 can be represented in Fig.6.27.

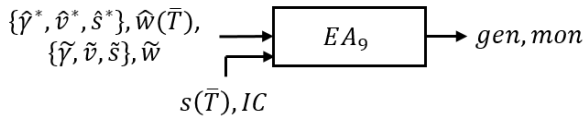


Figure 6.27: Graphical representation of EA_9 for the case study.

6.4.3.3 The main action

MA_{21}: Determining the steering point. This action is to determine the steering point $t_{d'}$ for Scenario 2, 3 and 4. Refer to the reference architecture for the definition of the steering point.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS39: The predicted trajectory or the process model may change while MA_{21} is deciding $t_{d'}$ based on the old prediction or process model. Such change may make t_d or $Scea$ become outdated already when $t_{d'}$ is being generated.

SCS40: MA_{21} may take too long to calculate the steering point, so long that there is less than $T1$ time before the $feasibility$ or $satisfiability$ is going to be violated.

Defining the action. The input-output-transformation of MA_{21} is defined in Table.6.28. If **SCS40** happens, RfR_7, CM_9 and CM_{10} are sent accordingly. According to the reference architecture, there is no trigger event for this action. The guard condition is summarized in Table.6.29, where $gen = true$ is to make sure when the prediction or the process model is being updated, this action is paused until all the updates are completed, which addresses **SCS39**.

Table 6.28: The input-output-transformation of MA_{21}.

Input source	**Input**	**Output**	**Transformation**
EA_8	$Scea, t_d$	$t_{d'}$, if it is generated before $t_d - T1$; otherwise, • Scenario 2: Send RfR_7 to get a new descent trajectory. • Scenario 3: Enter the contingency mode by issuing CM_9. • Scenario 4: Enter the contingency mode by issuing CM_{10}.	Decide the steering point.
		$Scea, t_d$	Pass $Scea, t_d$ from the input.

MA_{22}: Determining the 4dt waypoints. This action is to determine the 4dt waypoints segment by segment. Four 4dt waypoints are selected (Fig.6.28):

Table 6.29: The trigger event, the guard condition and the duration of MA_{21}

	Content	Info Source	
Trigger	NA	NA	
Guard	$\{stage = Stage1 \vee Stage2\} \wedge \{Scea = \{2,3,4\}\}$ $\wedge\{gen = true\}$	$stage$	EA_1
		$Scea$	EA_8
		gen	EA_9
Duration	NA	NA	

$(x_0, h_0, rta_0), (x_1, h_1, rta_1), (x_2, h_2, rta_2), (x_B, h_B, rta)$. The predicted descent trajectory is hence comprised of the lines between two adjacent waypoints. In this way, the resulting descent trajectory is confined by the red boundaries, where all hazards with respect to "inadequate altitude" can be avoided. Furthermore, as explained in the reference architecture, the duration of $T2$ and $rta_{i+1} - rta_i$ matters for the prediction. In this case study, we prescribe $T2 < rta_{i+1} - rta_i$. In other words, the waypoints must be selected in a way that $T2 < rta_{i+1} - rta_i$ always holds.

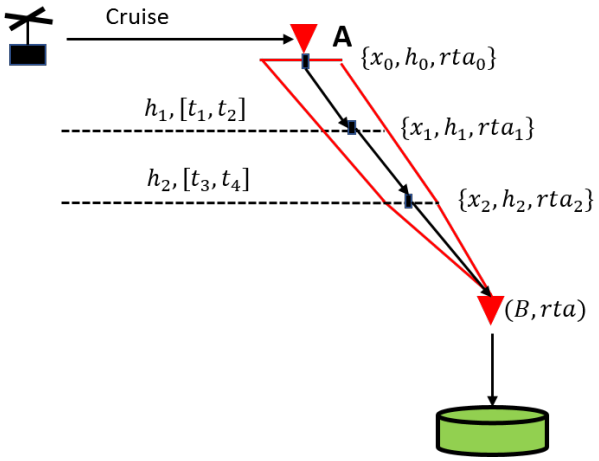


Figure 6.28: Four waypoints are picked $(x_0, h_0, rta_0), (x_1, h_1, rta_1), (x_2, h_2, rta_2)$, (x_B, h_B, rta) so that the resulting descent trajectory will be confined by the red boundaries.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS41: It is possible that a 4dt waypoint that is feasible for its own segment makes the next segment infeasible. For example, rta_2 is selected too

close to rta, which makes it impossible to descend to Point B on time.

SCS42: It is possible two target waypoints are too close in time, so close that there is not enough time to generate the next waypoint. For example, (x_1, h_1, rta_1) has to be decided before $rta_0 - T2$, and the controller must start generate (x_2, h_2, rta_2) before $rta_1 - T1$. If (x_1, h_1, rta_1) is generated at $rta_0 - T2$, then $rta_1 - T1$ must be after $rta_0 - T2$ (i.e. $T1 - T2 < rta_1 - rta_0$) because (x_2, h_2, rta_2) can only be generated after (x_1, h_1, rta_1) is generated.

SCS43: It is possible that the target waypoint is generated based on an outdated prediction of the descent trajectory. As shown in Fig.6.29, currently the vehicle is descending following the predicted trajectory to waypoint w1. At time t_1, the vehicle starts to generate the next waypoint. Before the next waypoint is generated, a wind gust blows the vehicle off of the original trajectory to waypoint w2. After that, at time t_2, the next waypoint w3 is generated. Because w3 is generated based on the original prediction of the descent trajectory, the descent angle to w3 is γ. However, the reality is that vehicle has to descend from w2 to w3, which yields a descent angle γ'. Obviously, $\gamma' > \gamma$. It is possible that γ' is greater than the upper bound of the acceptable descent angle, which leads to the vehicle into infeasibility for descent. Root cause of this problem is that, the generate theme is updated based on outdated output of the predict theme and/or the monitor theme.

SCS44: It is possible that the $\{Scea, t_d\}$ used to generate $t_{d'}$ becomes outdated when $t_{d'}$ reaches MA_{22}, because MA_{21} takes too long to generate $t_{d'}$.

SCS45: It is possible that the predicted cruise trajectory or velocity changes, which will lead to the change of the initial waypoint (x_0, h_0, rta_0) and all the waypoints after.

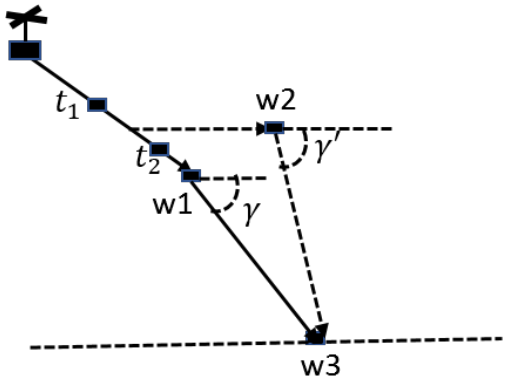


Figure 6.29: The target waypoint can be generated based on an outdated prediction of the descent trajectory.

SCS46: It is possible that the controller reacts too fast to the change of headwind prediction $\widehat{w}(t)$, the desired descent trajectory $s(T)$ or the predicted dynamic trajectory of the process model $\{\widehat{v}, \widehat{\gamma}, \widehat{s}(t)\}$. For example, when $\widehat{w}(t)$ changes, the controller needs to wait after $\{\widehat{v}, \widehat{\gamma}, \widehat{s}(t)\}$ is updated to update the target waypoints; when $s(T)$ or $\{\widehat{v}, \widehat{\gamma}, \widehat{s}(t)\}$ changes, the controller needs to wait after the $Scea$ is updated to update the target waypoints. Root cause of this possibility is that both the generate theme and the monitor theme take time, and the generate theme must wait until these two themes are properly updated to maintain the synchronization.

SCS47: It is possible that the target waypoint cannot be found.

SCS48: It is possible that the target waypoint is found too late.

Defining the action. The input-output-transformation of MA_{22} is defined in Table.6.30. **SCS41** is addressed by considering EC when generating the 4dt waypoint; **SCS42** is addressed by selecting a rta_{i+1} that makes $rta_{i+1} - rta_i > T1 - T2$; **SCS44** is addressed by Syn; **SCS45** is addressed by taking $\widehat{s}^i(t)$ and $\widehat{v}^i(t)$ as inputs; **SCS46** is addressed by taking $s(T)$ from MA_{29} and $(\widehat{v}_i, \widehat{\gamma}_i, \widehat{s}(T_i), \widehat{w}(t)_i)$ from MA_{28}, where $(\widehat{v}_i, \widehat{\gamma}_i, \widehat{s}(T_i), \widehat{w}(t)_i)$ is the prediction of $(v, \gamma, s(t), w)$ between rta_i and rta_{i+1}; **SCS47** and **SCS48** are addressed by sending out RfR_6, RfR_7, CM_8, CM_9 and CM_{10}. According to the reference architecture, the trigger event, the guard

condition and the duration of MA_{22} are summarized in Table.6.31. **SCS43** is addressed by the guard condition $gen = true$.

Table 6.30: The input-output-transformation of MA_{22}. For Scenario 2, 3 and 4, $(x_i, h_i) = \widehat{s}(t_{d'})$ and $rta_i = t_{d'}$.

Input source	**Input**	**Output**	**Transformation**
MA_{21}	$Scea, t_d$	Syn	$Syn \leftarrow true$ if $(Scea, t_d)$ from the two sources are the same, otherwise $Syn \leftarrow false$.
EA_8	$Scea, t_d$		
MA_{22}	Syn	$\widehat{s}(0), \widehat{v}(0), (x_{i+1}, h_{i+1}, rta_{i+1})$. If $(x_{i+1}, h_{i+1}, rta_{i+1})$ cannot be found, or is found too late, then send: • RfR_6 and CM_8 for Scenario 1; • RfR_7 for Scenario 2; • CM_9 for Scenario 3; • CM_{10} for Scenario 4.	If $Syn = false$, do nothing; otherwise generate the next 4dt waypoint $(x_{i+1}, h_{i+1}, rta_{i+1})$ so that $rta_{i+1} - rta_i > T2$.
MA_{28}	(x_i, h_i, rta_i)		
The controller for cruise	$\widehat{s}^i(t), \widehat{v}^i(t)$		
MA_{29}	$s(\overline{T})$		
MA_{21}	$t_{d'}$		
MA_{28}	$\widehat{v}_i, \widehat{\gamma}_i, \widehat{s}(T_i)$, $\widehat{w}(t), EC$		

Table 6.31: The trigger event, the guard condition and the duration of MA_{22}

	Content	**Info Source**	
Trigger	$s(\overline{T}) : false \rightarrow true$	$s(\overline{T})$	EA_7
Guard	$\{stage = Stage1 \vee Stage2\} \wedge \{gen = true\}$	$stage$	EA_1
		gen	EA_9
Duration	NA	NA	

MA_{23}: Predicting the system states and the output behavior. This action is to predict the system states and the output behavior so that the monitor theme can check track the “feasibility” and “satisfiability” in real time.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS49: It is possible that the predicted cruise trajectory or velocity changes, which may lead to the change of the predictions of descent trajectory.

SCS50: It is possible that MA_{23} takes too long to make a prediction.

SCS51: It is possible that the real states (i.e. $\widetilde{v}, \widetilde{s}$ and $\widetilde{\gamma}$) or parameter (i.e. $\widetilde{w}$) deviates from the predicted value, in which case a new prediction must be made.

SCS52: It is possible that the predicted initial condition (i.e. $\widehat{v}, \widehat{s}$ and $\widehat{\gamma}$) or parameter (i.e. $\widehat{w}(t)$) changes, in which case a new prediction must be made.

SCS49 and PR50 are straightforward. We are now explaining SCS51 and SCS52. As shown in Fig.6.30, the vehicle is currently descending to (x_i, h_i, rta_i), and the next waypoint is $(x_{i+1}, h_{i+1}, rta_{i+1})$, with the two solid arrows as the predicted trajectory. For example, if the predicted headwind $\widehat{w}(t)_{i+1}$ for the second segment changes, although the position $\widehat{s}(t)$ may not need to change, the prediction of the instructed velocity $\widehat{Vel}_{i+1}$ needs to change. The changed $\widehat{Vel}_{i+1}$ may lead to $|\widehat{Vel}_{i+1} - \widehat{v}_i| > 20$ knots, a violation of the constraint of the process model. This is an example of SCS52, where the prediction of the parameter changes.

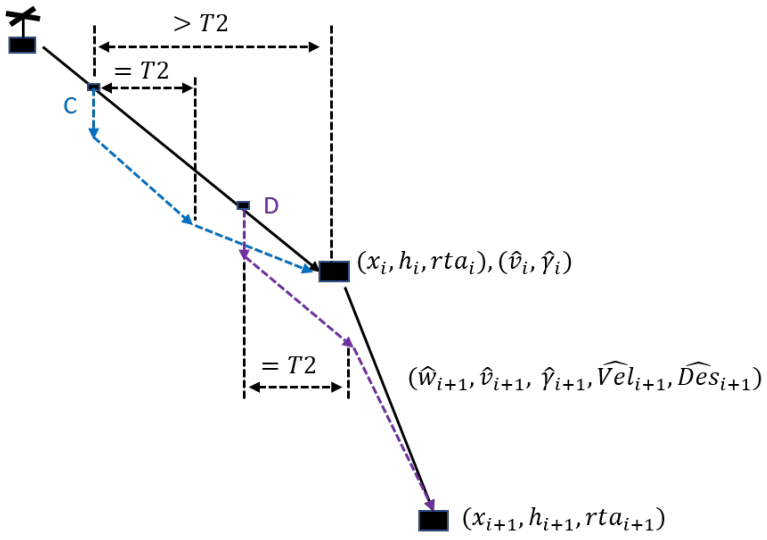


Figure 6.30: Both the change of the current prediction and the deviation from the current prediction requires new prediction.

Furthermore, another example is that when the vehicle drops altitude at Point C, which is deviation from the predicted trajectory. Because the vehicle needs

at least $T2$ to issue a new instruction, the vehicle continue executing the old instruction before the new instruction is received, which leads to the blue trajectory parallel to the first solid arrow. Assuming Point C is far enough from (x_i, h_i, rta_i) so that this waypoint still can be reached on time. The new predicted trajectory is the last segment of the blue arrow. However, the predicted velocity $\widehat{v}_i$ and the descent angle $\widehat{\gamma}_i$ are not the same. This changes the initial condition for the segment between rta_i and rta_{i+1}, which leads to an update of the prediction. This is an example of both SCS51 and SCS52.

Moreover, if the vehicle drops altitude at point D that is close to (x_i, h_i), (x_i, h_i, rta_i) cannot be reached any more. The vehicle first flies the purple trajectory parallel to the first solid arrow due to the $T2$ duration, and then flies directly to $(x_{i+1}, h_{i+1}, rta_{i+1})$ in the last segment of the purple arrow. This is also an example of both SCS51 and SCS52, but with a more obvious demonstration of the trajectory change.

Finally, the two segments (blue and purple) that are parallel to the first solid arrow are calculated based on the control inputs of the next task (i.e. $\widehat{Vel}_{i+1}$ and $\widehat{Des}_{i+1}$ in this case), they corresponds to the $\widehat{x}^*(T^*)$ of the reference architecture.

Defining the action. The input-output-transformation of MA_{23} is defined in Table.6.32. **SCS49** is addressed by taking $\widehat{s}(0)$ and $\widehat{v}(0)$ as the inputs; **SCS51** and **SCS52** are addressed by following $f_{24|25|26}$ of the reference architecture. According to the reference architecture, there is no trigger event for this action, and the guard condition is summarized in Table.6.33. **SCS50** is partially addressed by the requirement on the duration of this action.

MA_{24}: Predicting the control input. This action is to calculate the control input $(\widehat{Vel}_i, \widehat{Des}_i)$ (and $(\widehat{Vel}_{i+1}, \widehat{Des}_{i+1})$) based on the process model using the predicted system states $(\widehat{\gamma}_i, \widehat{v}_i)$ (and $(\widehat{\gamma}_{i+1}, \widehat{v}_{i+1})$) and the predicted parameter $\widehat{w}(t)$.

Safety-critical scenarios. Defining this action according to the reference ar-

Table 6.32: The input-output-transformation of MA_{23}.

Input source	**Input**	**Output**	**Transformation**
MA_{32}	$\widehat{s^*}(t), \widehat{r^*}, \widehat{v^*}$	$\widehat{s}(T_i), \widehat{s}(T_{i+1}), \widehat{\gamma}_i, \widehat{\gamma}_{i+1}, \widehat{v}_i, \widehat{v}_{i+1}, \widehat{w}(t), (x_i, h_i, rta_i), (x_{i+1}, h_{i+1}, rta_{i+1})$.	Predict the states and the output by following $f_{24\|25\|26}$ of the reference architecture, and update them when necessary.
Weather service provider	$\widehat{w}(t), \widetilde{w}$		
The vehicle	$\widetilde{s}(t), \widetilde{\gamma}, \widetilde{v}$		
MA_{22}	$(x_{i+1}, h_{i+1}, rta_{i+1})$ (x_i, h_i, rta_i), $\widehat{s}(0), \widehat{v}(0)$		

Table 6.33: The trigger event, the guard condition and the duration of MA_{23}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$stage = Stage1 \vee Stage2$	$stage$	EA_1
Duration	The duration of this action must be as short as possible. The first e_{23} time must be removed from the prediction.	NA	

chitecture reveals the following safety-critical scenarios.

SCS53: It is possible that MA_{24} takes too long to make the prediction.

Defining the action. The input-output-transformation of MA_{24} is defined in Table.6.34; the trigger event, the guard condition and the duration are summarized in Table.6.35. **SCS53** is partially addressed by the requirement on the duration of this action.

Table 6.34: The input-output-transformation of MA_{24}.

Input source	**Input**	**Output**	**Transformation**
MA_{23}	$\widehat{s}(T_i), \widehat{s}(T_{i+1}), \widehat{\gamma}_i, \widehat{\gamma}_{i+1}, \widehat{v}_i, \widehat{v}_{i+1}, \widehat{w}(t), (x_i, h_i, rta_i), (x_{i+1}, h_{i+1}, rta_{i+1})$.	$\widehat{Vel}_i, \widehat{Des}_i, \widehat{Vel}_{i+1}, \widehat{Des}_{i+1}, \widehat{s}(T_i), \widehat{s}(T_{i+1}), \widehat{\gamma}_i, \widehat{\gamma}_{i+1}, \widehat{v}_i, \widehat{v}_{i+1}, \widehat{w}(t), (x_i, h_i, rta_i), (x_{i+1}, h_{i+1}, rta_{i+1})$.	Predicting the instructed velocity and the descent angle.

Table 6.35: The trigger event, the guard condition and the duration of MA_{24}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$stage = Stage1 \vee Stage2$	$stage$	EA_1
Duration	The duration of this action must be as short as possible. The first e_{24} time must be removed from the prediction.	NA	

MA_{25}: Examining the assumptions of the process model at the current moment. This action is to check whether the assumptions of the process model of the process model defined in Method 2 are satisfied, to make sure the process model is valid for the descent at the current moment.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS54: The process model can become invalid due to the violation of the assumptions.

SCS55: It is possible that MA_{25} takes too long to decide the satisfiability of the assumptions.

Defining the action. The input-output-transformation of MA_{25} is defined in Table.6.36, which is entirely to address **SCS54**. The trigger event, the guard condition and the duration are summarized in Table.6.37. **SCS55** is partially addressed by the requirement on the duration of this action.

MA_{26}: Examining the constraints of the process model at the current moment. This action is to check whether the constraints of the process model of the process model defined in Method 2 are satisfied, to make sure the process model is valid for the descent at the current moment.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

Table 6.36: The input-output-transformation of MA_{25}.

Input source	**Input**	**Output**	**Transformation**
The vertiport	The current traffic congestion level, tcl	• The explicit constraints • CM_{11} if the assumptions of the process model are not satisfied.	Output the explicit constraints defined in Method 2, if all the assumptions of the process model are satisfied.
	The current descent procedure, dp		
	The current requirement on the traffic throughput, ttp		
	The current vertiport layout, vl		
The weather provider	The current weather condition, wc		
	The current operational status of the weather service provider, swp		
The vehicle	The current cross-wind, cw		
	The current battery level, bl		
	The current temperature, tem		
	The payload type, pt		
Terrain map provider	The ground elevation, ge		
	The type of ground habitat, gh		

Table 6.37: The trigger event, the guard condition and the duration of MA_{25}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$stage = Stage1 \vee Stage2$	$stage$	EA_1
Duration	The duration of this action must be as short as possible.	NA	

SCS56: The process model may become invalid because of the violation of the constraints.

SCS57: It is possible that the states/parameter changes while the satisfiability

of the constraints of the process model is being decided.

SCS58: It is possible that MA_{26} takes too long to decide the applicability of the process model.

Defining the action. The input-output-transformation of MA_{26} is defined in Table.6.38, which is entirely to address **SCS56** and **SCS57**. The trigger event, the guard condition and the duration are summarized in Table.6.39. **SCS58** is partially addressed by the requirement on the duration of this action.

Table 6.38: The input-output-transformation of MA_{26}.

Input source	**Input**	**Output**	**Transformation**
The vehicle	$\widetilde{v}, \widetilde{\gamma}, \widetilde{s}, \widetilde{w}$	CM_{12}	If $\{\widetilde{v}, \widetilde{\gamma}, \widetilde{s}, \widetilde{w}\} \notin EC$, then send CM_{12}.
MA_{25}	The constraints of the process model EC		

Table 6.39: The trigger event, the guard condition and the duration of MA_{26}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$stage = Stage1 \vee Stage2$	$stage$	EA_1
Duration	The duration of this action must be as short as possible.	NA	

MA_{27}: Examining the assumptions of the process model for the time period under study. This action is to check whether the assumptions of the process model will be satisfied for the time period under study. If the assumptions of the process model are predicted to be violated at some time point, the process model is only valid before that time.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS59: The process model can become invalid at some time point in the future due to the violation of the assumptions of the process model.

SCS60: It is possible that MA_{27} takes too long to decide the satisfiability of the assumptions of the process model.

Defining the action. The input-output-transformation of MA_{27} is defined in Table.6.40, which is entirely to address **SCS59**. The trigger event, the guard condition and the duration are summarized in Table.6.41. **SCS60** is partially addressed by the requirement on the duration of this action.

Table 6.40: The input-output-transformation of MA_{27}.

Input source	**Input**	**Output**	**Transformation**
The vertiport	The predicted traffic congestion level, $\widehat{tcl}$	• The explicit constraints EC. • CM_{13} if the assumptions of the process model will be violated.	Output the explicit constraints defined in Method 2, if all the assumptions of the process model will be satisfied during the time period under study.
	The planned change of the descent procedure, $\widehat{dp}$		
	The planned change of the requirement on the traffic throughput, $\widehat{ttp}$		
	The planned change of the vertiport layout, $\widehat{vl}$		
The weather service provider	The predicted weather condition, $\widehat{wc}$		
	The predicted operational status of the weather service provider, $\widehat{swp}$		
	The predicted cross-wind, $\widehat{cw}$		
	The predicted temperature, $\widehat{tem}$		
The vehicle	The predicted battery level, $\widehat{bl}$		

MA_{28}: Examining the constraints of the process model for the time period

Table 6.41: The trigger event, the guard condition and the duration of MA_{27}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$stage = Stage1 \vee Stage2$	$stage$	EA_1
Duration	The duration of this action must be as short as possible.	NA	

under study. This action is to check whether the constraints of the process model will be violated during the achievement of the control references, i.e. the "feasibility" of the control reference.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS61: The process model can become invalid at some time point in the future due to the violation of the constraints of the process model. For example, the headwind is predicted to become so strong that the descent is not feasible.

SCS62: It is possible that the predict theme needs update while MA_{28} is deciding the feasibility of the control reference.

SCS63: It is possible that MA_{28} takes too long to decide the feasibility of the control reference.

Defining the action. The input-output-transformation of MA_{28} is defined in Table.6.42, which is entirely to address **SCS61**. The trigger event, the guard condition and the duration are summarized in Table.6.43. **SCS62** is addressed by the guard condition $mon = true$, and **SCS63** is partially addressed by the requirement on the duration of this action.

MA_{29}: Examining the satisfiability of the predicted descent trajectory. This action is to check whether the predicted descent trajectory will satisfy the de-

Table 6.42: The input-output-transformation of MA_{28}.

Input source	**Input**	**Output**	**Transformation**
MA_{24}	$\widehat{Vel_i}, \widehat{Des_i}, \widehat{Vel_{i+1}}$, $\widehat{Des_{i+1}}, \widehat{s}(T_i)$, $\widehat{s}(T_{i+1}), \widehat{\gamma}_i, \widehat{\gamma}_{i+1}$, $\widehat{v}_i, \widehat{v}_{i+1}, \widehat{w}(t)$, (x_i, h_i, rta_i), $(x_{i+1}, h_{i+1}, rta_{i+1})$.	CM_{14} if $\widehat{w}(t)$ violates EC, otherwise Fea, EC, $t_{d1}, \widehat{s}(T_i), \widehat{s}(T_{i+1})$, $\widehat{\gamma}_i, \widehat{\gamma}_{i+1}, \widehat{v}_i, \widehat{v}_{i+1}, \widehat{w}(t)$, (x_i, h_i, rta_i), $(x_{i+1}, h_{i+1}, rta_{i+1})$	Check whether $\widehat{Vel_i}, \widehat{Des_i}$, $\widehat{Vel_{i+1}}, \widehat{Des_{i+1}}, \widehat{s}(T_i), \widehat{s}(T_{i+1})$, $\widehat{\gamma}_i, \widehat{\gamma}_{i+1}, \widehat{v}_i, \widehat{v}_{i+1}, \widehat{w}(t)$ satisfies EC.
MA_{27}	The constraints of the process model EC		

Table 6.43: The trigger event, the guard condition and the duration of MA_{28}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{stage = Stage1 \vee Stage2\} \wedge \{mon = true\}$	$stage$	EA_1
		mon	EA_9
Duration	The duration of this action must be as short as possible.	NA	

sired descent trajectory during the achievement of the control references, i.e. the "satisfiability" of the descent trajectory.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS64: The predicted descent trajectory can violate the desired descent trajectory at some time point in the future.

SCS65: It is possible that the predicted descent trajectory needs update while MA_{29} is deciding the satisfiability of the descent trajectory.

SCS66: It is possible that MA_{29} takes too long to decide the satisfiability of the descent trajectory.

Defining the action. The input-output-transformation of MA_{29} is defined in

Table.6.44, which is entirely to address **SCS64**. The trigger event, the guard condition and the duration are summarized in Table.6.45. **SCS65** is addressed by the guard condition $mon = true$, and **SCS66** is partially addressed by the requirement on the duration of this action.

Table 6.44: The input-output-transformation of MA_{29}.

Input source	**Input**	**Output**	**Transformation**
MA_{24}	$\widehat{s}(T_i), \widehat{s}(T_{i+1})$	$sat, t_{d2}, s(\overline{T})$	Deciding the satisfiability of the predicted descent trajectory.
EA_7	$s(\overline{T})$		

Table 6.45: The trigger event, the guard condition and the duration of MA_{29}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$\{stage = Stage1 \vee Stage2\} \wedge \{mon = true\}$	$stage$	EA_1
		mon	EA_9
Duration	The duration of this action must be as short as possible.	NA	

6.4.4 Task 3: Issue the instruction of speed and descent angle.

6.4.4.1 The overall description

Overall, Task 3 is to generate the control action based on the given control reference from Task 2. In this example, this task is to generate the instruction for the descent speed and the descent angle following (6.1). Furthermore, there are two delays in this task. One is the time delay introduced the control loop, which is $T2$ as previously explained. The other is the time for the control input to fully take effect at the output of the controlled process. We define such delay as $T3$ as shown in Fig.6.31. $T3$ implies that the controller cannot change the states of the controlled process within $T3$ from the current time stamp. $T2$ implies that the controller has to start calculating the control action at least $T2$ in advance in order to change the state of the controlled process at one time point.

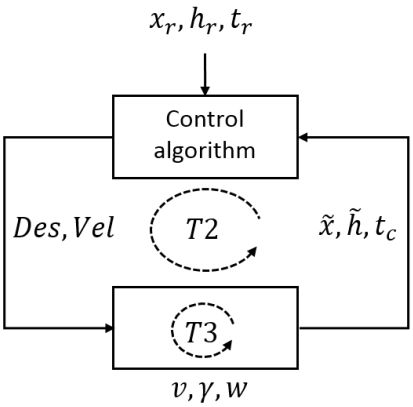


Figure 6.31: The time delay introduced by the controlled process is $T3$.

Because of $T2$ and $T3$, Task 3 has to achieve more than (6.1). We zoom in to the actions happening due to the time delays and use the following example in Fig.6.32 to explain the generate theme, the monitor theme and the predict theme of Task 3.

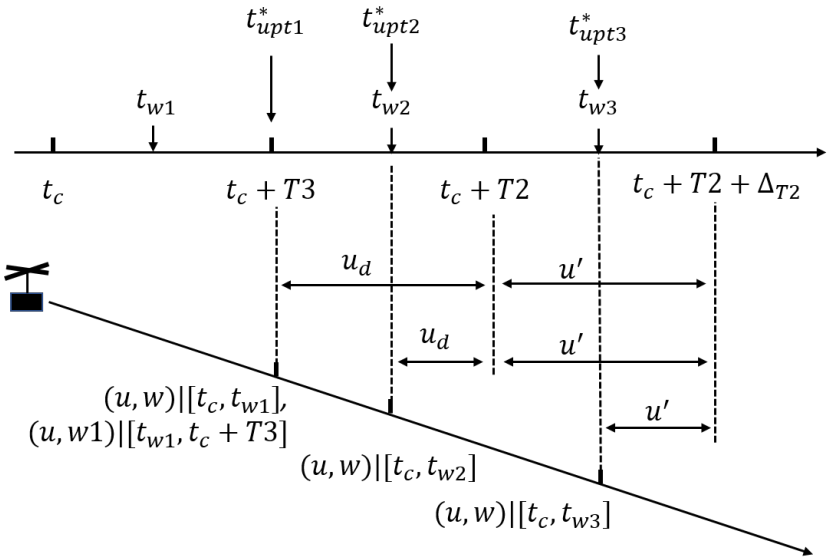


Figure 6.32: Zooming in to the time delay of $T2$ and $T3$ using the change of headwind as an example.

First, at the current time t_c, the controller is to generate the control action that is to take effect at $t_c + T2 + \Delta_{T2}$. This is a common practice, especially for a system that operates at a slower pace. For example, the air traffic controller usually issues the instruction a certain before the time that the instruction is supposed to be followed, so that the pilot can have enough time to execute the

instructions. As a result, the system states of $[t_c, t_c + T2 + \Delta_{T2}]$ are already determined by the control action decided at $[t_c - T2 - \Delta_{T2}, t_c]$.

However, if at the same time the headwind w is predicted to change at some point within $[t_c, t_c + T2 + \Delta_{T2}]$, the control actions that are already generated during $[t_c - T2 - \Delta_{T2}, t_c]$ must change accordingly. For example, the headwind could be predicted to change during $[t_c, t_c + T3]$ at t_{w1}. Because of the time delay $T3$, the earliest time to adjust the control action is at $t_c + T3$. New control action can only be given starting from $t^*_{upt1} = t_c + T3$, where the system states can be predicted by calculating the original control action u with the original headwind w for $[t_c, t_{w1}]$ (denoted as $(u, w)|[t_c, t_{w1}]$) and with the new headwind $w1$ for $[t_{w1}, t_c + T3]$ (denoted as $(u, w1)|[t_{w1}, t_c + T3]$). However, because it takes $T2$ for the controller is to generate a control action following the control algorithm, the controller can only issue default control action u_d for $[t_c + T3, t_c + T2]$, and hence the system states at $t_c + T2$ (denoted as $\widehat{x}(t_c + T2)$) can be predicted by calculating u_d and $w1$ for $[t_c + T3, t_c + T2]$. With $\widehat{x}(t_c + T2)$ and $w1$, new control actions u' can be generated for $[t_c + T2, t_c + T2 + \Delta_{T2}]$. In this example, the predict theme calculates the system states of $[t_c, t_c + T2 + \Delta_{T2}]$ with the original u and w before t_{w1} and $w1$ for $[t_{w1}, t_c + T2 + \Delta_{T2}]$; the monitor theme selects the time point that the control action needs to be updated, i.e. $t^*_{upt1} = t_c + T3$ in this example; the generate theme decides the control action to be issued after t^*_{upt1}, i.e. u_d for $[t_c + T3, t_c + T2]$ and u' for $[t_c + T2, t_c + T2 + \Delta_{T2}]$.

Similarly, if the change of the headwind happens at t_{w2} during $[t_c + T3, t_c + T2]$, the predict theme first calculates the system states of $[t_c, t_c + T2 + \Delta_{T2}]$ with the original u and w before t_{w2} and $w2$ during $[t_{w2}, t_c + T2 + \Delta_{T2}]$; the monitor theme selects t_{w2} to update the control action; the generate theme decides to issue u_d for $[t_{w2}, t_c + T2]$ and u' for $[t_c + T2, t_c + T2 + \Delta_{T2}]$.

Finally, if the change of the headwind happens at t_{w3} during $[t_c + T2, t_c + T2]$, the predict theme first calculates the system states of $[t_c, t_c + T2 + \Delta_{T2}]$ with the

original u and w before t_{w3} and $w3$ during $[t_{w3}, t_c + T2 + \Delta_{T2}]$; the monitor theme selects t_{w3} to update the control action; the generate theme decides to issue u' for $[t_c + T2, t_c + T2 + \Delta_{T2}]$.

6.4.4.2 The enabling action

Because the segments in this example are determined by the 4dt waypoints and the control input (i.e. the instruction) are supposed to be the same for each 4dt waypoints, EA_{10} and EA_{11} does not apply to this example. As a result, we only explain EA_{12}.

EA_{12}: Setting the priority. This action is to set the priority for the generate theme and the monitor theme when the inputs of predict theme or the monitor theme change.

- Trigger event: NA.
- Guard condition: NA.
- Input: $\{\widehat{w}(t)(\overline{T}), \{\widetilde{\gamma}, \widetilde{v}, \widetilde{s}\}, \widehat{s}^i(t), \widehat{v}^i(t)\}$, and (x_i, h_i, rta_i).
- Output: $gen1$ and $mon1$.
- Transformation: If $\{\widehat{w}(t)(\overline{T}), \{\widetilde{\gamma}, \widetilde{v}, \widetilde{s}\}, \widehat{s}^i(t), \widehat{v}^i(t)\}$ changes, $gen1 \leftarrow false$ and $mon1 \leftarrow false$; after the predict theme is updated, $mon1 \leftarrow true$; after the monitor theme is updated, $gen1 \leftarrow true$. If (x_i, h_i, rta_i) changes, $gen1 \leftarrow false$; after the monitor theme is updated, $gen1 \leftarrow true$.

Graphically, Action EA_9 can be represented in Fig.6.27.

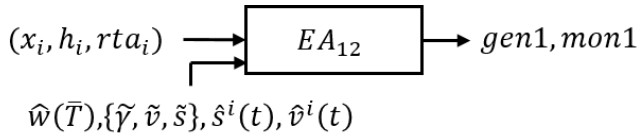


Figure 6.33: Graphical representation of EA_{12} for the case study.

6.4.4.3 *The main action*

MA_{30}: Generating the descent instruction. This action is to generate the control action, i.e. the instruction for descent speed and descent angle.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS67: The generated waypoint (x_i, h_i, rta_i), the predicted headwind $\widehat{w}(t)$, the predicted cruise trajectory and speed $\widehat{s}^i(t), \widehat{v}^i(t)$, the time point to update the control action T^*_{upt} and the predicted state states $x^*(T^*)$ may change, which requires a recalculation of the instruction for the descent speed and the descent angle.

SCS68: If $x^*(T^*)$ changes, it is possible that the instruction is updated before T^*_{upt} is updated accordingly first.

SCS69: If $\widehat{s}^i(t), \widehat{v}^i(t)$ or $\widehat{w}(t)$ changes, it is possible that the instruction is updated before the predict theme and the monitor theme is updated accordingly first.

SCS70: If the generated waypoint (x_i, h_i, rta_i) changes, it is possible that the instruction is updated before T^*_{upt} is updated accordingly first.

SCS71: MA_{30} can take too long, so long that the instruction cannot be updated from T^*_{upt}.

Defining the action. The input-output-transformation of MA_{30} is defined in Table.6.46, which is entirely to address **SCS67**. The fact that $\widehat{x}^*(T^*)$ comes from MA_{23} is to address **SCS68**. The trigger event, the guard condition and the duration are summarized in Table.6.47. **SCS69** and SCS70 are addressed by the guard condition $gen1 = true$, and **SCS71** is partially addressed by the requirement on the duration of this action.

Table 6.46: The input-output-transformation of MA_{30}.

Input source	**Input**	**Output**	**Transformation**
MA_{32}	The time to update the instruction T^*_{upt}	The descent instruction $u(T^*)$	Calculating the descent instruction.
	The predicted system states $\widehat{s}^*(t), \widehat{r}^*, \widehat{v}^*$		
MA_{22}	The generated waypoint (x_i, h_i, rta_i)		
The weather service provider	The predicted headwind $\widehat{w}(t)$		
Controller for cruise	The predicted trajectory $\widehat{s}^i(t)$		
	The predicted speed $\widehat{v}^i(t)$		

Table 6.47: The trigger event, the guard condition and the duration of MA_{30}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$gen1 = true$	$gen1$	EA_{12}
Duration	The duration of this action must be as short as possible.	NA	

MA_{31}: Issuing the instruction. This action is to issue the default control action when MA_{30} cannot find the instruction from $t_c + T3$ and after.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS72: It is possible that the instruction cannot be found or cannot be found in time.

Defining the action. The input-output-transformation of MA_{31} is defined in Table.6.48, which is entirely to address **SCS72**. No trigger event or the guard condition are defined for this action, and this action is considered instantaneous.

Table 6.48: The input-output-transformation of MA_{31}.

Input source	Input	Output	Transformation
MA_{30}	$u(T^*)$	$u(T^*)$	When $u(t_c + T3) = false$, then $u(t_c + T3) \leftarrow true$.

MA_{32}: Predicting the system states. This action is to predict the system states based on the previously issued instructions. Note that the prediction made by this action is different from the predication in Task 2 in that this action makes prediction based on the previously issued instruction, while the prediction in Task 2 is based on the prediction of future instruction. As a result, the prediction made by this action is on a shorter time scale. In fact, the prediction made in this action is one of the inputs for the prediction of Task 2 (see MA_{23}).

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS73: The predicted headwind $\widehat{w}(t)$, the predicted cruise trajectory and speed $\widehat{s}^i(t), \widehat{v}^i(t)$ and the descent instruction can change, which requires a re-calculation of the prediction of the system states.

SCS74: It is possible that the current real states $(\widetilde{s}, \widetilde{v}, \widetilde{\gamma})$ deviates from the previous prediction.

SCS75: It is possible that this action takes too long.

Defining the action. The input-output-transformation of MA_{32} is defined in Table.6.49, which is entirely to address **SCS73** and **SCS74**. The trigger event, the guard condition and the duration are summarized in Table.6.50. **SCS75** is partially addressed by the requirement on the duration of this action.

MA_{33}: Monitoring the issued descent instruction. This action is to monitor the issued descent instructions in case they need to be updated, and decide from

Table 6.49: The input-output-transformation of MA_{32}.

Input source	**Input**	**Output**	**Transformation**
The vehicle	The current states $\widetilde{s}, \widetilde{\gamma}, \widetilde{v}$	The predicted system states $\widehat{s}^*(t), \widehat{r}^*, \widehat{v}^*$	Calculating the predicted system states.
MA_{31}	The descent instruction $u(T^*)$		
The weather service provider	The predicted headwind $\widehat{w}(t)$		
The controller for cruise	The predicted trajectory $\widehat{s}^i(t)$		
	The predicted speed $\widehat{v}^i(t)$		

Table 6.50: The trigger event, the guard condition and the duration of MA_{32}

	Content	**Info Source**
Trigger	NA	NA
Guard	NA	NA
Duration	The duration of this action must be as short as possible. The first e_{24} time must be removed from the prediction.	NA

which time point to update.

Safety-critical scenarios. Defining this action according to the reference architecture reveals the following safety-critical scenarios.

SCS76: The generated waypoint (x_i, h_i, rta_i), the predicted headwind $\widehat{w}(t)$, the predicted cruise trajectory and speed $\widehat{s}^i(t), \widehat{v}^i(t)$, and the predicted state states $x^*(T^*)$ may change, which can require an update of the previously issued instruction.

SCS77: If $\widehat{s}^i(t), \widehat{v}^i(t)$ or $\widehat{w}(t)$ changes, it is possible that the time point to update is decided before the new $x^*(T^*)$ is generated.

SCS78: MA_{33} can take too long, so long that the instruction cannot be updated for T^*_{upt}.

Defining the action. The input-output-transformation of MA_{33} is defined in Table.6.51, which is entirely to address **SCS76**. The trigger event, the guard condition and the duration are summarized in Table.6.52. **SCS77** is addressed by the guard condition $mon1 = true$, and **SCS78** is partially addressed by the requirement on the duration of this action.

Table 6.51: The input-output-transformation of MA_{33}.

Input source	**Input**	**Output**	**Transformation**
MA_{32}	The predicted system states $\widehat{s}^*(t), \widehat{r}^*, \widehat{v}^*$	The time point T^*_{upt} to update the instruction, and $\widehat{s}^*(t), \widehat{r}^*, \widehat{v}^*$ passed from MA_{32}.	Calculating T^*_{upt}.
MA_{22}	The generated waypoint (x_i, h_i, rta_i)		
The weather service provider	The predicted headwind $\widehat{w}(t)$		
The controller for cruise	The predicted trajectory $\widehat{s}^i(t)$		
	The predicted speed $\widehat{v}^i(t)$		

Table 6.52: The trigger event, the guard condition and the duration of MA_{33}

	Content	**Info Source**	
Trigger	NA	NA	
Guard	$mon1 = true$	$gen1$	EA_{12}
Duration	The duration of this action must be as short as possible.	NA	

6.5 Summary

In this case study, we applied the reference architecture to the design of a descent function guided by the hazard of "inadequate altitude". As a result, we identified 78 safety-critical scenarios that need to be addressed adequately otherwise may lead to hazardous situation. Furthermore, a functional architecture (Fig.6.34) of the controller is developed based on the reference reference, to

achieve the descent goal and avoid the 78 safety-critical scenarios developing into hazardous situation. The full readable N^2 diagram can be found in Appendix E.

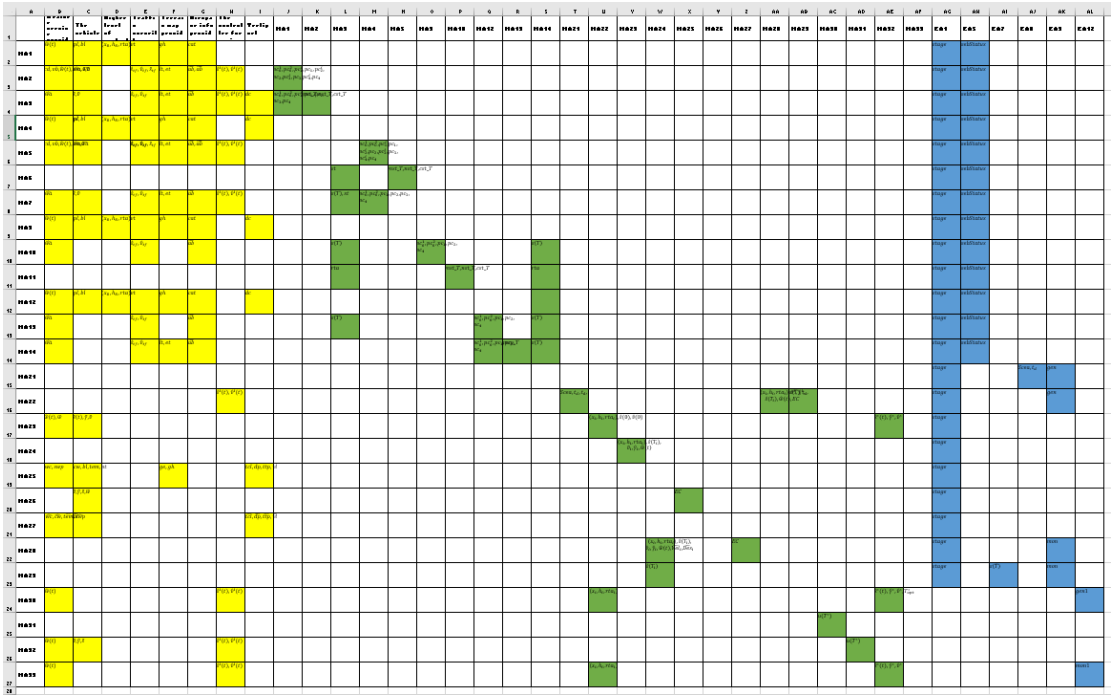


Figure 6.34: Applying the reference architecture to the case study result in the N^2 diagram, where the yellow cell represents the interaction between the main actions and the environment and the controlled process, the green cell represents the interaction among the main actions, and the blue cell represents the interactions between the main actions and the enabling actions. Refer to Appendix E for the full readable N^2 diagram.

Chapter 7

Conclusion

7.1 Summary

Model-based Safety Assessment (MBSA) has gained tremendous traction for the past two decades in the safety community. However, to apply MBSA for certification, one still needs to demonstrate that the model used in MBSA includes all the possible safety-critical scenarios that the actual system will encounter in the actual operation. The current MBSA approaches are not equipped for this task.

To tackle this problem, we propose a new safety-guided design methodology (called STPA+) to complement MBSA for safety assurance. STPA+ treats a system as a control structure, and is comprised of three methods. Method 1 derives safety constraints from the hazard to make sure the safety constraints adequately reflect the hazard under study; Method 2 defines the model of controlled process to make sure the model is properly constrained both explicitly and implicitly; Method 3 defines a safe controller by providing a reference architecture to make sure the controller defined based on the reference architecture has all the safety-critical scenarios (without failure) addressed.

Finally, Method 1 and Method 2 are the requirement analysis step, and Method 3 is the functional analysis in the context of Systems Engineering (Fig.7.1). Usually, safety assessment (MBSA in this case) takes places after the initial functional architecture is determined. Fig.7.1 shows that STPA+ is accomplished right before MBSA is supposed to start, which implies that STPA+ and MBSA also fit together temporally. Together, "STPA+MBSA" provides a strong and comprehensive argument for safety assurance.

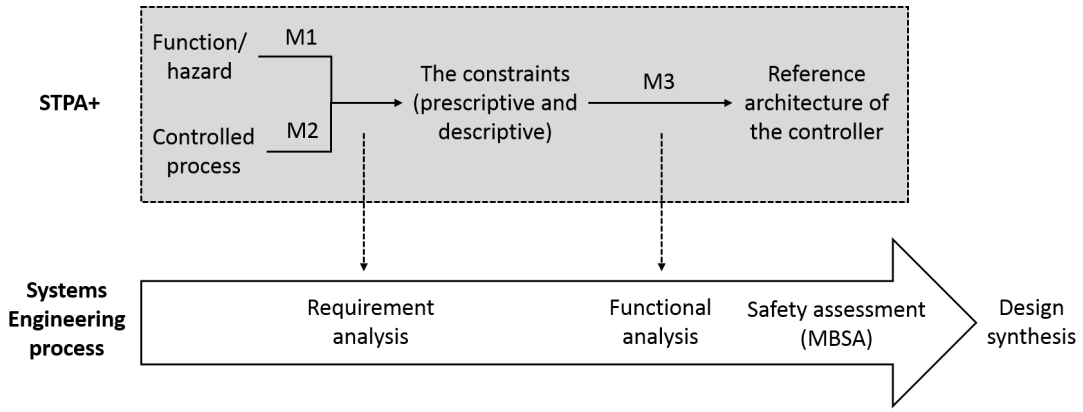


Figure 7.1: STPA+ in the context of the Systems Engineering process. M1, M2 and M3 mean Method 1, 2 and 3.

7.2 Contribution

In general, this dissertation identifies a gap of MBSA, which is that most MBSA approaches take the "model" as given. If certain safety-critical scenarios are not included or considered in the given design solution (the left shoulder of Fig.1.7), or the properties do not correctly reflect the hazard (the right shoulder of Fig.1.7), the results of MBSA cannot be fully trusted for safety assurance.

Specifically, we claim four contributions for this dissertation.

- Overall, this dissertation develops a safety-guided design methodology STPA+ to identify and address the safety-critical scenarios without component-

level failure for the definition of the design solution. Compared with the original STPA, STPA+ provides better methodological support in refining the hazardous scenarios associated with control into the safety-critical scenarios that the design solution can directly act upon. It is a contribution to identifying and addressing “hazards without failure”.

- Method 1 converts the logic of the STPA “unsafe control action” to “what is safe” to make sure the safety constraints derived from the hazard adequately reflect the hazard under study. The method provided to define the constraints on the start and stop times is a refined way to identify the “context” associated with the original STPA “unsafe control action”. Compared with the guide-words-based traditional hazard identification techniques (e.g., HAZOP), Method 1 has a more precise and more concrete model that explains why a specific intended output behavior starts/stops too early/late, which hence is also a contribution to the general hazard identification literature.

- Method 2 defines the model of the controlled process to make sure the model is properly constrained both explicitly and implicitly. It focuses on the definition of the boundary conditions (i.e., operational envelop) and the assumptions of the model of the controlled process, which is a contribution to both the safety community and the general model-based design community.

- Method 3 defines a safe controller by providing a reference architecture to ensure the controller defined based on the reference architecture has all the safety-critical scenarios (without component-level failure) addressed. It provides detailed support to identify the design errors of a controller and address them from the beginning with a reference architecture. To the best of our knowledge, there are no such works in the current literature.

In summary, Method 1 strengthens the right shoulder of Fig.1.7; Method 2&3

strengthens the left shoulder of Fig.1.7. Together, "STPA+MBSA" provides a strong and comprehensive argument for the safety assurance of a cyber-physical human system.

7.3 Future work

We have identified three future directions for STPA+.

First, STPA+ is developed to complement MBSA. The specifications resulting from STPA+ are organized in the construct of {input, output, transformation, trigger, guard}, which is very common in many MBSA languages. The next step is to translate this construct to one or more mainstream MBSA languages. In this way, the result of STPA+, a design solution, can be directly translated into a model that is ready for MBSA analysis. Because the problem of missing safety-critical scenarios due to non-failure causal factors has already been addressed by STPA+, the safety engineers can entirely focus on modeling failures and conducting the MBSA analysis with the tool supports available to the specific languages. This is the solution to Challenge D explained on Page 29.

Second, STPA+ focuses on addressing the safety-critical scenarios that do not involve failures/faults. There is an ISO standard under development, "ISO/PAS 21448: Road vehicles — Safety of the intended functionality (SOTIF)". This standard is about minimizing the hazardous scenarios caused by non-failure factors. STPA+ can be used precisely to demonstrate the compliance of SOTIF.

Finally, STPA+ is developed initially to insist on defining the design solution from the beginning. However, it is also possible that STPA+ cannot be applied when the system is built, for example, a legacy system or an AI-based system. Because STPA+ does not need an existing design solution to identify the safety-critical scenarios, STPA+ can also be used to design a run-time safety monitor as a safety enhancement to these systems by identifying critical scenarios and

providing run-time alerts, and intervening when the human/AI controller fails to act.

Appendix A: Terms

We acknowledge the fact that some terms used in this dissertation are overloaded. They can mean different things in different contexts in the general Systems Engineering and System Safety community. To avoid confusion, we define these terms for this dissertation and this dissertation only. Obviously, the Systems Engineering community needs to agree on unified definitions of these terms, but this is by no means our intention here.

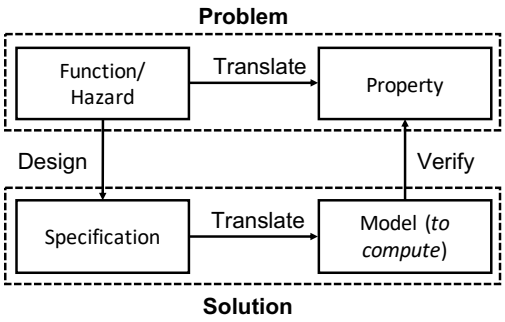


Figure 7.2: A general engineering design process to define (and distinguish) some of the key terms used in the dissertation.

Model: In general, we defined (in the attached MBSA report) a spectrum of models in the model-based world. At one end is "model-to-describe" (such as engineering drawing, SysML model, etc.), to describe the design solution to the problem; on the other end is "model-to-compute" (such as a state-space representation), while is also a representation of the design solution, mainly to verify whether the design solution indeed solves the problem through computation. Most modeling language positions somewhere between the two ends, having different capabilities in expression and computation. However, in this dissertation, "model" specifically refers to "model-to-compute" or the computation capability of the model.

Specification: Specifications are a representation of the design solution to achieve to given functional goal and/or avoid hazard (Fig.7.2). Although speci-

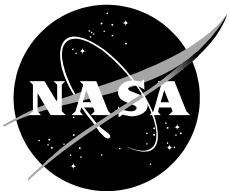
fications can also be organized in the form of a model, they are mainly used to *describe* the intended design solution. In this sense, specification is the "model-to-describe". The translation from the specification (or "model-to-describe") to the model (-to-compute) often loses information.

Property: It is a representation of the given functional goal and hazard particularly for verification (Fig.7.2). In Model Checking, they are called liveness property for the functional goal and safety property for the hazard.

Requirement: Requirement and specification are usually used interchangeably in the Systems Engineering community. But in this dissertation, requirements are specifically referred to the design activities that need to be accomplished according to STPA+.

Appendix B: MBSA review

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 STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

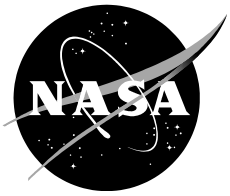
For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- Help desk contact information:

https://www.sti.nasa.gov/sti-contact-form/
and select the "General" help request type.

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.

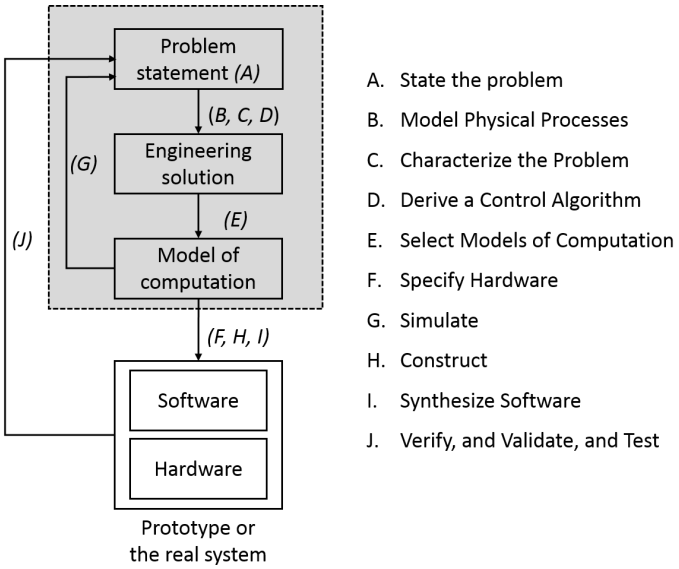


Figure 1. The MBD process. The arrows on the left can be mapped to the steps to the right (adopted from ref. [106]). The inner loop (shaded area) is the model-based analysis.

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.

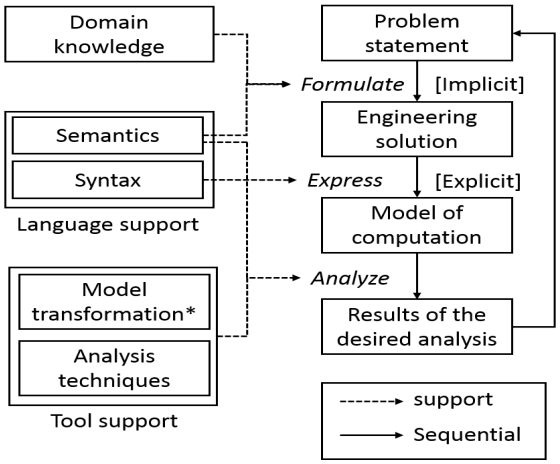


Figure 2. The MBA process, where the artifacts on the right-hand side are adopted from the inner loop of figure 1 and the left-hand side are the supports for the respective activities.

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.*

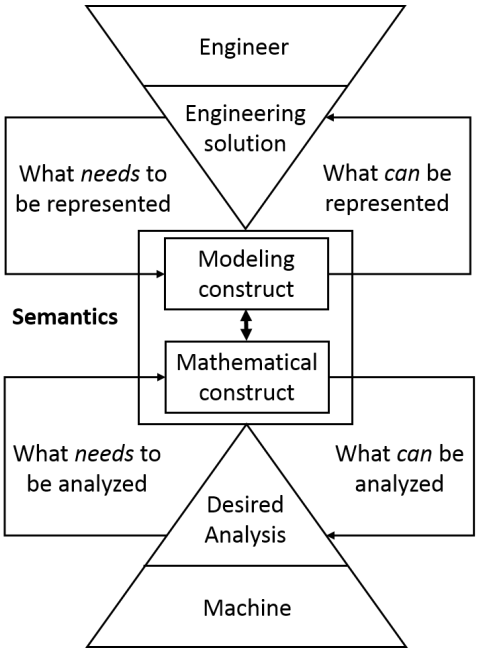


Figure 3. The semantics of the modeling language are at the center of an MBA process in the view of a methodology developer.

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.

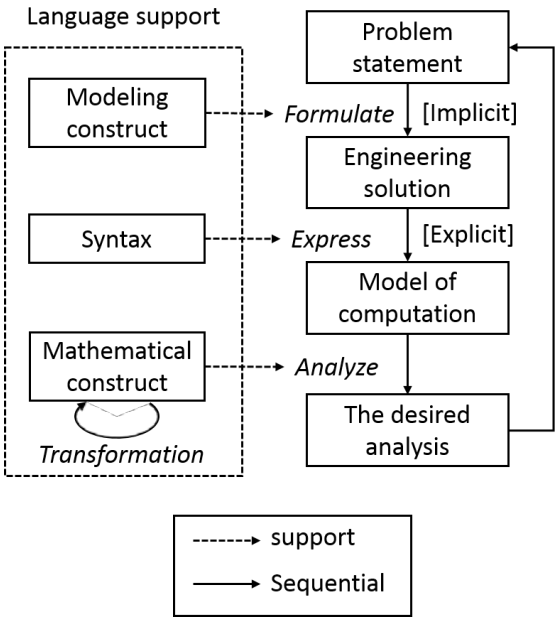


Figure 4. The MBA process and the language support. While the analytical process represents a series of (iterative) steps [right], it is supported by particular choices with respect to the modeling language [left]. The discussion of MBSA will follow closely with this process.

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.

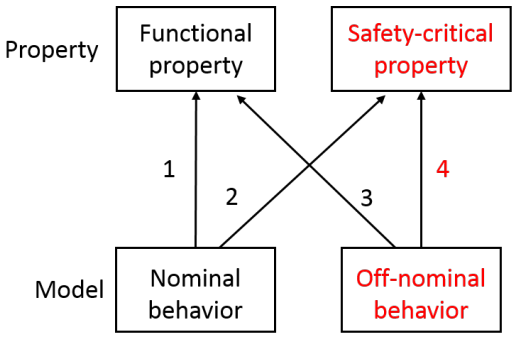


Figure 5. Four types of inductive safety analysis. The start of the arrow is verified against the end of the arrow. No. 4 is one of the defining features of MBSA.

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.

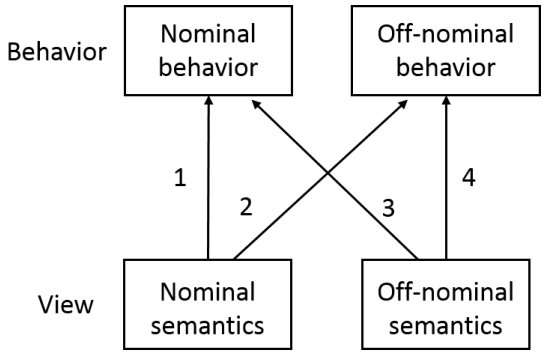


Figure 6. Nominal behaviors can be represented in both the nominal and the off-nominal views. Same for the off-nominal behavior. The arrow means "represent" in this figure.

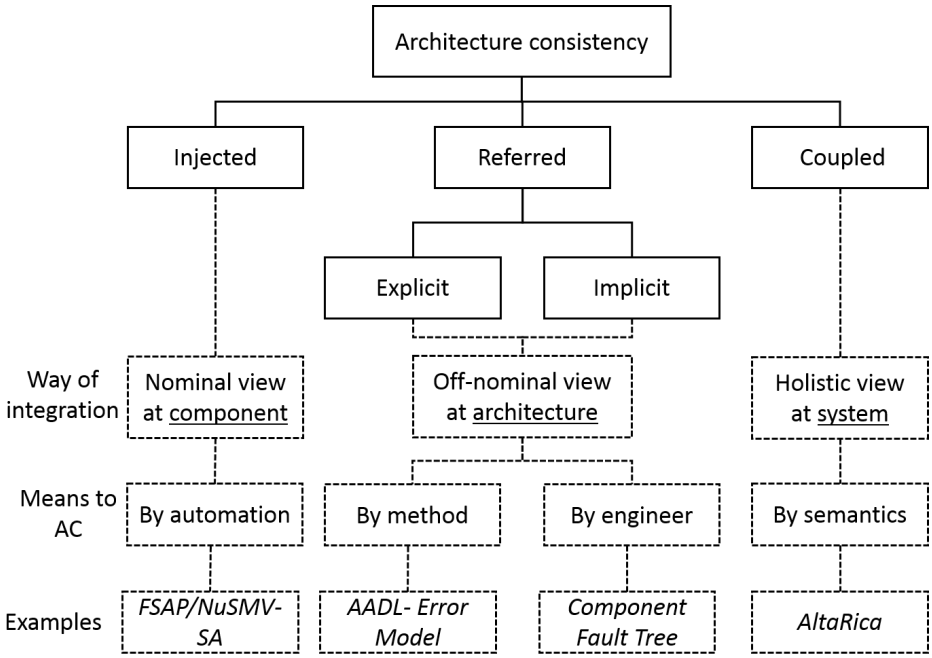


Figure 7. Notable patterns to achieve Architecture Consistency.

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

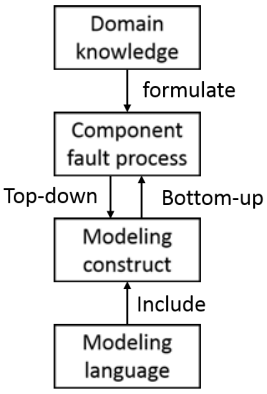


Figure 8. Formulating the component fault process top down and bottom up.

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).

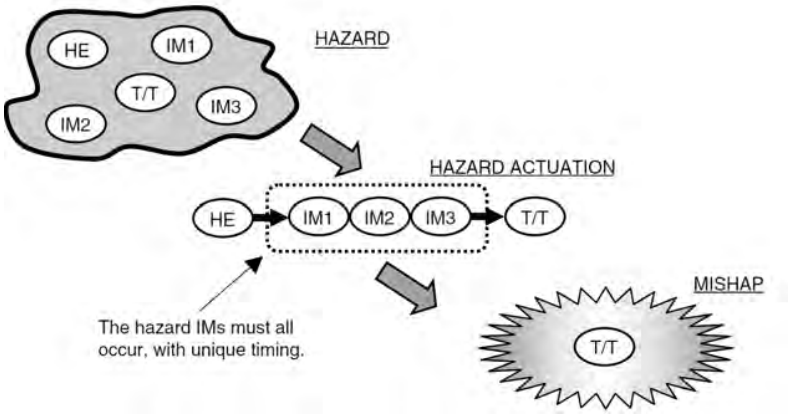


Figure 9. The hazard actuation (adopted from figure 2.5 of [123]).

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).

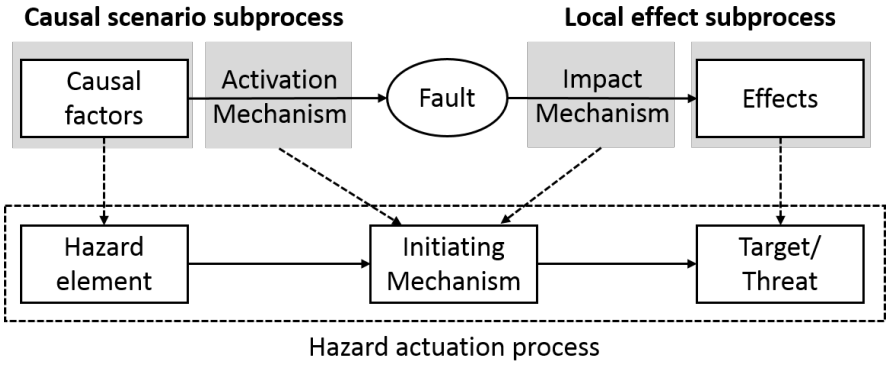


Figure 10. The structure to depict the component fault process appears at the top. It can be mapped to the hazard actuation process at the bottom. The solid arrow represents the causal sequence and the dotted arrow represents the mapping relationship between the proposed structure and the hazard actuation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

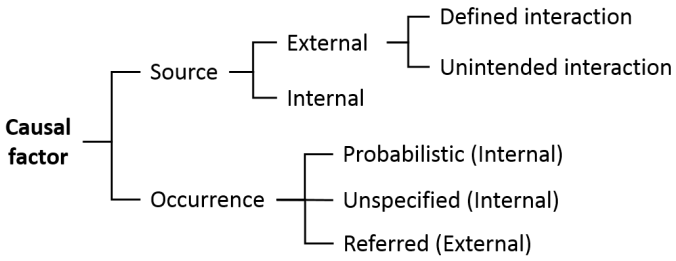


Figure 11. Notable patterns for the casual factors associated with a component fault.

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.

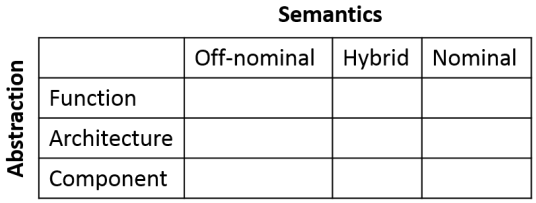


Figure 12. Notable patterns to model the fault effects on a component. A detailed classification of a sample pool of MBSA works is given in the Appendix.

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.

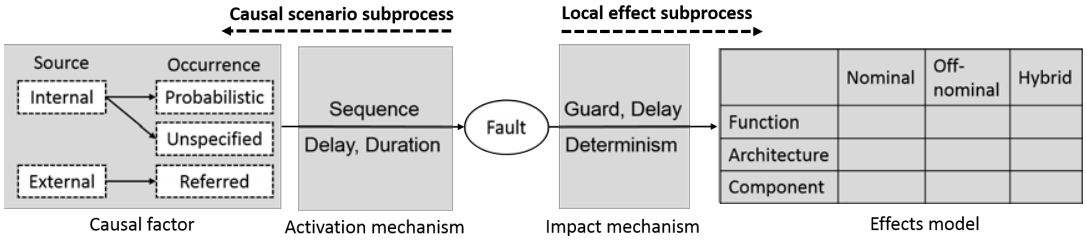


Figure 13. The resulting phenomenon-centric framework to describe the hard facts of the component fault process by combing the structure in section 5.2 and notable patterns in section 5.3.

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

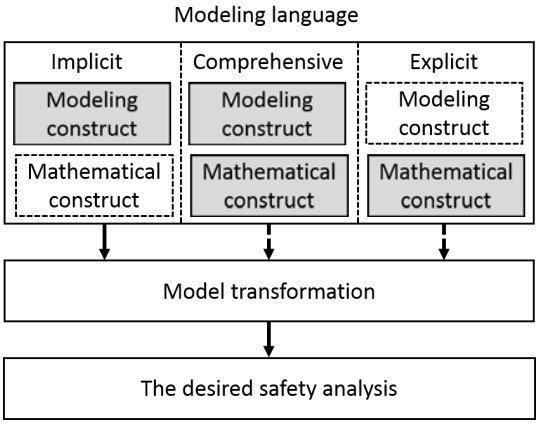


Figure 14. The notable patterns of the mathematical construct.

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).

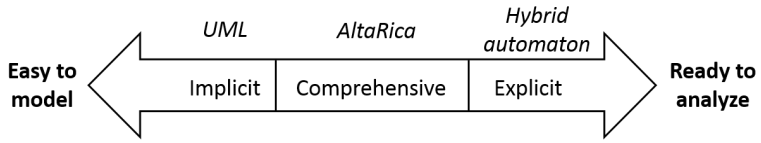


Figure 15. Comparing the three types of MBSA languages.

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.

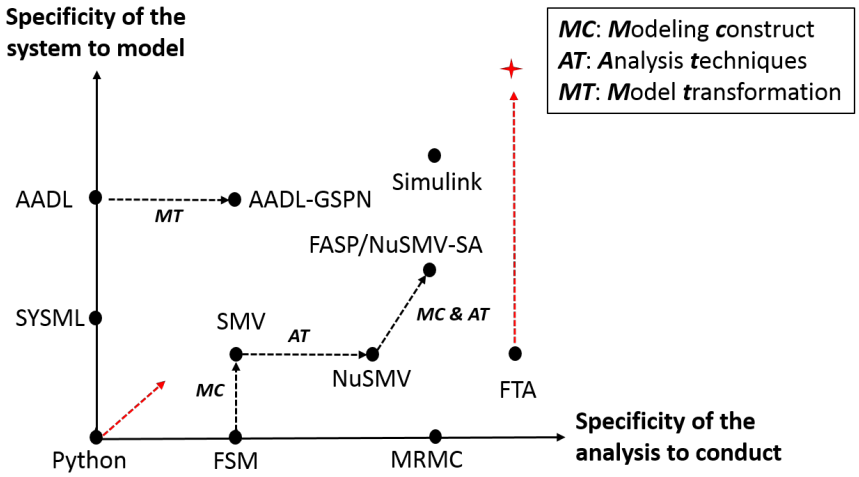


Figure 16. Comparison of modeling language with regard to how much information is assumed about the system to be modelled (the *y*-axis) and the desired analysis (the *x*-axis). The origin is the general language, like Python (not to imply it is MBSA language), that does not assume any information at 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.

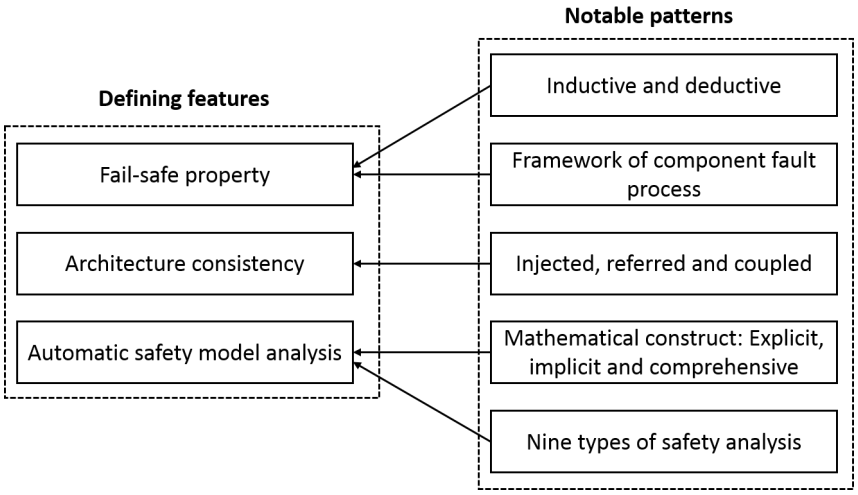


Figure 17. A summary of the defining features and notable patterns. Arrow means "implements" here.

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)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-03-2021	Contractor Report	*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

Appendix C: The design of the reference architecture

The three tasks of the controller follow the timeline of Fig.1. All tasks end at the same time, which is the SEB comes to the planned end, but they start at different time. Task 1 starts at the earliest time when the functional goal is received from the upper level of control; Task 2 starts when the SEB is defined by Task 1; Task 3 starts when the control reference is defined by Task 2; the controlled process starts the execution when the control action is defined by Task 3.

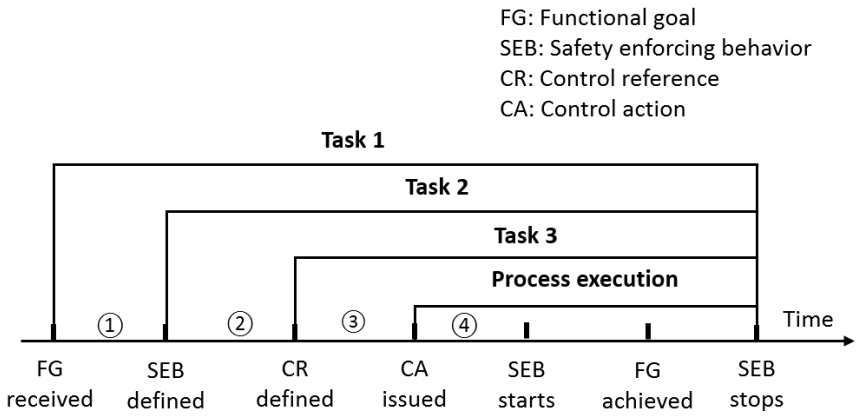


Figure 1: The general timeline of the three tasks of a controller.

1 The building block

The building block of the reference architecture is an *action*, which is in the formalism below and follow the graphical format in Fig.2.

{Input-output-transformation-Trigger-Guard}.

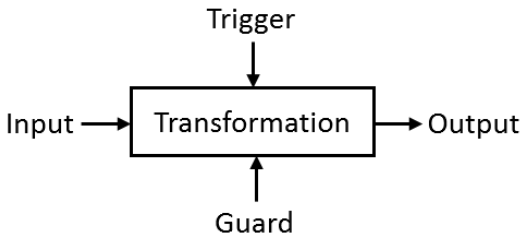


Figure 2: The graphical representation of an action written in the proposed formalism. All the actions will follow such format in the rest of this paper.

- Transformation, input and output: It is the mapping (of data, energy, material, etc) from the input to the output. An action *must* have a transformation and an output. The input either comes from other actions or elements out of the design scope.

- Trigger: The occurrence of a trigger event initiates the activation of the action. If the trigger is not depicted in the action graph, then as long as the guard condition is true, the action can take place *immediately*, given the inputs are available.
- Guard: The guard condition has two functions: (1) if the guard condition is not satisfied, then the action cannot start, and (2) for the action that is still in execution, it has to stop if the guard condition becomes unsatisfactory. If the guard is not depicted in the action graph, then once the trigger event occurs, the action can take place freely, given the inputs are available.

Furthermore, we make the following prescription for an action:

- The connection between actions does not add any time delay or value deviation to whatever being communicated.
- Even if the trigger event occurred and the guard condition is satisfied, the action still needs all the inputs to be effective to execute. If any of the input becomes $null$, the action sets the output to $null$ *immediately*. Note that setting the output to $null$ does not terminate an action. Furthermore, if the input changes while the action is being executed, the current execution stops immediately (without changing the current output value), and a new execution starts with the new input value.
- Once an action starts, it can only be terminated by the guard condition becoming false.
- For an action with multiple outputs, the outputs are always synchronized.
- An action always takes time. However, we assume in this work that all enabling actions can be accomplished instantaneously, and the expected time for main action MA_i is denoted as e_i.
- Currently, only the conditions to issue "Request for Resolution (RfR)" are all defined, but how the RfRs should be resolved is not considered, which is a limitation of the current version of reference architecture. We will address this limitation in the future versions.

In summary, the reference architecture is comprised of a set of connected actions. Each action has a transformation that determines the mapping between the input and the output whose occurrence are defined by the "trigger event" and "guard condition". Note that the difference between an "action" and a "transformation" is non-trivial. It is part of the reason that why an old system is not necessarily safe to use in a new environment even for the same operation, because the new environment might require a different set of trigger events and guard conditions to start and stop the operation.

2 Task 1: Perceive prescriptive constraints

2.1 Fundamentals

We present the workflow of Task 1 in this subsection. The workflow consists of a set of transformations that together describes how Task 1 works in a conceptual sequence, specifically from Stage 1 to Stage 3 and following the sequence that each transformation is described within the respective "themes".

As explained in the previous section, Task 1 is to generate the SEB (and update it if necessary) that will achieve the functional goal and to avoid the hazard at the same time. Three stages are selected from the general timeline in Fig.1 based on the applicable safety requirements.

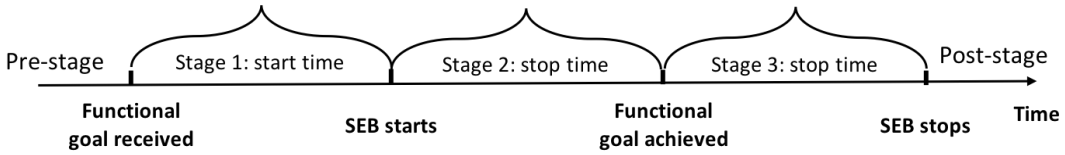


Figure 3: Three stages of Task 1. Stage 3 is different from Stage 2 in that (msp_0, nsp_0, csp_0) is not included in the safety requirements for Stage 3 because the functional goal has already been achieved at that time.

- Stage 1: This stage begins when the functional goal is received and has to find the satisfactory SEB before the must-start time window (if applicable) expires.
- Stage 2: This stage begins when the planned SEB starts and has to make sure the output behavior stops before the must-stop time window (if applicable) expires.
- Stage 3: This stage begins when the functional goal is achieved and has to make sure the output behavior stops before the must-stop time window (if applicable) expires. It is different from Stage 2 in that (msp_0, nsp_0, csp_0) can be taken out of the safety requirements because the functional goal has already been achieved.

Each stage has two main themes of actions, which are (1) generating the SEB and (2) monitoring the SEB. In the rest of this subsection, we are going to explain the workflow of Task 1 organized by these two themes. Note that the workflow of each theme at each stage is comprised of a set of transformations.

2.1.1 Stage 1

In this stage, the generation of the SEB (Fig.4) is triggered by the receive of the functional goal or the update signal from the monitoring theme (Fig.5).

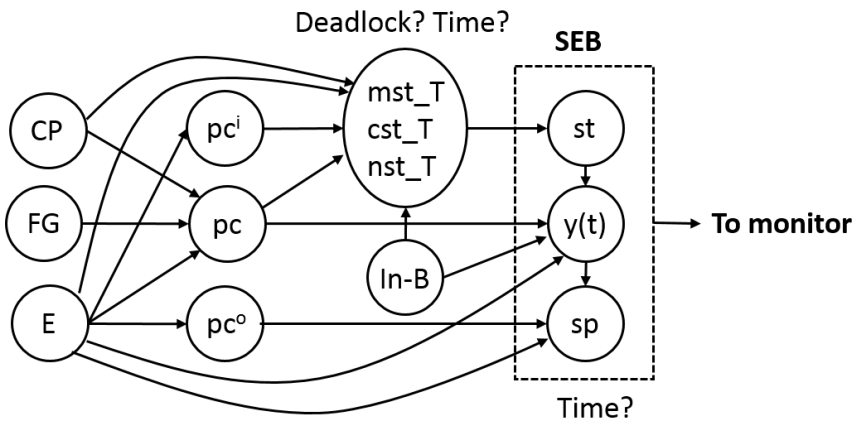


Figure 4: Generating the SEB. In-B: in-behavior; E: environment; st: start time; sp: stop time.

Five transformations are defined as following to generate the SEB (Fig.4).

$$\begin{cases} (E, FG, CP) \xrightarrow{f_1} (pc^i, pc, pc^o) \\ (E, In\text{-}B, pc^i, pc, CP) \xrightarrow{f_2} (mst\_T, nst\_T, cst\_T) \\ (mst\_T, nst\_T, cst\_T) \xrightarrow{f_3} st \\ (In-B, st, E, pc) \xrightarrow{f_4} y(t) \\ (y(t), E, pc^o) \xrightarrow{f_5} sp \end{cases}$$

- f_1: This transformation is to determine the performance constraints based on the observation of the external environment, the controlled process, and the functional goal received from higher level. The performance constraint may vary over some attributes from the external environment. For example, the distance to the terrain depends on the type of the terrain, such as residence, power line, etc. The functional goal (FG) is used to determine the associated performance constraint pc_0, which is part of pc.
- f_2: This transformation is to decide the safety requirements for the start time of the SEB based on the external environment, the controlled process, the projection of the in-behavior and the performance constraints of the in-behavior and the intended behavior. Furthermore, it is possible that there is internal conflicts (i.e. deadlock) between mst_T and nst_T, in which case no start time exists to satisfy both mst_T and nst_T. See the "safety requirements" for a detailed example. Various resolution can be taken to address this situation, for example requesting change of the in-behavior, or request change of the external environment (if it can be changed), or simply waiting if the external environment is highly dynamic. Regardless of the resolution, this conflict must be resolved before mst_T expires, otherwise the system will enter the contingency mode.
- f_3: This transformation is simply the selection of the start time based on the determined safety requirements from f_2.

- f_4: This transformation is to determine the desired evolutionary trajectory of the SEB, which has to start with the value of the $In-B$ at st and satisfy its performance constraint pc with respect to the external environment E.
- f_5: This transformation is to make sure the SEB stops in a way that will not lead to the violation of the performance constraints (pc^o) of the out-behavior.

f_3, f_4 and f_5 are highly coupled, because $y(t)$ starts with the st and the stop time sp is decided based on a tentative $y(t)$. If f_4 or f_5 cannot find the $y(t)$ or sp, it has to be fed back to f_3 for a new st. Therefore, the three transformations can be composed as the transformation $f_{3|4|5}$ below. Furthermore, it might take an extended period of time to find $st, y(t)$ and sp (i.e. the SEB). The bottom line is that a satisfactory SEB has to be found before mst_T expires, otherwise the system will enter the contingency mode.

$$(In-B, E, pc, pc^o, mst\_T, nst\_T, cst\_T) \xrightarrow{f_{3|4|5}} (st, y(t), sp)$$

As a whole, the generation of the SEB (Fig.4) takes the in-behavior and information from the environment, and outputs the start time (st), the stop time (sp) and the dynamic trajectory ($y(t)$). It has to be accomplished within mst_T.

$$(E, FG, In\text{-}B, CP) \rightarrow (st, y(t), sp) \tag{1}$$

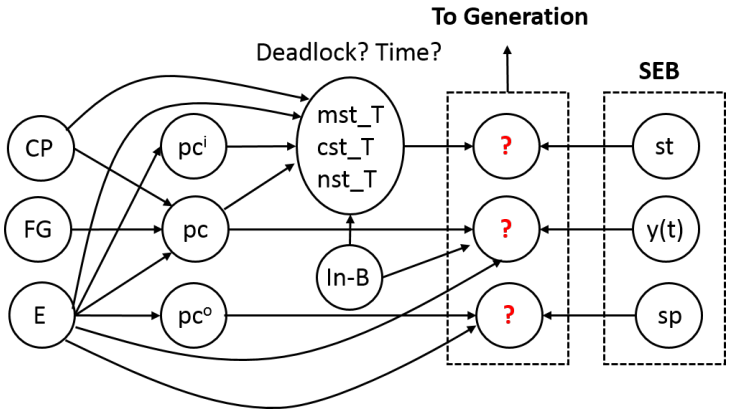


Figure 5: Monitoring the SEB in Stage 1. In-B: in-behavior; E: environment; st: start time; sp: stop time.

Five transformations are identified for monitoring the SEB (Fig.5), triggered by the change of the projection of the in-behavior, the environment or the functional goal. **We assume** if any of $st, y(t)$ or sp is unsatisfactory, the defined SEB is discarded and a new one must be generated, because the controller should not knowingly start an unsatisfactory output behavior. In this way, the five transformations make sure the planned SEB is always satisfactory and the controller will not execute an unsatisfactory SEB.

$$\begin{cases} (E, FG, CP) \xrightarrow{f_1} (pc^i, pc, pc^o) \\ (E, In\text{-}B, CP, pc^i, pc) \xrightarrow{f_2} (mst\_T, nst\_T, cst\_T) \\ (st, mst\_T, nst\_T, cst\_T) \xrightarrow{f_6} sebStatus_1 \\ (In - B, y(t), pc, E) \xrightarrow{f_7} sebStatus_2 \\ (sp, pc^o, E) \xrightarrow{f_8} sebStatus_3 \end{cases}$$

- f_1 and f_2 are already defined for generating the SEB, and hence can be reused.
- f_6: This transformation is to check whether the defined start time (st) still satisfies the safety requirements (mst_T, nst_T, cst_T). If not, then $sebStatus_1$ is set $false$.
- f_7: This transformation is to check whether the defined $y(t)$ still satisfies pc. If not, then $sebStatus_2$ is set $false$.
- f_8: This transformation is to make sure that the defined stop time (sp) still will not lead to the violation of the performance constraints of the out-behavior (pc^o). If not, then $sebStatus_3$ is set $false$.

As a whole, the monitoring of the SEB can be written as below:

$$(In\text{-}B, E, CP, FG, st, y(t), sp) \rightarrow sebStatus_1/sebStatus_2/sebStatus_3 \quad (2)$$

where $sebStatus_1/sebStatus_2/sebStatus_3$ are the outputs of f_6, f_7 or f_8 respectively. If any of $sebStatus_1/sebStatus_2/sebStatus_3$ is $false$, a new SEB must be generated, and hence the actions in Fig.4 will be triggered.

2.1.2 Stage 2 and 3

Both Stage 2 and Stage 3 are to make sure the output behavior stops before the mst_T expires. The only difference is that $(mst_0\_T, nst_0\_T, cst_0\_T)$ is not considered when Stage 2 is accomplished. But the transformations in the two stages are quite similar. For this reason, we only explain the shared transformations once to avoid repetition. For the different transformations, we specify the applicable stages. Furthermore, entering Stage 2, there is always a generated SEB to start with. Therefore, we first introduce the monitoring theme (Fig.6); if the current SEB becomes invalid, then the generating theme will be triggered (Fig.7).

Three transformations are identified for monitoring the SEB (Fig.6), triggered by the change of the environment and/or the functional goal. In this way, the transformations make sure the SEB being executed is valid in the changing environment.

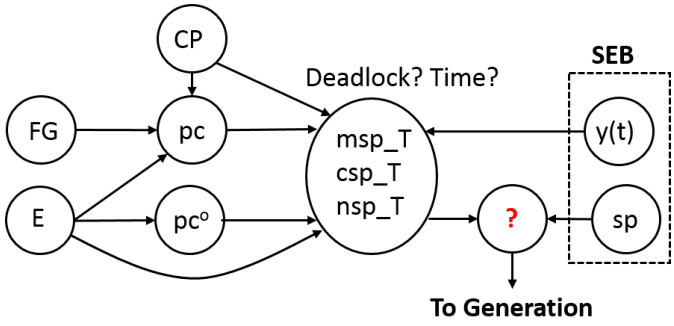


Figure 6: Monitoring the SEB in Stage 2 and Stage 3.

$$\begin{cases} \text{Stage 2}: (CP, E, FG) \xrightarrow{f_9} (pc, pc^o) \\ \text{Stage 3}: (E, CP) \xrightarrow{f_{10}} (pc, pc^o) \\ (E, y(t), pc^o, pc, CP) \xrightarrow{f_{11}} (msp\_T, nsp\_T, csp\_T) \\ \text{Stage 2}: (sp, msp\_T, nsp\_T, csp\_T) \xrightarrow{f_{12}} sebStatus_4 \\ \text{Stage 3}: (sp, msp\_T, nsp\_T, csp\_T) \xrightarrow{f_{12}} sebStatus_5 \end{cases}$$

- f_9: Similar to f_1, this transformation is to calculate the performance constraints based on the observation of the external environment, the controlled process and the functional goal received from higher level. But because the SEB already started in Stage 2, the information associated with the in-behavior is discarded.
- f_{10}: Similar to f_9, this transformation is to calculate the performance constraints based on the observation of the external environment and the controlled process. But because the functional goal is achieved in Stage 3, the information associated with the functional goal is discarded.
- f_{11}: This transformation is to decide the safety requirements on the stop time. Similar to f_2, there can be a deadlock between the safety requirements, and this deadlock has to be resolved before msp_T expires regardless of the specific solutions, otherwise the system must enter the contingency mode. Furthermore, this transformation is conditioned on the conformance of the defined SEB. In other words, if the defined SEB is deviated by the output behavior, the resulting (msp_T, nsp_T, csp_T) cannot be trusted. Therefore, no deviation of the SEB is the *guard condition* of this transformation.
- f_{12}: This transformation is to check whether the defined stop time (sp) still satisfies the safety requirements (msp_T, nsp_T, csp_T). If not, $sebStatus$ is set $false$ and a new SBE must be generated. In this case, the "generate" theme in Fig.7 is triggered.

As a whole, the monitoring of the SEB can be written as below, conditioned on

no deviation of $y(t)$:

$$\begin{cases} \text{Stage 2} : (E, FG, y(t), sp) \rightarrow sebStatus_4 \\ \text{Stage 3} : (E, y(t), sp) \rightarrow sebStatus_5 \end{cases} \tag{3}$$

If f_{12} returns $false$, the task as a whole will return false. In this case, a new SEB must be generated, and hence the actions in Fig.7 will be triggered.

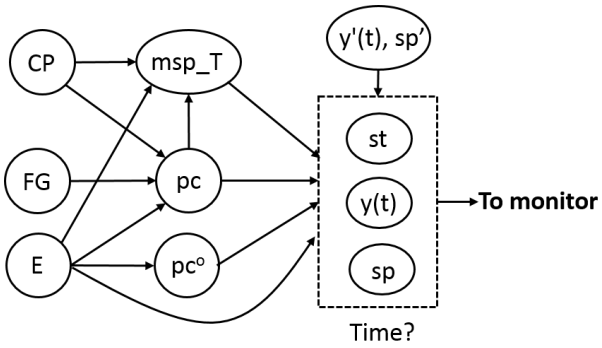


Figure 7: Generating the SEB in Stage 2 and Stage 3.

Four actions are identified for generating the SEB (Fig.7), triggered by the request from the monitoring theme.

$$\begin{cases} \text{Stage 2} : (CP, E, FG) \xrightarrow{f_9} (pc, pc^o) \\ \text{Stage 3} : (CP, E) \xrightarrow{f_{10}} (pc, pc^o) \\ (E, CP, y'(t), pc) \xrightarrow{f_{13}} msp\_T \\ (E, y'(t), pc, msp\_T, sp') \xrightarrow{f_{14}} (st, y(t)) \\ (y(t), E, pc^o, sp') \xrightarrow{f_{15}} sp \end{cases}$$

- f_9 and f_{10} have been defined previously, and hence is reused here.
- f_{13}: This transformation is to calculate the must-stop time window based on the current SEB $y'(t)$ that is to be replaced.
- f_{14}: This transformation is to calculate the start time st and the dynamic trajectory of the SEB $y(t)$, where st has to be picked before msp_T expires and before the current stop time sp' (that is to be replaced) passes; the initial condition $y(st)$ must be consistent with $y'(t)$ at st.
- f_{15}: This transformation is to calculate the stop time (sp) so that it will not lead to the violation of the out-behavior.

f_{14} and f_{15} are coupled because the point to stop can only be chosen based on the $y(t)$. If the sp cannot be found, a new $y(t)$ needs to found. Therefore, the two functions can be composed as one function $f_{14|15}$ below. Furthermore, it can take an extended period of time to find $y(t)$ and sp. But they must be found and

executed before msp_T (from f_{13}) expires, otherwise the system has to enter the contingency mode.

$$(E, pc, pc^o, y'(t), msp\_T, sp') \xrightarrow{f_{14|15}} (st, y(t), sp)$$

As a whole, the generating of the SEB can be written as below, conditioned on no deviation of the current $y(t)$:

$$\begin{cases} \text{Stage 2} : (E, FG, CP, y'(t), sp') \rightarrow (st, y(t), sp) \\ \text{Stage 3} : (E, CP, y'(t), sp') \rightarrow (st, y(t), sp) \end{cases} \tag{4}$$

If a new SEB cannot be found before msp_T, the system must enter the contingency mode.

As a summary, the workflow of Task 1 is comprised of the following transformations. Note that some of transformations will appear in multiple places of the reference architecture in different actions, which is why this reference architecture is a "logical architectrue".

$$\begin{cases} (E, FG, CP) \xrightarrow{f_1} (pc^i, pc, pc^o) \\ (E, In\text{-}B, pc^i, pc, CP) \xrightarrow{f_2} (mst\_T, nst\_T, cst\_T) \\ (In - B, E, pc, pc^o, mst\_T, nst\_T, cst\_T) \xrightarrow{f_{3|4|5}} (st, y(t), sp) \\ (st, mst\_T, nst\_T, cst\_T) \xrightarrow{f_6} sebStatus_1 \\ (In - B, y(t), pc, E) \xrightarrow{f_7} sebStatus_2 \\ (sp, pc^o, E) \xrightarrow{f_8} sebStatus_3 \\ (E, FG, CP) \xrightarrow{f_9} (pc, pc^o) \\ (CP, E) \xrightarrow{f_{10}} (pc, pc^o) \\ (E, y(t), pc^o, pc, CP) \xrightarrow{f_{11}} (msp\_T, nsp\_T, csp\_T) \\ (sp, msp\_T, nsp\_T, csp\_T) \xrightarrow{f_{12}} sebStatus_4/sebStatus_5 \\ (E, CP, y'(t), pc) \xrightarrow{f_{13}} msp\_T \\ (E, y'(t), pc, pc^o, msp\_T, sp') \xrightarrow{f_{14|15}} (st, y(t), sp) \end{cases}$$

2.2 The enabling actions (EAs) of Task 1

The actions for Task 1 is comprised of two groups: the main actions and the enabling actions. The main actions are a set of actions that refines Task 1 by specifying the transformations identified in the workflow with the trigger events and the guard conditions. The enabling actions are a set of actions that supports the main actions. When connected together, the actions (both the main actions and the enabling actions) comprise the reference architecture of Task 1. We introduce the enabling action first to set the stage for the main actions.

Deciding the stages (EA_1). The first enabling action is to decide which stage that Task 1 is currently in. Intuitively, the meaning of the three stages of Task 1 can be found in Fig.3. Formally, the three stages can be coined by two variables: the output behavior ($OB = \{on, off\}$) and the status of the functional goal ($fgStatus = \{false, Requested, Achieved\}$), where OB means whether the intended output behavior has started (on) or not (off), and $fgStatus$ means whether a valid functional goal is requested from the higher level and whether the requested functional goal is achieved ($Achieved$) or not ($Requested$). $false$ means the functional goal has not been requested or the requested functional goal is cancelled.

fgStatus	False	Requested		Achieved	
OB	off	off	on	on	off
Stage	preStage	Stage1	Stage2	Stage3	postStage

Figure 8: The definition the three stages of Task 1.

The three stages of Task 1 can be defined by these two variables (Fig.8). At $preStage$, the functional goal is not requested ($fgStatus = false$) and there is no output behavior ($OB = off$). Once the functional goal is requested ($fgStatus = Requested$) and before the intended output behavior starts ($OB = off$), the $stage$ transitions to $Stage1$. Then the intended output behavior starts ($OB = on$) and when the functional goal is not yet achieved ($fgStatus = Requested$), it is $Stage2$. Once the functional goal is achieved ($fgStatus = Achieved$), and if the intended output behavior is still on ($OB = on$), it is $Stage3$. As soon as the intended output behavior stops, it enters the $postStage$, which is an transient state that transitions to $preStage$ by setting the $fgStatus$ to be $false$ immediately.

However, a static Fig.8 is not enough to define the stages. $stage$ can only defined through transitions. For example, $Stage1$ is $fgStatus$ transitions from $false$ to $Requested$ while $OB = off$, rather than OB transitions from on to off while the $fgStatus = Requested$. Therefore, we exhausted all the 20 possible transitions and identified 8 meaningful transitions that drive the transitions between the states of $stage$. As a result, 8 sub-actions are defined accordingly as below:

1. Transitioning from $preStage$ to $Stage1$.
    - Trigger event: $fgStatus : false \rightarrow Requested$ [1].
    - Guard condition: $OB = off$.
    - Input: None.
    - Output: The $stage$.
    - Transformation: $stage \leftarrow Stage1$ [2].

[1] The notation $x : a \rightarrow b$ means the value of variable x transitions from a to b.
[2] The notation $x \leftarrow a$ means assign value a to variable x.

2. Transitioning from $Stage1$ to $Stage2$.
    - Trigger event: $OB : off \rightarrow on$.
    - Guard condition: $fgStatus = Requested$.
    - Input: None.
    - Output: The $stage$.
    - Transformation: $stage \leftarrow Stage2$.
3. Transitioning from $Stage2$ to $Stage3$.
    - Trigger event: $fgStatus : Requested \rightarrow Achieved$.
    - Guard condition: $OB = on$.
    - Input: None.
    - Output: The $stage$.
    - Transformation: $stage \leftarrow Stage3$.
4. Transitioning from $Stage3$ to $postStage$.
    - Trigger event: $OB : on \rightarrow off$.
    - Guard condition: $fgStatus = Achieved$.
    - Input: None.
    - Output: The $stage$ and $fgStatus$.
    - Transformation: $stage \leftarrow postStage$ and $fgStatus \leftarrow false$.
5. Transitioning from $postStage$ to $preStage$.
    - Trigger event: $fgStatus : Achieved \rightarrow false$.
    - Guard condition: $OB = off$.
    - Input: None.
    - Output: The $stage$.
    - Transformation: $stage \leftarrow preStage$.
6. Transitioning from $Stage1$ to $preStage$. This corresponds to the scenario where the functional goal is cancelled before the output behavior even starts.
    - Trigger event: $fgStatus : Requested \rightarrow false$.
    - Guard condition: $OB = off$.
    - Input: None.
    - Output: The $stage$.
    - Transformation: $stage \leftarrow preStage$.
7. Transitioning from $Stage1$ to $Stage3$. This corresponds to the scenario where the output behavior is discrete.

- Trigger event: $\{fgStatus : Requested \rightarrow Achieved\} \wedge \{OB : off \rightarrow on\}$.
- Guard condition: NA.
- Input: None.
- Output: The $stage$.
- Transformation: $stage \leftarrow Stage3$.

8. Transitioning from $Stage2$ to $postStage$. This corresponds to the scenario where the output behavior stops immediately after the functional goal is achieved.
    - Trigger event: $\{fgStatus : Requested \rightarrow Achieved\} \wedge \{OB : on \rightarrow off\}$.
    - Guard condition: NA.
    - Input: None.
    - Output: The $stage$ and $fgStatus$.
    - Transformation: $stage \leftarrow postStage$ and $fgStatus \leftarrow false$.

In addition, 2 extra sub-actions are defined when the functional goal is cancelled while the output behavior is still on. The response can only be defined on a case-by-case basis. Therefore, we define the controller enters the contingency mode as a general response.

9. Transitioning from $Stage2$ to contingency mode.
    - Trigger event: $fgStatus : Requested \rightarrow false$.
    - Guard condition: $OB = on$.
    - Input: None.
    - Output: CM_{15}.
    - Transformation: $CM_{15} \leftarrow true$.
10. Transitioning from $Stage3$ to contingency mode.
    - Trigger event: $fgStatus : Achieved \rightarrow false$.
    - Guard condition: $OB = on$.
    - Input: None.
    - Output: CM_{16}.
    - Transformation: $CM_{16} \leftarrow true$.

The rationale for the 12 excluded transitions are summarized in Table.1

As a result, EA_1 can be defined programmatically by the 10 transitions. The corresponding transformation is denoted as f_{16} (continuing the transformation

Table 1: The excluded 12 transitions. But the exclusion does not mean they will never happen. They can happen in the case of failure or design errors. But since we focus on getting the correct design without failure, the 12 transitions are excluded as they are against the correct design intent.

From	To	Rationale
preStage	Stage 3	The functional goal must be requested first and cannot jump directly from $false$ to $achieved$.
preStage	postStage	Same as above.
preStage	Stage2	Same as above.
Stage1	postStage	The functional goal cannot be achieved without the output behavior is on.
Stage2	Stage1	The output behavior is not supposed to stop before the functional goal is achieved.
Stage2	preStage	The output behavior always stops after the functional goal is cancelled, not at the same time.
Stage3	preStage	Same as above.
Stage3	Stage1	An achieved functional goal cannot be "unachieved".
Stage3	Stage2	Same as above.
postStage	Stage3	$postStage$ always automatically transitions to $preStage$ immediately by setting the functional goal to $false$.
postStage	Stage1	Same as above.
postStage	Stage2	Same as above.

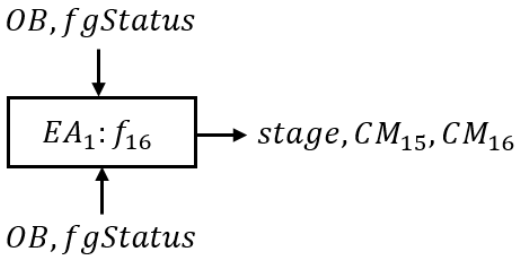


Figure 9: Graphical representation of EA_1

numbering from the workflow section). Graphically, Action EA_1 can be represented in Fig.9.

Start/stop watcher (EA_2). This action is to address the possibility that mst_T or msp_T is violated. Specifically, if the intended output behavior does not start before mst_T expires, or the intended output behavior does not stop before msp_T expires, the system enters contingency mode by sending out CM_4. Therefore, the following two sub-actions are defined.

Sub-action 1:

- Trigger event: mst_T expires.

- Guard condition: $stage = Stage1$.
- Input: OB.
- Output: CM_4.
- Transformation (f_{17}): If $OB = false$, then $CM_4 \leftarrow true$.

Sub-action 2:

- Trigger event: msp_T expires.
- Guard condition: $stage = Stage2 \vee Stage3$.
- Input: OB.
- Output: CM_4.
- Transformation (f_{17}): If $OB = true$, then $CM_4 \leftarrow true$.

Graphically, Action EA_2 can be represented in Fig.10.

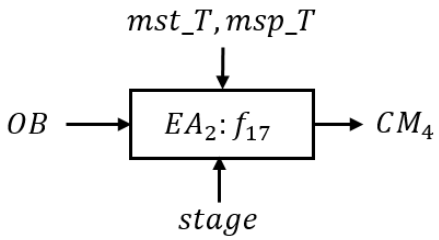


Figure 10: Graphical representation of EA_2.

Deviation watcher (EA_3). At any time the real output behavior must satisfy the planned SEB. Consequently, deviation can be denoted as $\widetilde{y}(t) \notin y(t)$. Although deviation of the real output from the planned SEB does not necessarily lead to hazard, it means the safety is not enforced and the controller has no control over whether the system will enter a hazardous state. In fact, the whole point of Task 2 and Task 3 is to make sure the real output behavior is well bounded by the planned SEB, or react before the planned SEB is deviated. However, because of the unavoidable aleatoric uncertainty, it is always possible that the real output behavior deviates from the planned SEB before any actions can be applied as a response. This is why EA_3 is defined, to capture such deviation in a timely manner. However, once the deviation is detected, the activities to address it vary from case to case. Therefore, we simply define that the controller enters contingency mode upon the deviation, and leave the resolution to the specific applications.

Therefore, EA_3 can be defined as below programmatically:

- Trigger event: $\{\widetilde{y}(t) \notin y(t)\}$ transitions from $false$ to $true$.

- Guard condition: $stage = Stage2 \vee Stage3$.
- Input: NA.
- Output: CM_{17}.
- Transformation (f_{18}): $CM_{17} \leftarrow true$.

Graphically, Action EA_3 can be represented in Fig.11.

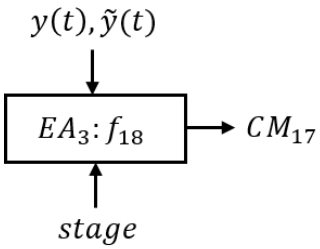


Figure 11: Graphical representation of EA_3. Because it is a placeholder, input and output are not specified.

Contingency mode (EA_4). We identified 17 situations throughout the reference architecture that will fail to achieve the functional goal and avoid the hazard at the same time. Appropriate actions are needed to address each situation to avoid the hazard. However, because the actions have to be decided on a case-by-case basis, we leave the specific actions for each contingency mode to the specific design applications by defining a placeholder transformation f_{19} to represent all the potential activities.

Therefore, EA_4 can be defined as below programmatically.

- Trigger event: $CM_1 \vee ... \vee CM_{17} : false \rightarrow true$.
- Guard condition: $stage! = (preStage \vee postStage)$.
- Input: NA.
- Output: NA.
- Transformation: Execute f_{19}.

Graphically, Action EA_4 can be represented in Fig.12.

SEB status monitor (EA_5). A generated SEB might become unsatisfactory due to the change in the environment, the functional goal and the controlled process. This action is to constantly monitor the satisfiability of the generated SEB, denoted by the output variable $sebStatus$. If a new SEB is generated, $sebStatus$ is set to $true$ automatically; if the monitor theme detects the current SEB is not satisfactory any more, $sebStatus$ will be set false. In addition, if the unsatisfiability is not found at $Stage1$, then the SEB is set to $false$ to prevent the execution

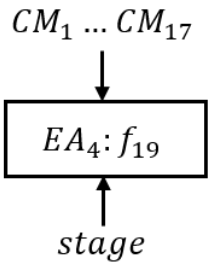


Figure 12: Graphical representation of EA_4. Because it is a placeholder, input and output are not specified.

of an unsatisfactory SEB. As a result, we define the following three sub-actions for EA_5:
Sub-action 1:

- Trigger event: $sebStatus_1 \vee sebStatus_2 \vee sebStatus_3 \vee sebStatus_4 \vee sebStatus_5 : true \rightarrow false$.
- Guard condition: $stage! = (preStage \vee postStage)$.
- Input: NA.
- Output: $sebStatus$.
- Transformation: $sebStatus \leftarrow false$.

Sub-action 2:

- Trigger event: $sebStatus_1 \vee sebStatus_2 \vee sebStatus_3 : true \rightarrow false$.
- Guard condition: $stage = Stage1$.
- Input: NA.
- Output: $(st, y(t), sp)$.
- Transformation: $(st, y(t), sp) \leftarrow false$.

Sub-action 3:

- Trigger event: $(st, y(t), sp)$ changes.
- Guard condition: $stage! = (preStage \vee postStage) \wedge (st, y(t), sp)! = false$
- Input: NA.
- Output: $sebStatus$.
- Transformation: $sebStatus \leftarrow true$.

Graphically, Action EA_5 can be represented in Fig.13.

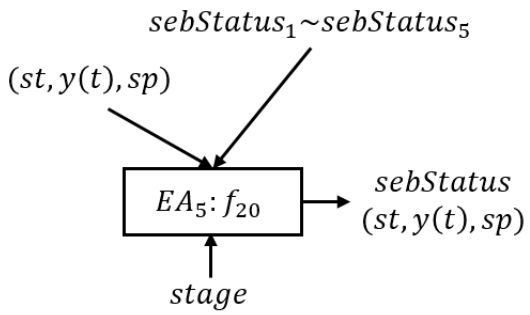


Figure 13: Graphical representation of EA_5.

2.3 The main actions (MAs) of Task 1

The main actions refine Task 1 by specifying the transformations identified in the workflow with the trigger events and the guard conditions. As explain in the workflow, the controller transitions between the themes based on the specific operational conditions, and each theme is further refined by a set of actions. Therefore, we first define the themes at each stage with the construct of an action, and then zoom in each theme to specify the comprising actions.

2.3.1 Defining the "themes" of each stage.

Although the "theme" is not an internal construct of the reference architecture, we define them as high level actions as stepping stones to "divide and conquer" Task 1. Later, the themes will be refined by more specific actions.

First, the transformation of each theme at each stage has already been explained through (1)∼(4). We summarize them as below.

$$\text{Stage 1:} \begin{cases} \text{Generate: } (E, FG, In\text{-}B) \rightarrow (st, y(t), sp) \\ \text{Monitor: } (In\text{-}B, E, FG, st, y(t), sp) \rightarrow sebStatus \end{cases}$$

$$\text{Stage 2:} \begin{cases} \text{Generate: } (E, FG, y(t), \tilde{y}(t_c)) \rightarrow (y(t), sp) \\ \text{Monitor: } (E, FG, y(t), sp) \rightarrow sebStatus \end{cases}$$

$$\text{Stage 3:} \begin{cases} \text{Generate: } (E, y(t), \tilde{y}(t_c)) \rightarrow (y(t), sp) \\ \text{Monitor: } (E, y(t), sp) \rightarrow sebStatus \end{cases}$$

Furthermore, the transitions between the themes through the three stages are described in Fig.14. We explain the transitions of each theme now.

- Generate theme of Stage 1: This theme is triggered once the Stage 1 starts (i.e. the right-ward arrow) or the generated SEB becomes unsatisfactory before it is executed (i.e. the bottom-up arrow). Furthermore, this theme can only operate when the functional goal is true and when the timeline is still in Stage 1. Therefore, programmatically, we define the following trigger

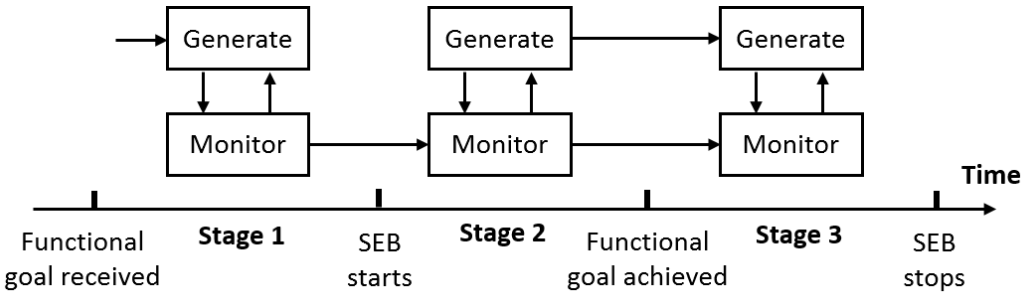


Figure 14: The transitions (denoted by the arrows) between the themes through the three stages. The "generate" theme at Stage 1 is the start point of the entire process.

events and guard conditions.

$$\begin{cases} \text{Trigger: } \{stage : preStage \rightarrow Stage1\} \vee \{sebStatus : true \rightarrow false\} \\ \text{Guard: } \{stage = Stage1\} \wedge \{sebStatus = false\} \end{cases}$$

- Monitor theme of Stage 1: This theme is triggered once a new SEB is found by the "generte" theme and while the functional goal is true and the timeline is still in Stage 1. Therefore, programmatically, we define the following trigger events and guard conditions:

$$\begin{cases} \text{Trigger: } \{sebStatus : false \rightarrow true\} \\ \text{Guard: } \{stage = Stage1\} \wedge \{sebStatus = true\} \end{cases}$$

- Monitor theme of Stage 2: First, the transition from Stage 1 to Stage 2 can only be accomplished from the monitor theme of Stage 1 to the monitor theme of Stage 2 (the right-ward arrow), because if the controller is in the generate theme of Stage 1, it means there is no satisfactory SEB found for the output behavior to execute, and hence by definition Stage 1 is not finished. Second, when the timeline is at Stage 2, the monitor theme can be triggered once a new SEB is found by the generate theme (the top-down arrow). Therefore, programmatically, we define the following trigger events and guard conditions.

$$\begin{cases} \text{Trigger: } \{stage : Stage1 \rightarrow Stage2\} \vee \{sebStatus : false \rightarrow true\} \\ \text{Guard: } \{stage = Stage2\} \wedge \{sebStatus = true\} \end{cases}$$

- Generate theme of Stage 2: This theme is triggered once the SEB being executed becomes unsatisfactory (the bottom-up arrow). Therefore, programmatically, we define the following trigger events and guard conditions.

$$\begin{cases} \text{Trigger: } \{sebStatus : true \rightarrow false\} \\ \text{Guard: } \{stage = Stage2\} \wedge \{sebStatus = false\} \end{cases}$$

- Generate theme of Stage 3: This theme can be triggered by the transition from Stage 2 to Stage 3 (the right-ward arrow) and while a satisfactory SEB is still unavailable, or triggered by the current SEB becomes unsatisfactory (the bottom-up arrow). Therefore, programmatically, we define the following trigger events and guard conditions.

$$\begin{cases} \text{Trigger: } \{stage : Stage2 \rightarrow Stage3\} \vee \{sebStatus : true \rightarrow false\} \\ \text{Guard: } \{stage = Stage3\} \wedge \{sebStatus = false\} \end{cases}$$

- Monitor theme of Stage 3: This theme can be triggered by the transition from Stage 2 to Stage 3 (the right-ward arrow) and while the SEB being executed is still satisfactory, or triggered by a new SEB is found (the top-down arrow). Therefore, programmatically, we define the following trigger events and guard conditions.

$$\begin{cases} \text{Trigger: } \{stage : Stage2 \rightarrow Stage3\} \vee \{sebStatus : false \rightarrow true\} \\ \text{Guard: } \{stage = Stage3\} \wedge \{sebStatus = true\} \end{cases}$$

2.3.2 Defining the main actions

We zoom into and refine each theme at each stage to define the comprising actions in this section.

The generate theme at Stage 1. We define three main actions for the generate theme (Fig.15) in accordance with the three transformations f_1, f_2 and $f_{3|4|5}$ identified in the workflow of Fig.4. Now we explain the main actions one by one.

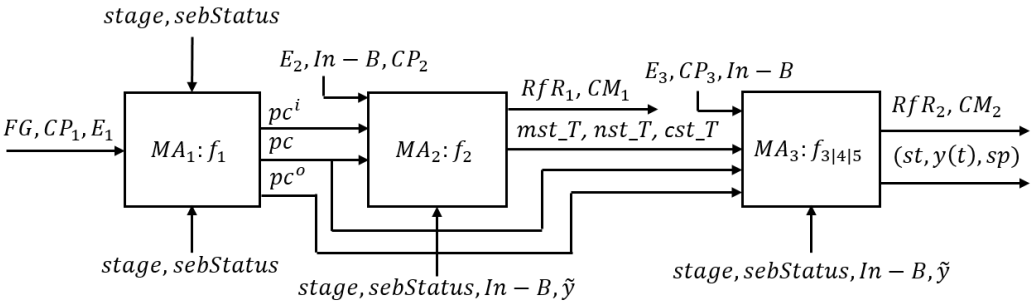


Figure 15: The main actions of the generate theme at Stage 1.

- $\boldsymbol{MA_1}$
  - Transformation: f_1 is to generate the performance constraints of the in-behavior, the intended behavior and the out-behavior (pc^i, pc, pc^o) by logic negation of the given hazard. Because the performance constraints might vary from the specific context, for example the minimal distance from the terrain depends on the type of terrain such as residence, power grid and natural area, information from the environment E needs to be taken to decide the performance constraints if necessary. Finally, the functional goal FG is used to generate pc_0.
  - Duration: The expected time duration e_1 must be short enough so that there will be enough time for MA_2 to generate mst_T before mst_T expires. There is always a chance that e_1 is too long that MA_2 eventually generates mst_T after mst_T expires. Therefore, a long enough

look-ahead time must be selected to decide mst_T early to lower the likelihood of an expired mst_T to an acceptable level.

- Output: Each of pc^i, pc and pc^o is a set because usually there are more than one hazard that are considered. For example, the performance constraints of hazard i will be denoted as pc_i^i, pc_i or pc_i^o. pc_0, translated from the functional goal, is a special case because it only constrains the intended behavior. Therefore, there is only pc_0 with respect to the functional goal and $pc_0 \in pc$.
- Input: First, FG is the functional goal from higher level of control. Second, E_1 is a set of information that comes from the environment to generate the performance constraints and CP_1 is a set of information about the controlled process to generate the performance constraints. The specific information items to observe from the environment are determined with the definition of f_1. Note that the "environment" here means any object that is outside the controller and the controlled process, not necessarily the natural environment.
- Trigger: Individually, there is no trigger event for MA_1. But because the generate theme starts with MA_1, MA_1 inherited the trigger event defined for the generate theme to start the generate theme as a whole. Therefore, the trigger of MA_1 is $\{stage : preStage \rightarrow Stage1\} \vee \{sebStatus : true \rightarrow false\}$.
- Guard: Individually, there is no guard condition for MA_1. But the guard conditions for the generate theme are also the guard conditions for each of the comprising main actions. Therefore, the guard conditions for MA_1 is $\{stage = Stage1\} \wedge \{sebStatus = false\}$.

- $\boldsymbol{MA_2}$
  - Transformation: f_2 is to generate the safety requirements for the intended output behavior to start. Assuming the guard condition is satisfied, it first calculates (mst_T, nst_T, cst_T) with from the inputs (refer to the "safety requirements" paper for details). If there is internal conflicts, then send a request to the higher level of control(i.e. RfR_1) for a new FG or wait for the environment to change. Furthermore, if no viable mst_T, nst_T and cst_T can be found before $\overline{mst\_T} - T1 - e_3$, then the controller stops the main action and enters the contingency mode by sending out CM_1 to EA_4.
  - Duration: The expected time duration e_2 must be short enough so that mst_T will not be generated after mst_T expires. However, there is always a chance that e_2 is too long that mst_T is generated after mst_T expires. Therefore, a long enough look-ahead time must be selected to decide mst_T early to lower the likelihood of an expired mst_T to an acceptable level.
  - Output: The safety requirements (mst_T, nst_T, cst_T) for the intended behavior to start. RfR_1 [3] is sent to the higher level of control to adjust the functional goal. CM_1 is sent to EA_4 to enter contingency mode.

[3] RfR stands for Request for Resolution.

- Input: pc^i and pc are the performance constraints for the in-behavior and the intended behavior. $In-B$ is the projected trajectory of the in-behavior; E_2 and CP_2 are the two sets of information from the environment and the controlled process to generate the time window for the start time.
- Trigger: NA.
- Guard: Similar to MA_1, the guard conditions for MA_2 includes $\{stage = Stage1\} \wedge \{sebStatus = false\}$. In addition, if $In-B$ is deviated, (mst_T, nst_T, cst_T) cannot be trusted. Therefore, no deviation between $In-B$ and $\widetilde{y}$ is also a guard condition, where $\widetilde{y}$ is the observed output value in real time.

- $\boldsymbol{MA_3}$
  - Transformation: $f_{3|4|5}$ is to calculate the SEB whose start time satisfies (mst_T, nst_T, cst_T), dynamic trajectory respects pc and stop time will not lead to a violation of pc^o. In the case that a satisfactory SEB cannot be found, RfR_2 should be sent to the higher level of control to adjust the functional goal. If a satisfactory SEB cannot be found time $T1$ before $\overline{mst\_T}$, the controller stops the main action and enters the contingency mode by sending CM_2 to EA_4.
  - Duration: None. The duration may affect whether there will be enough time that the defined output behavior can be executed, but the safety concerns that the desired output behavior will not start before mst_T is addressed by EA_2. [4]
  - Output: $(st, y(t), sp)$, where $t \in [st, sp]$ is the SEB that is intended to be achieved by the output behavior. The RfR_2 is the request to the higher level of control for a new functional goal when no satisfactory SEB can be found. CM_2 is sent to EA_4 to enter the contingency mode.
  - Input: All the inputs $(In-B, mst\_T, nst\_T, cst\_T, pc, pc^o)$ are explained before. Note that E_3 and CP_3 are the sets of information from the environment and the controlled process respectively to generate the SEB; $y(st)$ must be consistent with $In-B$.
  - Trigger: NA.
  - Guard: Similar to MA_1 and MA_2, MA_3 also inherited the guard condition of the generate theme as a whole, which is $\{stage = Stage1\} \wedge \{sebStatus = false\}$. In addition, if $In-B$ is deviated, there is no way to predict $y(st)$. Therefore, no deviation between $In-B$ and $\widetilde{y}$ is also a guard condition, where $\widetilde{y}$ is the observed output value in real time.

In summary, the generate theme at Stage 1 can be represented as an action in Fig.16, where $E = \{E_1, E_2, E_3\}$ and $CP = \{CP_1, CP_2, CP_3\}$. The sources of the all the external interactions with other themes, the enabling actions and the elements out of the design scope are specified in Fig.17.

[4] This is another example that this reference architecture is **safety-guided**.

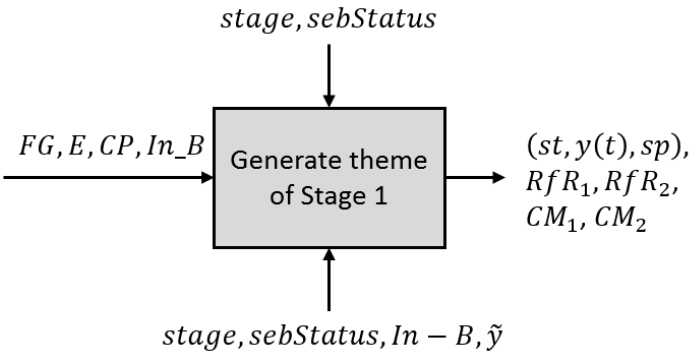


Figure 16: The generate theme at Stage 1 represented as an action. The internal interactions are hidden within the grey box.

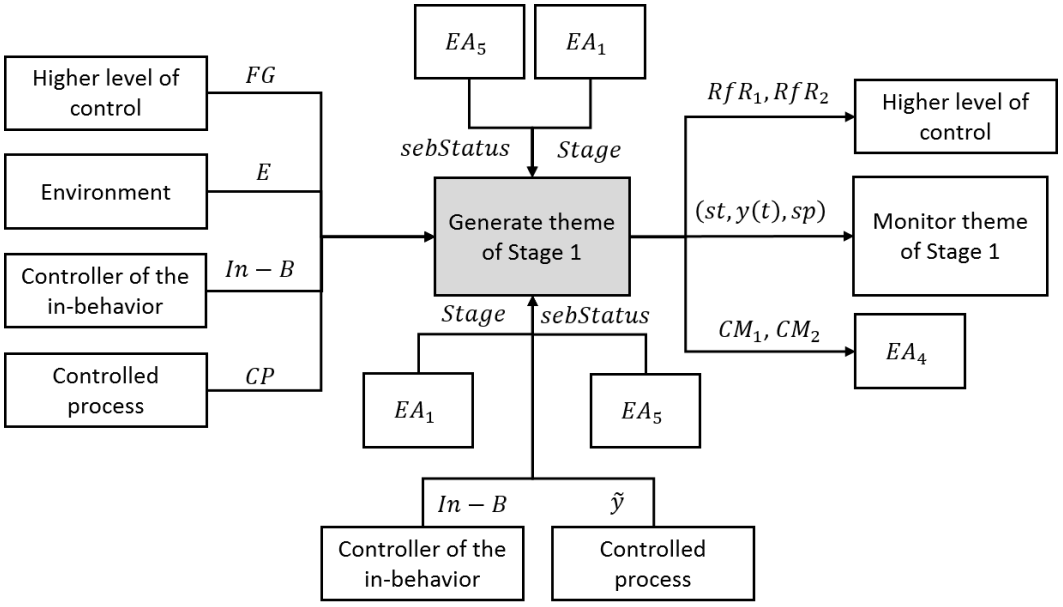


Figure 17: The external interactions of the generate theme at Stage 1 as a whole.

The monitor theme at Stage 1. We define five main actions for the monitor theme (Fig.18) in accordance with the five transformations f_1, f_2, f_6, f_7 and f_8 identified in the workflow of Fig.5. The reason that st is passed from MA_5 to MA_6 is to make sure MA_6 makes decisions based on the latest time window for the start time. Now we explain the main actions one by one.

- $\boldsymbol{MA_4}$
  - Transformation: Same as MA_1.
  - Duration: The expected time duration e_4 must be short enough so that there will be enough time for MA_5 to generate mst_T and nst_T before the selected st passes and before mst_T expires. However, there is always a chance that e_4 is too long that MA_5 eventually generates mst_T and nst_T after st passes or mst_T expires. Therefore, a long enough look-ahead time must be selected to decide mst_T early to

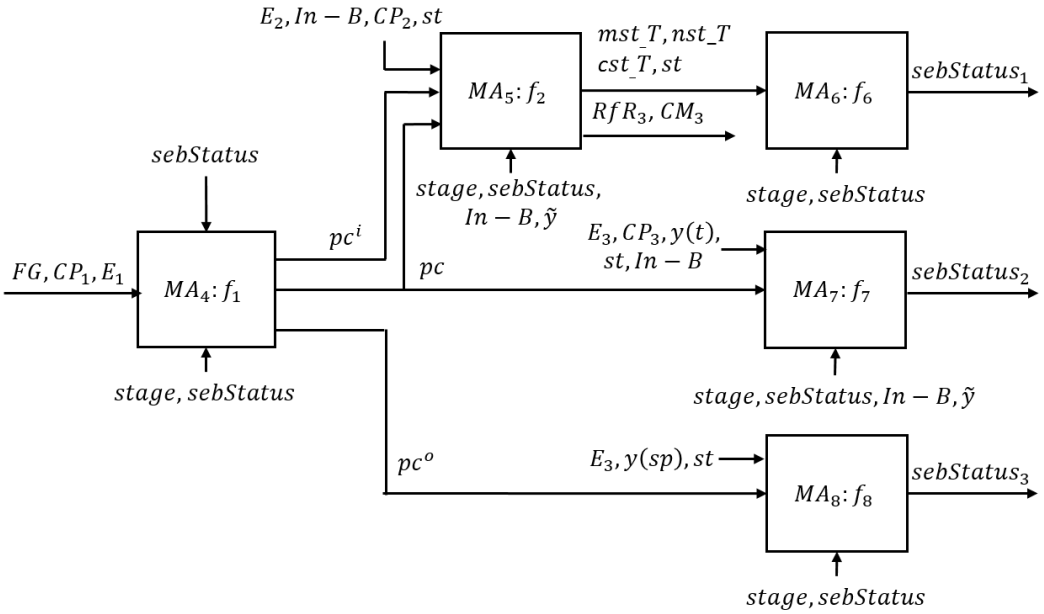


Figure 18: The main actions of the monitor theme at Stage 1.

lower the likelihood of an expired mst_T or a passed st to an acceptable level.

- Output: Same as MA_1.
- Input: Same as MA_1. Note that as long as MA_4 has been triggered and the guard condition is satisfied, MA_4 will always respond to any input change.
- Trigger: Individually, there is no trigger event for MA_4. But because the monitor theme starts with MA_4, MA_4 inherited the trigger event defined for the monitor theme to start the monitor theme as a whole. Therefore, the trigger of MA_4 is $\{sebStatus : true \rightarrow false\}$. Refer to the last section for the rationale.
- Guard: Individually, there is no the guard condition for MA_4. But the guard condition of MA_4 is inherited from the monitor theme as a whole, which is $\{stage = Stage1\} \wedge \{sebStatus = true\}$.

- $\boldsymbol{MA_5}$
  - Transformation: Similar to MA_2. The different part is that if there is internal conflict between mst_T and nst_T, RfR_3 is sent to higher level of control for resolution; if no viable start time can be found from mst_T, nst_T and cst_T before st, then assign $false$ to mst_T, nst_T and cst_T at $st-e_6$; if no viable start time can be found from mst_T, nst_T and cst_T before $\overline{mst\_T} - T1 - e_6 - e_1 - e_2 - e_3$, then the controller stops the main action and enters the contingency mode by sending out CM_3 to EA_4.
  - Duration: The expected time duration e_5 to generate mst_T and nst_T must be short enough so that there will be enough time to generate mst_T and nst_T before the selected st passes and mst_T expires.

However, there is always a chance that e_5 is too long that mst_T and nst_T are generated after st passes or mst_T expires. Therefore, a long enough look-ahead time must be selected to decide mst_T early to lower the likelihood of an expired mst_T or a passed st to an acceptable level.

- Output: Same as MA_2. RfR_3 and CM_3 in MA_5 serve the same purpose of RfR_1 and CM_1 in MA_2.
- Input: Same as MA_2 with the only difference st.
- Trigger: NA.
- Guard: $\{stage = Stage1\} \wedge \{sebStatus = true\}$, which is inherited from the monitor theme at Stage 1 as a whole. In addition, if $In - B$ can be deviated, (mst_T, nst_T, cst_T) cannot be trusted. Therefore, no deviation between $In - B$ and $\widetilde{y}$ is also a guard condition, where $\widetilde{y}$ is the observed output value in real time.

- $\boldsymbol{MA_6}$
  - Transformation: f_6 is to make sure that the intended start time st satisfies the latest time constraints (mst_T, nst_T, cst_T), i.e. $st \in cst\_T$ or $st \in mst\_T \cap \neg nst\_T$. If the satisfiability of st cannot be determined before st, then $sebStatus_1 \leftarrow false$, because the output behavior should not start if the controller is not sure whether the start time will lead to hazard.
  - Duration: No requirement on the expected duration.
  - Output: $sebStatus_1$ is to denote whether the start time st satisfies the time constraints.
  - Input: st is the intended start time passed from MA_5; (mst_T, nst_T, cst_T) is the latest time constraints for the intended start time.
  - Trigger: NA.
  - Guard: $\{stage = Stage1\} \wedge \{sebStatus = true\}$, which is inherited from the monitor theme at Stage 1 as a whole.

- $\boldsymbol{MA_7}$
  - Transformation: f_7 is to make sure that the planned dynamic trajectory $y(t)$ satisfies the performance constraints of the intended behavior pc and $y(st)$ is consistent with the $In - B$ at st. If the satisfiability of $y(t)$ cannot be determined before st, then $sebStatus_2 \leftarrow false$, because the output behavior should not start if the controller is not sure whether the start time will lead to hazard.
  - Duration: No requirement on the expected duration.
  - Output: $sebStatus_2$ is to denote whether $y(t)$ satisfies pc.
  - Input: $y(t)$, where $t \in [st, sp]$, is the planned dynamic trajectory that comes from MA_3; pc is the latest performance constraints of the intended behavior; E_3 and CP_3 are sets of information from the environment and the controlled process respectively that is used to generate the SEB; $In - B$ is the projected trajectory of the in-behavior.

- Trigger: NA.
- Guard: $\{stage = Stage1\} \wedge \{sebStatus = true\}$, which is inherited from the monitor theme at Stage 1 as a whole. In addition, if $In-B$ is deviated, there is no way to predict $y(st)$. Therefore, no deviation between $In-B$ and $\widetilde{y}$ is also a guard condition, where $\widetilde{y}$ is the observed output value in real time.

- $\boldsymbol{MA_8}$
  - Transformation: f_8 is to make sure that the intended stop time sp will not lead to the violation of the performance constraints of the out-behavior pc^o. If the satisfiability of sp cannot be determined before st, then $sebStatus_3 \leftarrow false$, because the output behavior should not start if the controller is not sure whether the stop time will lead to hazard.
  - Duration: No requirement on the expected time duration.
  - Output: $sebStatus_3$ is to denote whether pc^o will be violated if the intended output beahvior stops at time sp.
  - Input: $y(sp)$ is the value of the SEB at time sp from MA_3; pc^o is the latest performance constraints of the out-behavior from MA_4; E_3 is a set of information from the environment that is used to generate the SEB.
  - Trigger: NA.
  - Guard: $\{stage = Stage1\} \wedge \{sebStatus = true\}$, which is inherited from the monitor theme at Stage 1 as a whole.

In summary, the monitor theme at Stage 1 can be represented as an action in Fig.19, where $E = \{E_1, E_2, E_3\}$ and $CP = \{CP_1, CP_2, CP_3\}$. The sources of the all the external interactions with other themes, the enabling actions and the elements out of the design scope are specified in Fig.20.

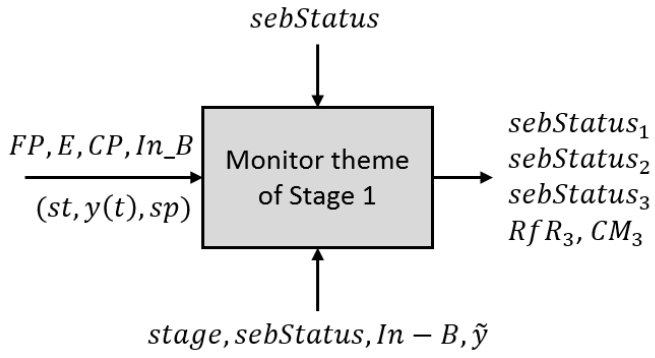


Figure 19: The monitor theme at Stage 1 represented as an action. The internal interactions are hidden within the grey box.

The monitor theme at Stage 2. We define three main actions for the monitor theme (Fig.21) at Stage 2 in accordance with the three transformations f_9, f_{11} and f_{12} identified in the workflow of Fig.6. Now we explain the main actions one by one.

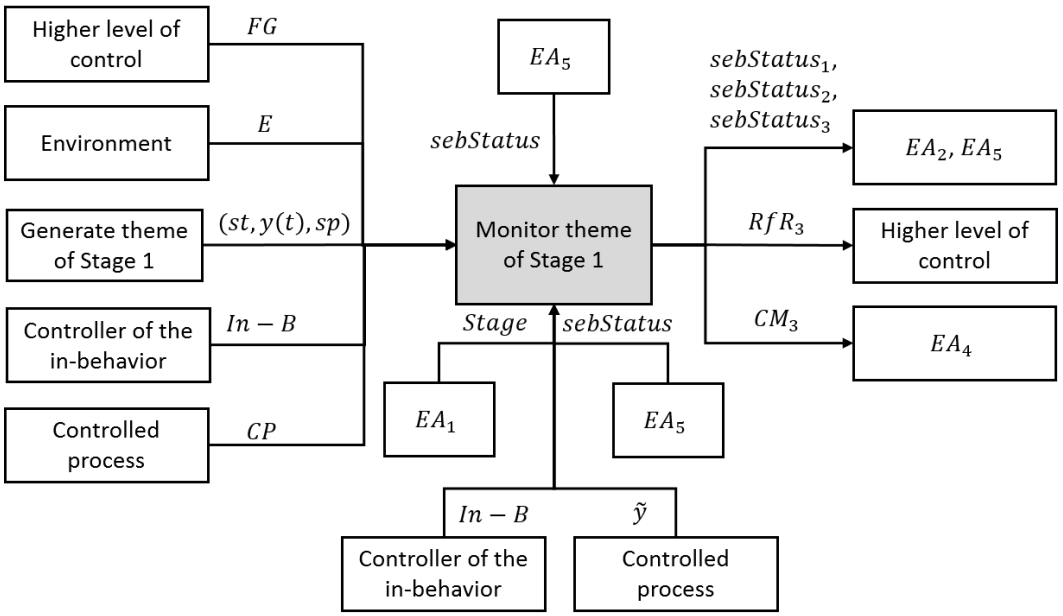


Figure 20: The external interactions of the monitor theme at Stage 1 as a whole.

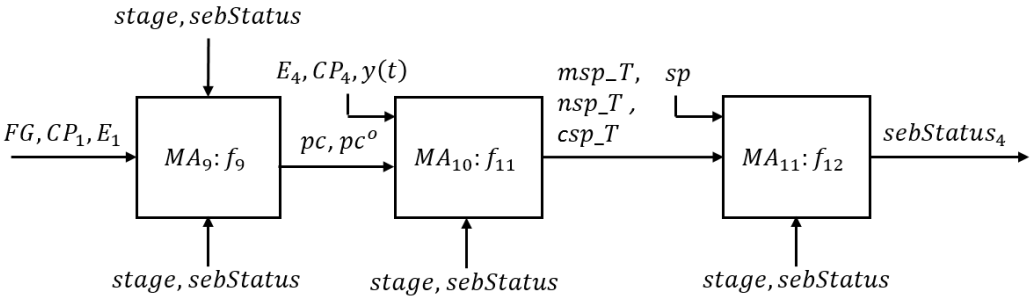


Figure 21: The main actions of the monitor theme at Stage 2.

- $\boldsymbol{MA_9}$
  - Transformation: f_9 is similar to f_1 as both are to generate the performance constraints. The only difference is that the performance constraints of the in-behavior (pc^i) is dropped at Stage 2 because the in-behavior has already stopped and it is the intended output behavior in progress.
  - Duration: The expected time duration e_9 must be short enough so that there will be enough time for MA_{10} to generate msp_T and nsp_T before sp passes and msp_T expires. However, there is always a chance that e_9 is too long that MA_{10} eventually generates msp_T and nsp_T after sp passes or msp_T expires. Therefore, a long enough look-ahead time must be selected to decide msp_T early to lower the likelihood of an expired msp_T or a passed sp to an acceptable level.
  - Output: Similar to f_1 but without pc^i, i.e. (pc, pc^o).
  - Input: The functional goal FG to calculate pc_0 which goes into pc; a set of information about the elements in the environment that determines pc and pc^o. Note that as long as MA_9 has been triggered and the guard

condition is satisfied, MA_9 will always transform the new E_1 and FG_1, which is consistent with Fig.6 that the change of E_1 or FG_1 triggers a new workflow.

- Trigger: Individually, there is no trigger event for MA_9. But because the monitor theme starts with MA_9, MA_9 inherited the trigger event defined for the monitor theme to start the monitor theme as a whole. Therefore, the trigger of MA_9 is $\{stage : Stage1 \rightarrow Stage2\} \vee \{sebStatus : false \rightarrow true\}$. Refer to the last section for the rationale.
- Guard: Individually, there is no the guard condition for MA_9. But the guard condition of MA_9 is inherited from the monitor theme as a whole, which is $\{stage = Stage2\} \wedge \{sebStatus = true\}$.

- $\boldsymbol{MA_{10}}$
  - Transformation: f_{11} is to calculate the timing constraints (msp_T, nsp_T, csp_T) for the output behavior to stop. If there is internal conflict between msp_T and nsp_T, then assign $false$ to (msp_T, nsp_T, csp_T), in order to enter the generate theme so that such a conflict can be resolved by a new $y(T)$. The reason that MA_{10} does not send RfR to resolve such conflict is because a new $y(T)$ itself can resolve the conflict.
  - Duration: The expected time duration e_{10} must be short enough so that msp_T and nsp_T are generated before sp passes and msp_T expires. However, there is always a chance that e_{10} is too long that msp_T and nsp_T are eventually generated after sp passes or msp_T expires. Therefore, a long enough look-ahead time must be selected to decide msp_T early to lower the likelihood of an expired msp_T or a passed sp to an acceptable level.
  - Output: The timing constraints (msp_T, nsp_T, csp_T) for the output behavior to stop.
  - Input: pc^o and pc are the performance constraints for the out-behavior and the intended behavior. $y(t)$ is the planned trajectory of the intended behavior; E_4 and CP_4 is a set of information that is used to generate time window for the stop time. Note that $y(t)$ can be defined with different level of precision. The more precise $y(t)$ is required to be, the less room for uncertainty, but the more accurate to predict (msp_T, nsp_T, csp_T) with $y(t)$. For example, the 4d waypoint instruction is more flexible and robust than the speed-based instruction but harder to predict the airplane's position. This is design decision that can only be made on a case-by-case basis.
  - Trigger: NA.
  - Guard: The guard condition of the monitor theme as a whole is inherited by MA_{10}, i.e. $\{stage = Stage2\} \wedge \{sebStatus = true\}$. Furthermore, using $y(t)$ to calculate (msp_T, nsp_T, csp_T) requires that the real output behavior $\tilde{y}(t)$ adheres to the intended output behavior $y(t)$, i.e. $\tilde{y}(t) \in y(t)$. Note that the reason $\in$ is used here is because the intended output behavior does not have to be a single value at

each time. It is actually most likely to be an acceptable value range at each time because of both the epistemic uncertainty in top-down system design and the aleatoric uncertainty of the actual system. Therefore, the guard condition of MA_{10} is $\{stage = Stage2\} \wedge \{sebStatus = true\} \wedge \{\tilde{y}(t) \in y(t)\}$.

- $\boldsymbol{MA_{11}}$
  - Transformation: f_{12} is to compare the planned stop time with the latest respective timing constraints. Intuitively, it is to make sure that msp_T does not expired during $y(t)$ and the stop time is not within nsp_T. Formally, it can be represented as $sp \in csp\_T \cup msp\_T$ and $sp \notin nsp\_T$. In addition, if the satisfiability of sp cannot be determined before sp passes, then $sebStatus_3 \leftarrow false$, because the output behavior should not stop if the controller is not sure whether the stop time will lead to hazard.
  - Duration: None.
  - Output: $sebStatus_4$ denotes whether the planned stop time sp satisfies the respective timing constraints.
  - Input: sp is the planned stop time from MA_3 or MA_{14}; (msp_T, nsp_T, csp_T) is the latest time constraints for the intended stop time.
  - Trigger: NA.
  - Guard: $\{stage = Stage2\} \wedge \{sebStatus = true\}$, which is inherited from the monitor theme at Stage 2 as a whole.

In summary, the monitor theme at Stage 2 can be represented as an action in Fig.22, where $E = \{E_1, E_4\}$ and $CP = \{CP_1, CP_4\}$. The sources of the all the external interactions with other themes, the enabling actions and the elements out of the design scope are specified in Fig.23.

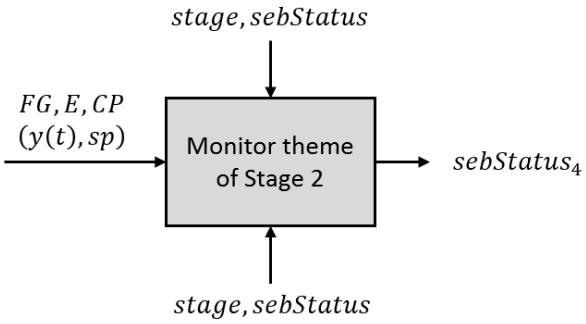


Figure 22: The monitor theme at Stage 2 represented as an action. The internal interactions are hidden within the grey box.

The generate theme at Stage 2. We define three main actions for the monitor theme (Fig.24) at Stage 2 in accordance with the three transformations f_9, f_{13} and $f_{14|15}$ identified in the workflow of Fig.7. Now we explain the main actions one by one.

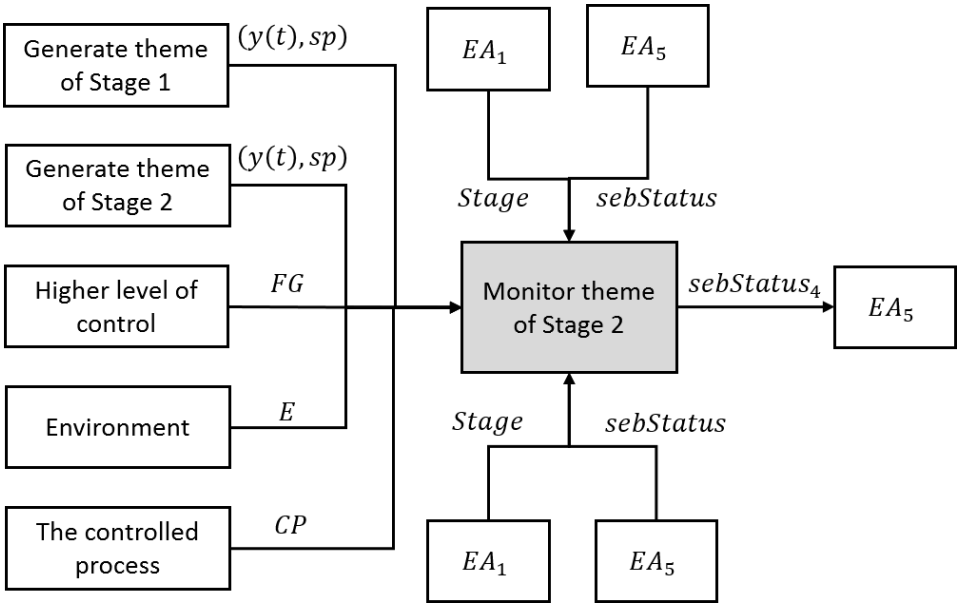


Figure 23: The external interactions of the monitor theme at Stage 2 as a whole. Note that $(y(t), sp)$ is only from the generate theme of Stage 1 at the beginning of Stage 2.

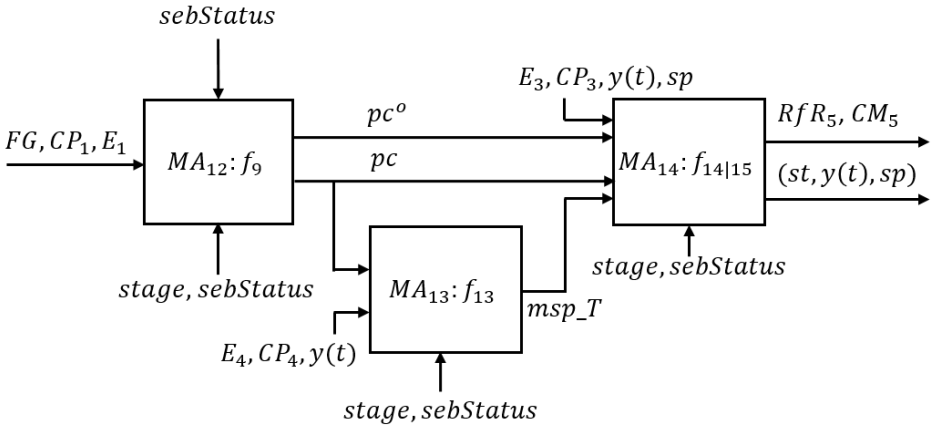


Figure 24: The main actions of the generate theme at Stage 2.

- $\boldsymbol{MA_{12}}$
  - Transformation: Same as MA_9.
  - Duration: The expected time duration e_{12} must be short enough so that there will be enough time for MA_{14} to generate the new SEB before sp passes, and for MA_{13} to generate msp_T before msp_T expires. However, there is always a chance that e_{12} is too long that pc and pc^o are generated after sp passes, or MA_{13} generates msp_T after msp_T expires or . Therefore, a long enough look-ahead time must be selected to decide msp_T early to lower the likelihood of an expired msp_T or a passed sp to an acceptable level.
  - Output: Same as MA_9.
  - Input: Same as MA_9.

- Trigger: Inherited from the generate theme at Stage 2 as a whole, $\{sebStatus : true \rightarrow false\}$.
- Guard: Inherited from the generate theme at Stage 2 as a whole, $\{stage = Stage2\} \wedge \{sebStatus = false\}$.

- $\boldsymbol{MA_{13}}$
  - Transformation: f_{13} is to calculate the must-stop time window in case that a new satisfactory SEB cannot be found by MA_{14} in time.
  - Duration: The expected time duration e_{13} must be short enough so that msp_T will be generated before msp_T expires and sp passes (MA_{14} cannot start finding the new SEB without a valid msp_T). However, there is always a chance that e_{13} is too long that eventually is generated msp_T after msp_T expires or sp passes. Therefore, a long enough look-ahead time must be selected to decide msp_T early to lower the likelihood of an expired msp_T or a passed sp to an acceptable level.
  - Output: msp_T is the time window to stop the current SEB.
  - Input: Same as MA_{10}.
  - Trigger: NA.
  - Guard: The guard condition is inherited from the generate theme at Stage 2 as a whole. Therefore, the guard condition for MA_{13} is $\{stage = Stage2\} \wedge \{sebStatus = false\}$.

- $\boldsymbol{MA_{14}}$
  - Transformation: $f_{14|15}$ is to calculate a new SEB $(st, y(t), sp)$ that will satisfy pc and pc^o. In the case that a satisfactory SEB cannot be found, RfR_5 should be sent to the higher level of control to adjust the functional goal. If a satisfactory SEB cannot be found time $T1$ before msp_T expires or $T1$ before sp passes, the controller stops the main action and enters the contingency mode by setting CM_5 to $true$.
  - Duration: None.
  - Output: The intended output behavior $(st, y(t), sp)$. The RfR_5 is the request to the higher level of control for a new functional goal (FG) when no satisfactory SEB can be found. CM_5 is the signal for EA_4 to take over when msp_T is about to expire.
  - Input: (pc, pc^o) are the performance constraints used to calculate the intended SEB; sp is the previously defined stop time; $y(t)$ is the SEB, based on which the initial condition of the new SEB is selected; E_3 and CP_3 are the sets of information from the environment and the controlled process respectively that is used to generate the SEB; msp_T is used to decide when to enter the contingency mode.
  - Trigger: NA.

– Guard: Part of the guard condition is inherited from the generate theme at Stage 2 as a whole, $\{stage = Stage2\} \wedge \{sebStatus = false\}$. Furthermore, the new SEB must be found before msp_T expires, hence $t_c \in msp\_T$ is also part of the guard condition. Therefore, the guard condition of MA_{14} is $\{stage = Stage2\} \wedge \{sebStatus = false\} \wedge \{t_c \in msp\_T\}$.

In summary, the generate theme at Stage 2 can be represented as an action in Fig.25, where $E = \{E_1, E_3, E_4\}$ and $CP = \{CP_1, CP_3, CP_4\}$. The sources of all the external interactions with other themes, the enabling actions and the elements out of the design scope are specified in Fig.26.

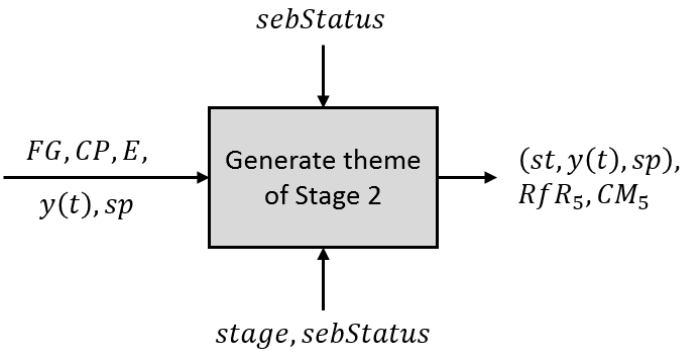


Figure 25: The generate theme at Stage 2 represented as an action. The internal interactions are hidden within the grey box.

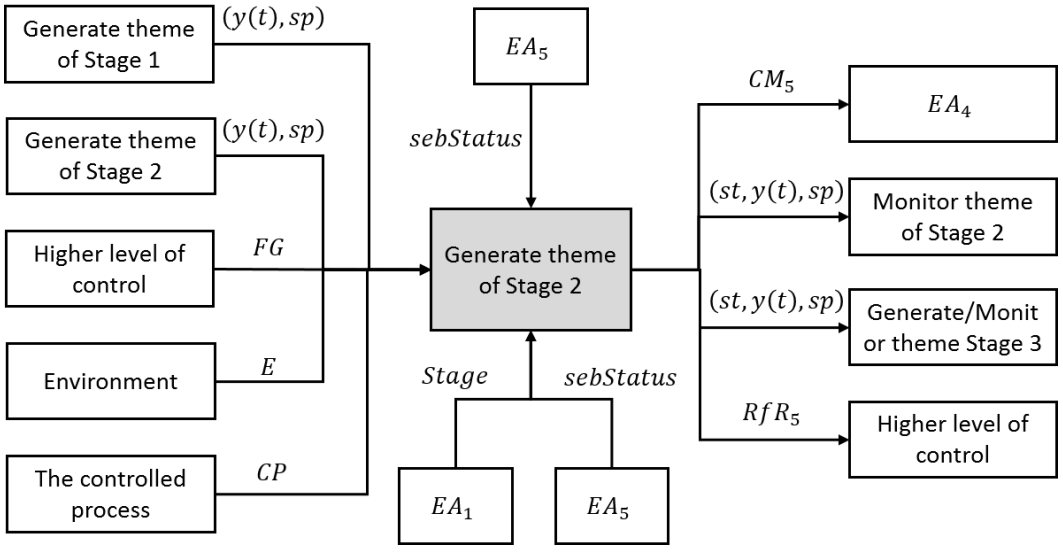


Figure 26: The external interactions of the generate theme at Stage 2 as a whole.

The generate theme at Stage 3. The individual actions of Stage 3 are almost the same as the actions of Stage 2 except the functional goal is achieved before Stage 3. This difference has two implications: (1) is that FG is not the input any more and (2) is that no RfR will be sent if a new SEB cannot be found. *For this reason, we simply list all the main actions for Stage 3 without detailed explanation.*

We define three main actions for the monitor theme (Fig.27) at Stage 3 in accordance with the three transformations f_{10}, f_{13} and $f_{14|15}$ identified in the workflow of Fig.7.

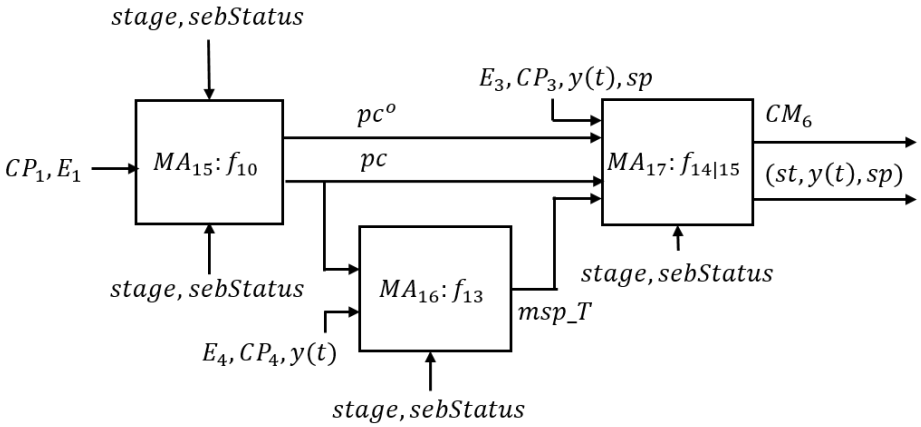


Figure 27: The main actions of the generate theme at Stage 3.

- $\boldsymbol{MA_{15}}$
  - Transformation: f_{10}.
  - Duration: Refer MA_{12}.
  - Output:Refer MA_{12}.
  - Input: E and CP.
  - Trigger: $\{stage : Stage2 \rightarrow Stage3\} \vee \{sebStatus : true \rightarrow false\}$.
  - Guard: $\{stage = Stage3\} \wedge \{sebStatus = false\}$.
- $\boldsymbol{MA_{16}}$
  - Transformation: Refer MA_{13}.
  - Duration: Refer MA_{13}.
  - Output: Refer MA_{13}.
  - Input: Refer MA_{13}.
  - Trigger: NA.
  - Guard: $\{stage = Stage3\} \wedge \{sebStatus = false\}$.
- $\boldsymbol{MA_{17}}$
  - Transformation: Refer MA_{14}.
  - Duration: Refer MA_{14}.
  - Output: $(st, y(t), sp)$ and CM_6.
  - Input: Refer MA_{14}.
  - Trigger: NA.
  - Guard: $\{stage = Stage3\} \wedge \{sebStatus = false\}$.

In summary, the generate theme at Stage 3 can be represented as an action in Fig.28, where $E = \{E_1, E_3, E_4\}$ and $CP = \{CP_1, CP_3, CP_4\}$. The sources of the all the external interactions with other themes, the enabling actions and the elements out of the design scope are specified in Fig.29.

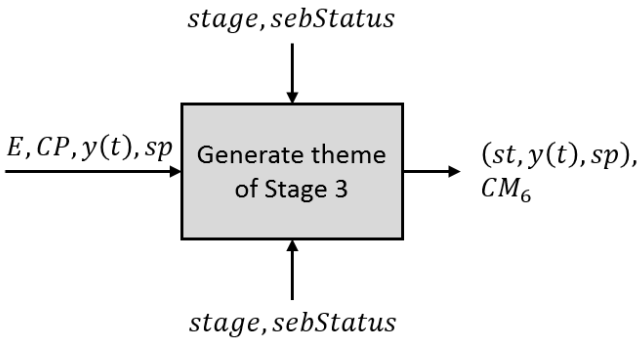


Figure 28: The generate theme at Stage 3 represented as an action. The internal interactions are hidden within the grey box.

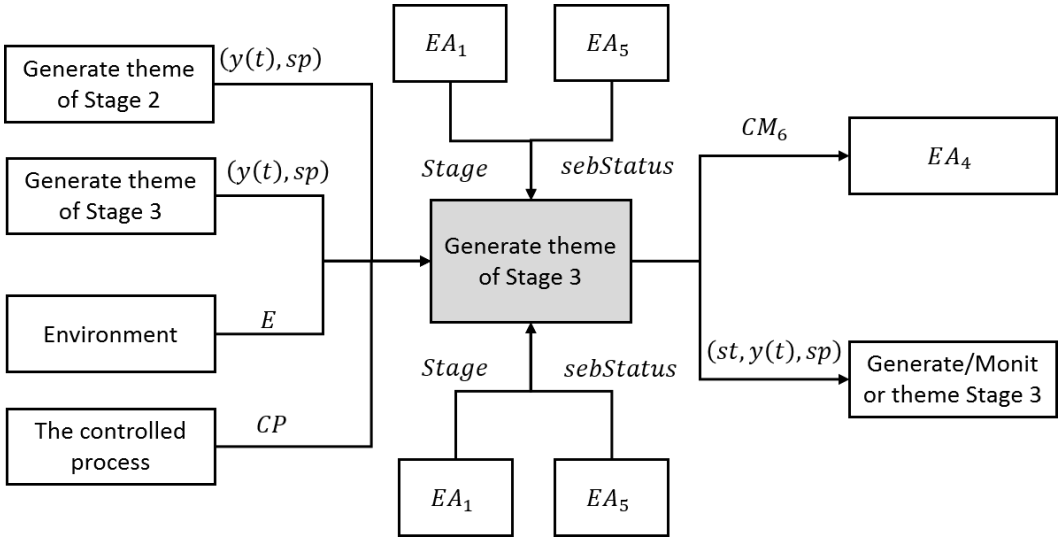


Figure 29: The external interactions of the generate theme at Stage 3 as a whole.

The monitor theme at Stage 3. We define three main actions for the monitor theme (Fig.30) at Stage 3 in accordance with the three transformations f_{10}, f_{11} and f_{12} identified in the workflow of Fig.6. Now we explain the main actions one by one.

- $\boldsymbol{MA_{18}}$
  - Transformation: Refer to MA_9.
  - Duration: Refer to MA_9.
  - Output: pc and pc^o.
  - Input: E and CP.
  - Trigger: $\{stage : Stage2 \rightarrow Stage3\} \vee \{sebStatus : false \rightarrow true\}$.

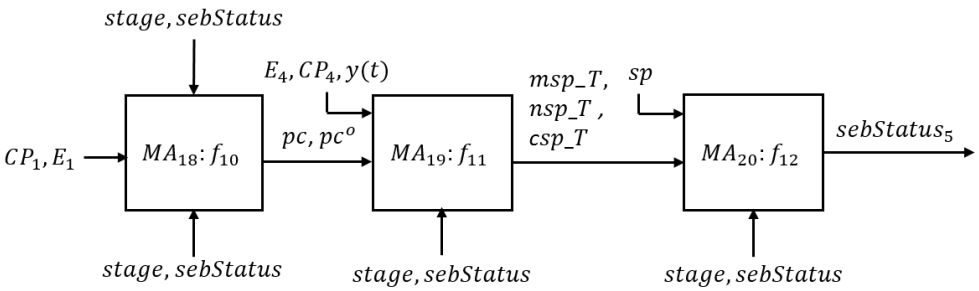


Figure 30: The main actions of the monitor theme at Stage 3.

- Guard: $\{stage = Stage3\} \wedge \{sebStatus = true\}$.

- $\boldsymbol{MA_{19}}$
  - Transformation: Refer to MA_{10}.
  - Duration: Refer to MA_{10}.
  - Output: (msp_T, nsp_T, csp_T).
  - Input: pc^o, pc, E, CP and $y(t)$.
  - Trigger: NA.
  - Guard: $\{stage = Stage3\} \wedge \{sebStatus = true\} \wedge \{\tilde{y}(t) \in y(t)\}$.

- $\boldsymbol{MA_{20}}$
  - Transformation: Refer to MA_{11}.
  - Duration: Refer to MA_{11}.
  - Output: $sebStatus_5$.
  - Input: (msp_T, nsp_T, csp_T) and sp.
  - Trigger: NA.
  - Guard: $\{stage = Stage3\} \wedge \{sebStatus = true\}$.

In summary, the monitor theme at Stage 3 can be represented as an action in Fig.31, where $E = \{E_1, E_4\}$ and $CP = \{CP_1, CP_4\}$. The sources of the all the external interactions with other themes, the enabling actions and the elements out of the design scope are specified in Fig.32.

3 Task 2: Plan the control reference

This section is highly coupled with Method 2. We will provide a paper reference here once we have a complete paper about Method 2.

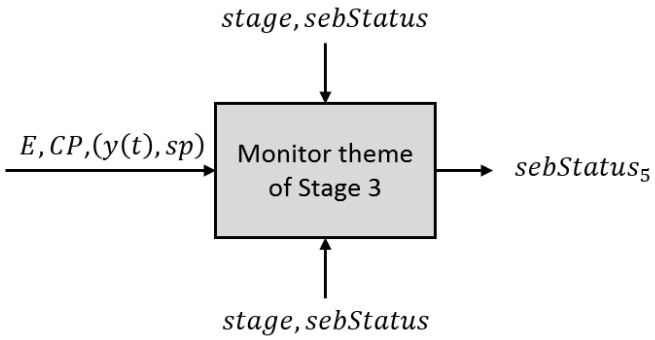


Figure 31: The monitor theme at Stage 3 represented as an action. The internal interactions are hidden within the grey box.

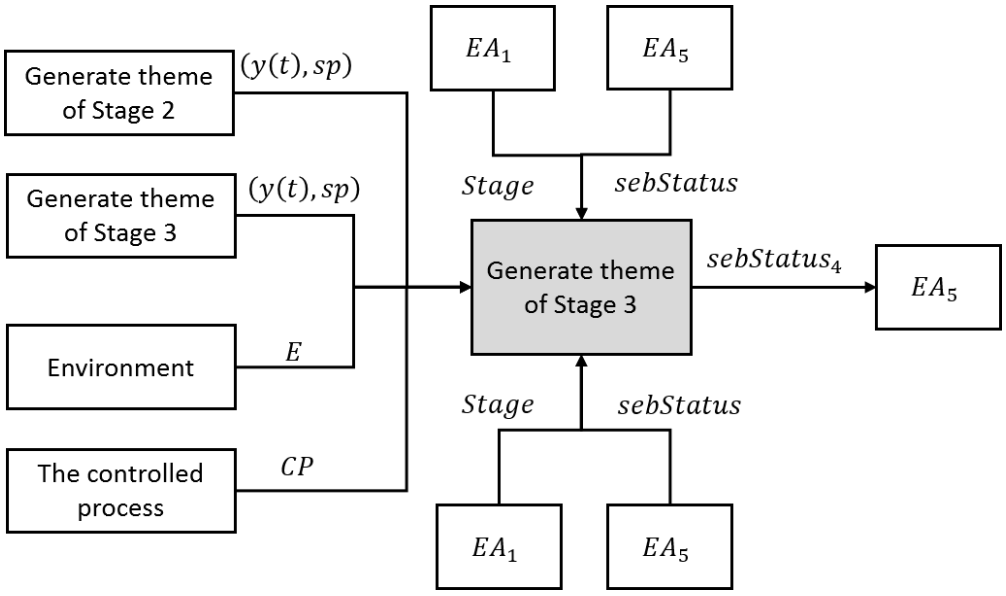


Figure 32: The external interactions of the monitor theme at Stage 3 as a whole.

3.1 Fundamentals

3.1.1 Mathematical preparation.

In general, the process model f and the control algorithm g can be represented as below, where r denotes the control reference. As defined previously, SEB is just an alias of $y(T) = \{y(t)|t \in T = [st, sp]\}$. In this section, we use $y(T)$ to simplify the mathematical expression of SEB. The uppercase T suggests that $y(T)$ is a set of $y(t)$ within the time interval T.

$$\begin{cases} \text{Process model: } (u(t), x(t), p) \xrightarrow{f} (\dot{x}, y(t))^T \\ \text{Control algorithm: } (r, x(t), p) \xrightarrow{g} u(t) \end{cases} \tag{5}$$

Plug g into f, we get:

$$(r, x(t), p) \xrightarrow{f \cdot g} (\dot{x}, y(t))^T$$

For an extended period of time, we can get the expression below. It implies given the process model and control algorithm, the dynamic trajectory of the output variable can be predicted by the initial condition, the parameter and the control reference.

$$(r(t_e), x(t_s), p) \xrightarrow{f \cdot g} (x(T), y(T))^T \text{ where } T = [t_s, t_e] \tag{6}$$

Focusing on the output, we can get (7) below by inverting $(f \cdot g)$. Because it is possible that T is chopped up into several segments and a control reference is generated for each segment, t_s and t_e are used to denote in general the start/end time of a segment, rather than st and sp which is the start/stop time of the entire SEB.

$$(y(T), x(t_s), p) \xrightarrow{(f \cdot g)^{-1}} r(t_e) \tag{7}$$

Note that (7) is not necessarily valid mathematically because $(f \cdot g)^{-1}$ might not exist. But it reveals the fact that the desired output behavior, f, g, the initial condition and the parameter all together determine the control reference. As a result, a more general and precise depiction of such relationship is (8), where the arrow simply means a mapping (not necessarily a function) exists between the two sides.

$$(y(T), x(t_s), p, f, g) \rightarrow r(t_e) \tag{8}$$

Finally, the parameter p can be defined as a single value or a value set (considering uncertainty). For the former case, change of the parameter is obvious. For the latter case, change of the parameter means the new value set includes elements that are not in the current value set, or the single value observed in real time in not included in the current value set; if the single value in real time varies but still within the current value set, it is not considered as a change.

3.1.2 Problem formulation

The value aspect. The goal of this task is to create control references to obtain the desired SEB decided in Task 1. Specifically, the control references can be created in the following three ways:

- Single control reference: only one control reference is created for the entire SEB.
- All at once: control All control references are created at once for the entire SEB.
- In batches: the control references are created in batches, each of which is for a segment of the SEB. The number of control reference in each batch can be 1, which in plain Ennglish is to say that the control reference is created one by one.

The problem formulation must include all the three different ways. For this purpose. An overall formulation of this task is generated in Fig.33. The desired SEB is “chopped up” into n segments, and the control references are created for

each segment *one at a time*. For the ith segment, a specific amount M_i of control references are created *at the same time* to achieve the desired SEB for the ith segment. In this way, all the three ways of creating the control references are included, and therefore it is a general one and can be used to derive a general set of actions for Task 2.

- Single: $n = 1, M_1 = 1$.
- All at once: $n = 1$ and $M_1 \geq 2$.
- One (batch) by one (batch): $n \geq 2$ and $M_i \geq 1$.

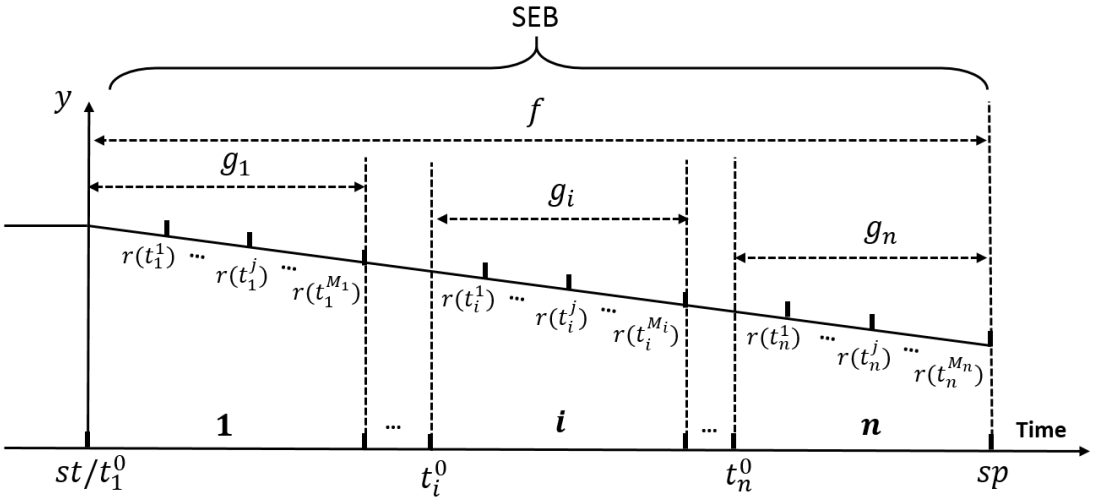


Figure 33: A general problem formulation of "planning the control reference".

Furthermore, on one hand, we prescribe that the control algorithm and the parameter are the same for all the control references of each segment. Note that p may change within one segment due to aleatoric uncertainty, and will be addressed accordingly. But such change is not anticipated when the control references are generated in the first place.

On the other hand, the control algorithm g and the parameter p may vary from segment to segment. Especially for the control algorithm, it is possible that multiple control algorithms are designed for the same process model depending on the performance requirement and the parameter of the controlled process. For example, when a precise distance between two cars is required, the Adaptive Cruise Control is usually applied to replace human driver to control the speed of the trailing car, which is an example that the performance requirement determines the different choices of the control algorithms over the same controlled process; a rocket usually switches to different flight control algorithms as the fuel's burning changes the overall weight of the rocket, which is an example that the different parameter values of the controlled process drives the change of applicability of the control algorithm. In fact, the phenomena of multiple control algorithms for the same controlled process are generally known as "mode" in general engineering design community. Similarly, the applicability of the control algorithm is also affected by the initial conditions of the controlled process.

From the perspective of feasibility, the initial conditions may affect whether the control algorithm can achieve the desired performance. Furthermore, considering the control algorithm and the controlled process as a system, the initial conditions may also affect the stability of the system as a whole. Therefore, in the most general sense, the control algorithm, as depicted in (9), is determined by the desired output behavior, the parameter and the initial condition of the controlled process.

$$(y(T), p, x_0) \rightarrow g, \text{ if } p \in P_g \text{ and } x_0 \in X_g \tag{9}$$

Furthermore, there might be multiple f for the same process under control. For example, the same amount of gas might generate (dramatically) different horsepower for the same engine in different operational temperatures. If the engine is supposed to work in a wide range temperatures, multiple f must be designed for the engine operations in different situations and applied correctly according to the specific operational temperatures. However, in this early work **we assume** the controlled process only has one f[5], meaning the process model is either applicable or not and there is no switching between different process models. The multiple-process-model scenario will be addressed in the future work.

Finally, with all the definition of the problem formulation, the control references of the ith segment are created in the way explained in (Fig.34).

- First, there are M_i control references to be created *at the same time* for the ith segment. These M_i control references yield M_i sub-segments for the ith segment. Each row in (Fig.34) corresponds to a sub-segment of the ith segment.
- Second, the initial state of the each sub-segment is the final state of the previous sub-segment. The initial state of the first sub-segment $t_i^0 \sim t_i^1$ comes from the final state of the $(i-1)$th segment.
- Third, given the initial state of the first sub-segment $x(t_i^0)$, (8) can be used to compute $r(t_i^1)$, where t_s and t_e in (8) are t_i^0 and t_i^1 respectively in this case.
- Fourth, with $r(t_i^1)$, (6) can be used to predict $x(T_i^1)$ (where $T_i^1 = [t_i^0, t_i^1]$). The final state of the first sub-segment $x(t_i^1)$ can be retrieved from $x(T_i^1)$ as the initial state for the second sub-segment.

As a result, all the control references can be created by applying the same process to all the sub-segments recursively. Mathematically, the process of creating all the control references for the ith segment can be expressed in (10).

$$(y(T_i), x(t_i^0), p_i, f, g_i) \rightarrow R_i \tag{10}$$

[5]Note that f is the transformation of the process model. One f does not mean one process model, because the process model also include Explicit Constraints and they can still change even f stays the same. Refer to Method 2 for details.

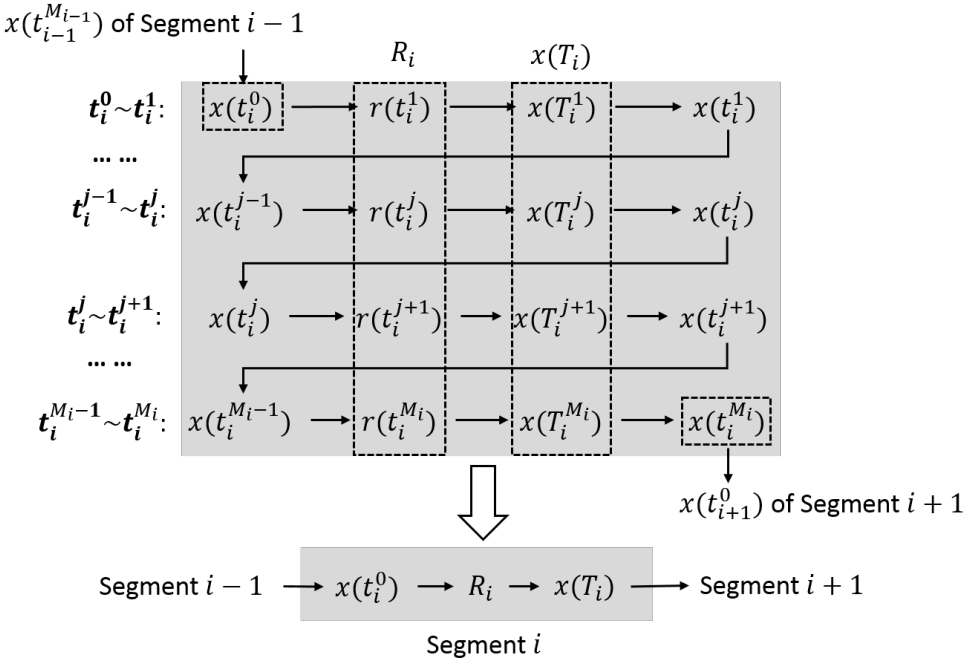


Figure 34: The process to create the control references for the ith segment.

where $y(T_i)$ is the ith segment of the SEB; p_i and g_i are the parameter and the control algorithm for the ith segment; $x(t_i^0)$ is the initial state of the ith segment and comes from the final state of the $(i-1)$th segment; R_i is the set of all the control references of the ith segment, $R_i = \{r(t_i^1), ..., r(t_i^{M_i})\}$.

Similarly, given $x(t_i^0)$ and R_i, the trajectory of T_i can also be predicted by (11) as below.

$$(R_i, x(t_i^0), p_i, f, g_i) \rightarrow (\widehat{x}(T_i), \widehat{y}(T_i))^T \text{ where } T_i = [t_i^0, t_i^{M_i}] \tag{11}$$

where $\widehat{x}(t_i^{M_i})$ is the predicted final state of the ith segment and hence is also the initial state of the $(i+1)$th segment.

The timing aspect. First, as shown in Fig.35, there is a general time delay, denoted as $T2$, from the real state of the controlled process $\widetilde{x}(t)$ being observed (i.e. the start point of the dotted arrow) to the real state is being affected (i.e. the start point of the dotted arrow), for example the process delay and sensor delay. As will shown in the following sections, $T1$ (previously defined) and $T2$ will determine the timing when the generate theme is triggered, when the controller sends out the RfR and when the controller enters the contingency mode.

Second, $T2$ is the expected time duration to calculate the control actions and for the control actions to take effect on the controlled process. For the ith segment in Fig.36, $t_i^0 - T1$ is the deadline that $y(T_i)$ has to be decided and $t_i^0 - T2$ is the deadline that R_i must be decided, and hence $T1 - T2$ is the expected time interval to generate the control references R_i. Similarly, for the $(i+1)$th segment, the generation of R_{i+1} must start no later than $t_{i+1}^0 - T1$. However, the calculation of R_{i+1} requires R_i. Because R_i is supposed to be generated before $t_i^0 - T2$, therefore the following must be true:

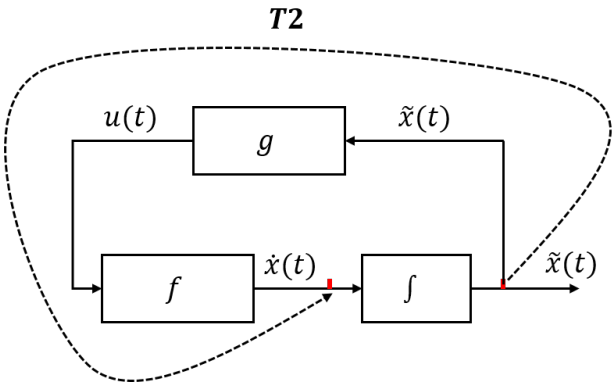


Figure 35: The time delay $T2$.

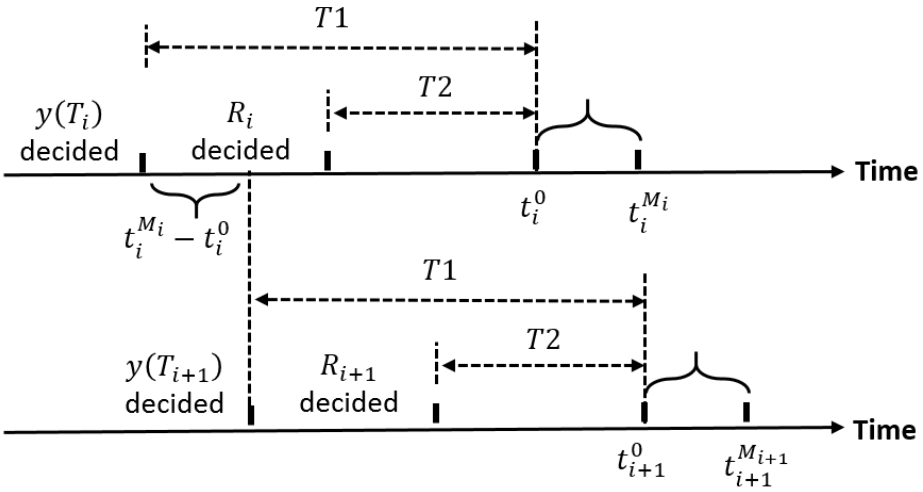


Figure 36: The time interval of each segment must be greater than $T1 - T2$, so that there will be enough time generate R_{i+1} after R_i. This figure shows unsatisfactory case.

$$t_i^0 - T2 < t_{i+1}^0 - T1$$

Because t_{i+1}^0 is the same as $t_i^{M_i}$, therefore we have (21):

$$t_i^{M_i} - t_i^0 > T1 - T2 \tag{12}$$

which means the length of each segment must be greater than the time it takes to generate the control reference. For this reason, Fig.36 is an actually unsatisfactory case.

3.1.3 The workflow of Task 2

Task 2 has three themes: a generate theme, a predict theme and a monitor theme. The generate theme is to create the control references that will achieve the desired SEB (i.e. $y(T)$); the predict theme is to continuously predict the future states of $\widehat{x}(t)$ and $\widehat{y}(t)$ with the generated control references, because aleatoric

uncertainty may create deviation from the planned (evolution) trajectory; the monitor theme is to make sure the process model and the control algorithm are valid and the predicted future states of $\widehat{x}(t)$ and $\widehat{y}(t)$ will not lead to a violation of the SEB.

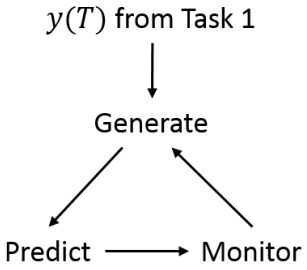


Figure 37: The conceptual relationship between the three themes of Task 2.

The generate theme. The generate theme is to decide the control references so that the desired SEB can be achieved by the controlled process. This statement entails two properties of the control references: *satisfiability* and *feasibility*. The former means that, the output behavior guided by the control references must satisfy the desired SEB, i.e. $\widehat{y}(\overline{T}) \subseteq y(\overline{T})$; the latter means that, the evolution of the controlled process must not violate the process model $(X, U, \dot{X}, Y)$, i.e. (13), where $\overline{T}$ is the segmented timeline of the desired SEB. Note that $\widehat{y}(\overline{T})$ only has the prediction till the last segment whose control references are generated, and it is possible that some of the later segments do not have control references. Therefore, we trivially assign $\emptyset$ to the segments without control references of $\widehat{y}(\overline{T})$ so that $\widehat{y}(\overline{T}) \subseteq y(\overline{T})$ and $\widehat{y}(\overline{T}) \not\subseteq y(\overline{T})$ are mathematically sound.

$$(\widehat{x}(\overline{T}) \not\subseteq X) \vee (\widehat{u}(\overline{T}) \not\subseteq U) \vee (\widehat{y}(\overline{T}) \not\subseteq Y) \vee (\widehat{\dot{x}}(\overline{T}) \not\subseteq \dot{X}) \tag{13}$$

Based on a combination of the truth of the two properties, four scenarios can be derived.

Scenario 1: Satisfactory and feasible. We denote the latest planned segment is the ith segment, this scenario is to generate the control references for the next $(i+1)$th segment. As shown in Fig.38, the end of the ith segment $t_i^{M_i}$ is the start of the $(i+1)$th segment t_{i+1}^0. Before $t_i^{M_i}$ is arrived at, the control references for the $(i+1)$th segment R_{i+1} must be generated. But the question is, how early before $t_i^{M_i}$ should this scenario be triggered?

Fig.38 is the timeline of this scenario. $T1$ (as defined previously) is the expected time interval for ②, ③ and ④; $T2$ is the the expected time (interval) of ③ and ④. This transformations are triggered by $t_c = t_i^{M_i} - T1 - \Delta_{T1}$ becoming true. If it is determined before $t_i^{M_i} - T1$ that no control algorithm or no control references can be found, a RfR_6 (with the intended start time) can be sent to Task 1 for a new SEB (that obviously will not lead to $\widehat{y}(\overline{T}) \not\subseteq y(\overline{T})$). If it is determined after $t_i^{M_i} - T1$ that no control algorithm or no control references can be found, then the controller enters the contingency mode. If upon the transition to this scenario, $t_c < t_i^{M_i} - T1$, then the controller enters the contingency mode

directly[6]. If no control references can be found before $t_i^{M_i} - T2$, the controller enters the contingency mode.

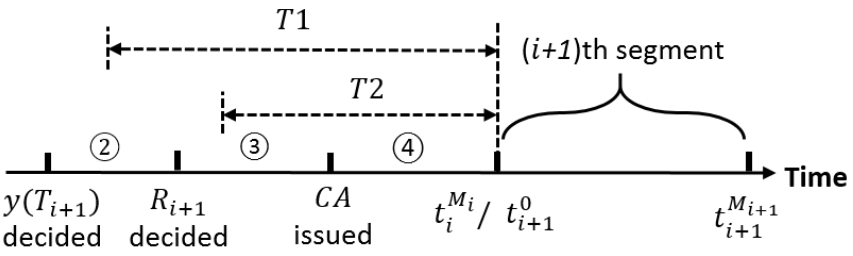


Figure 38: The timeline of Scenario 1 of the generate theme.

As a result, two transformations are identified for this scenario (Fig.39).

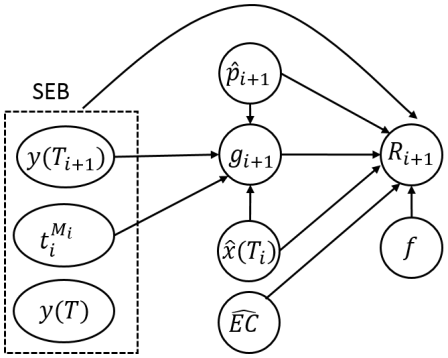


Figure 39: The workflow of Scenario 1 of the generate theme.

$$\begin{cases} (y(T_{i+1}), \widehat{x}(T_i), t_i^{M_i}, \widehat{p}_{i+1}) \xrightarrow{f_{21}} g_{i+1} \\ (y(T_{i+1}), y(T), \widehat{x}(T_i), \widehat{p}_{i+1}, f, g_{i+1}, t_i^{M_i}, \widehat{EC}) \xrightarrow{f_{22}} R_{i+1} \end{cases}.$$

where $0 \leq i \leq n-1$ and $\widehat{EC} = \{\widehat{X}, \widehat{U}, \widehat{P}, \widehat{\dot{X}}, \widehat{Y}\}$.

- f_{21}: This transformation is adopted from (9) that the control algorithm is determined by the desired performance, the parameter and the initial state. $y(T_{i+1})$ is the desired SEB for the $(i+1)$th segment and $\widehat{p}_{i+1}$ is the predicted parameter for the entire $(i+1)$th segment. The initial state is the value of $\widehat{x}(T_i)$ at $t_i^{M_i}$. Note that for the first segment (i.e. $i = 0$), $\widehat{x}^i(t)$ is from the controller of the in-behavior.
- f_{22}: This transformation is based on (10) to calculate the control references R_{i+1} for the $(i+1)$th segment. However, $\widehat{EC}$ and $y(T)$ are not in (10). They are to make sure R_{i+1} will not lead to infeasibility of the SEB after $t_{i+1}^{M^{i+1}}$. In addition, $t_{i+1}^{M^{i+1}}$ must be selected so that (21) is satisfied.

[6]This refers to the possibility that this scenario can be transitioned to from other scenarios, but it is too late to generate new control references.

If a RfR is received from Task 3 that no control actions can be found for a control reference, then this scenario is triggered and applicable by replacing all the control references after the infeasible one with new control references. But since we do not explain how RfR is handled in this version, we will reflect this scenario in the future versions.

Scenario 2: Unsatisfactory and feasible. This scenario is triggered by the detection that the control references will lead to the violation of the desired SEB at some point. Once triggered, this scenario is to resolve the violation before it happens.

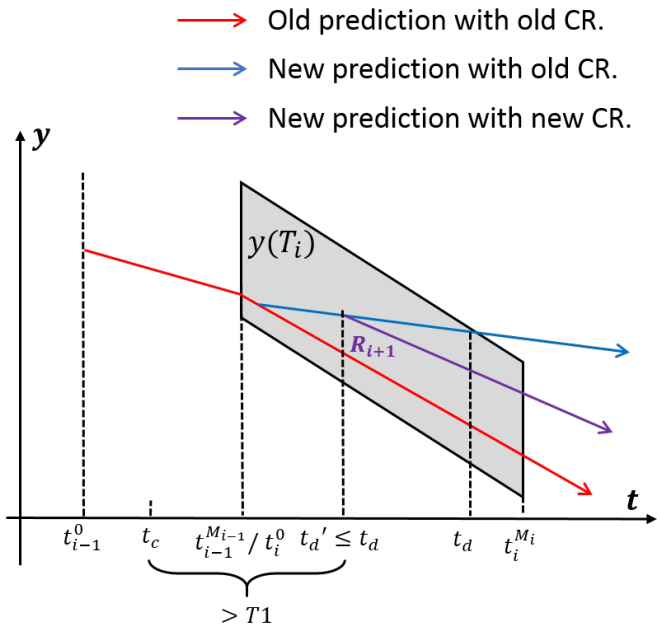


Figure 40: The scenario of $\widehat{y}(\overline{T}) \not\subseteq y(\overline{T})$. The specific violation is caused by the change of the predicted output behavior and resolved by new control references. Because $t_{d'}$ is the point where the "new prediction with new CR" deviates from the "new prediction with old CR", we call $t_{d'}$ the **steering point** hereafter.

Such potential violation can be caused by the change of the predicted output behavior due to uncertainty, or the change of the desired SEB coming from Task 1. To resolve such potential violation, the controller can either change the generated control references to adjust the predict output behavior to fit in the desired SEB, or to change the desired SEB to accommodate the latest predicted output behavior. However, the latter has to be accomplished by Task 1, while the former can be accomplished by the current task. Therefore, it is reasonable to try to adjust the predicted output behavior by changing the control references. If the violation cannot be resolved, a new SEB can be requested from Task 1 as a resolution.

Fig.40 is an example of the violation caused by the change of the predicted output behavior and resolved by adjusting the predicted output behavior with new control references. Originally, the red line is the planned output behavior and satisfies the desired SEB (the grey area). But now the new prediction (the blue line) shows that the output behavior with the current control references will

violate the desired SEB at $t = t_d$. After such violation is detected by the monitor theme, the generate theme has to replace the planned control references with the new ones to steer the output behavior back into the grey area. But not all the planned control references need to be replaced. As shown in the figure, the controller only has to replace the control references after t_d to steer the output behavior back into the grey area. Considering t_d might be too late to steer the output behavior, hence $t_{d'}$ is selected before t_d as the **steering point**. In fact, $t_{d'}$ has to satisfy the constraints of (14). The first is obviously to say that the output behavior must be steered before the violation happens, and the second is to say there must be enough time for the controller to decide the new control references, issue the control actions and for the control action to take effect on the output variable.

$$\begin{cases} t_{d'} \leq t_d \\ t_c + T1 < t_{d'} \end{cases} \tag{14}$$

Note that as long as $t_{d'}$ satisfies the constraints of (14), it does not have to be in the same segment of t_d. Denoting the selected $t_{d'}$ is located in the ith segment, $[t_i^0, t_{d'}]$ is considered the new ith segment, and all the planned control references after $t_{d'}$ will be replaced with new control references R_{i+1}. If $t_{d'}$ cannot be found before $t_d - T1$, or R_{i+1} cannot be found before $t_{d'} - T2$ for a selected $t_{d'}$[7], then RfR_7 can be sent to Task 1 for a new SEB to resolve the predicted violation before t_d. If no new SEB can be found before the deviation happens, the controller enters the contingency mode, which is already defined by EA_3.

Of note, it is possible that the new SEB, while may not resolve the violation, can push t_d to a later time so that new control references can be generated to adjust the predicted output behavior. We **do not consider** this scenario in this work.

As a result, four transformations are defined for the workflow of Scenario 2 (Fig.41).

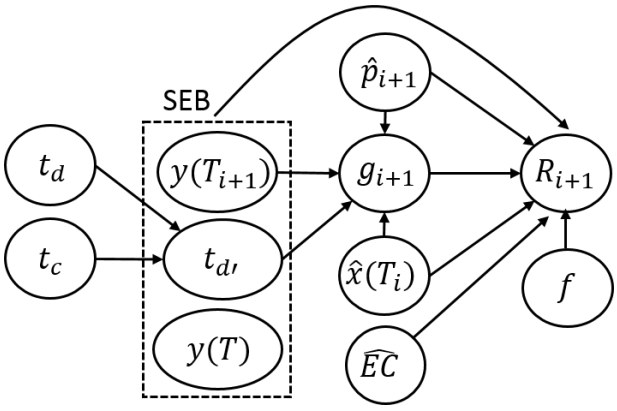


Figure 41: The workflow of the Scenario 2 of the generate theme.

[7] If R_{i+1} cannot be found for $t_{d'}$, it is only possible that a time before $t_{d'}$ can yield a R_{i+1}. But because t_c is already closing in the current $t_{d'}$, an earlier $t_{d'}$ is infeasible. This is why we do not try a different $t_{d'}$ that satisfies (14)

$$\begin{cases} (t_d, t_c) \xrightarrow{f_{23}} t_{d'} \\ (y(T_{i+1}), \widehat{p}_{i+1}, \widehat{x}(T_i), t_{d'}) \xrightarrow{f_{21}} g_{i+1} \\ (y(T_{i+1}), y(T), \widehat{x}(T_i), \widehat{p}_{i+1}, f, g_{i+1}, t_{d'}, \widehat{EC}) \xrightarrow{f_{22}} R_{i+1} \end{cases}$$

where $1 \leq i \leq n$ and $\widehat{EC} = \{\widehat{X}, \widehat{U}, \widehat{P}, \widehat{\dot{X}}, \widehat{Y}\}$.

- f_{23}: This transformation is to select the new start time $t_{d'}$ to satisfy the constraints in (14). t_d is received from the monitor theme.
- f_{21} and f_{22} have already been explained in the previous scenario.

Scenario 3: Satisfactory and infeasible. This scenario is when the dynamic trajectory of the controlled process is predicted to violate the process model at some point. If such violation is detected, a new control algorithm or a new set of control references must be generated before the violation happens.

The activities to address this scenario is similar to Scenario 2, where t_d in this scenario is the earliest time point when the predicted dynamic trajectory of the controlled process $(\widehat{x}(\overline{T}), \widehat{y}(\overline{T}), \widehat{u}(\overline{T}), \widehat{\dot{x}}(\overline{T}))$ violates the process model $(X, Y, U, \dot{X})$. Once t_d is decided for this scenario, all the three transformations of Scenario 2 can be applied to address Scenario 3. The only difference is that, if $t_{d'}$ cannot be found before $t_d - T1$, or R_{i+1} cannot be found before $t_{d'} - T2$, the controller enters the contingency mode, rather than requesting for a new SEB, because a new SEB cannot *directly* resolve this feasibility problem. However, note that it is possible that a new SEB may *indirectly* resolve this feasibility problem by leading to new control algorithm or new control references. But we **do not consider** such scenario in this version.

Scenario 4: Unsatisfactory and infeasible. This scenario is similar to Scenario 2 and Scenario 3 as all of them requires a new control algorithm or a new set of control references to address the problem. Therefore, the three transformations of Scenario 2 can also be applied to this scenario.

The only difference is that t_d of this scenario is the earlier time between t_{d1} and t_{d2}. In addition, because the control references are infeasible, requesting a new SEB will not resolve this problem. Therefore, same as Scenario 3, if $t_{d'}$ cannot be found before $t_d - T1$, or R_{i+1} cannot be found before $t_{d'} - T2$, the controller enters the contingency mode.

The predict theme. The scenarios of the generate theme are determined by the satisfiability and the feasibility, which requires a prediction of $(\widehat{x}(\overline{T}), \widehat{y}(\overline{T}), \widehat{u}(\overline{T}), \widehat{\dot{x}}(\overline{T}))$. We explain how to make such prediction in this section.

In principle, (11) can be adopted to predict $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ for the ith segment. As shown in (15), when the current time t_c is before the ith segment, the latest predicted initial state and the parameters can be used to predict $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$; when the current time is in the ith segment, the current observed state $\widetilde{x}$ and the

parameters $\widetilde{p}$ can be used to calculate the latest prediction of $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$.

$$\begin{cases} (R_i, \widehat{x}(t_i^0), \widehat{p}_i, f, g_i) \rightarrow (\widehat{x}(T_i), \widehat{y}(T_i))^T \text{ where } t_c < t_i^0 \text{ and } T_i = [t_i^0, t_i^{M_i}] \\ (R_i, \widetilde{x}, \widetilde{p}, f, g_i) \rightarrow (\widehat{x}(T_i), \widehat{y}(T_i))^T \text{ where } t_c \in [t_i^0, t_i^{M_i}] \text{ and } T_i = [t_c, t_i^{M_i}] \end{cases} \quad (15)$$

In reality, (15) becomes invalid when deviation happens. Taking the example of the second case of (15), where $t_c \in [t_i^0, t_i^{M_i}]$. First, because the time delay $T2$, a prediction model which "allows the system to use measured signal values at time $t - \delta$ to estimate the signal values at time t" **[?]** is usually used to compensate the time delay. Although such prediction model is used at the next Task 3 to generate the control action, it has impacts on the prediction of $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ at the current Task 2. As shown in Fig.42, at any given time $t_c \in [t_i^0, t_i^{M_i}]$, the real system state (denoted as $\widetilde{x}(t_c)$) is used to predict the system state at $t_c + T2$ (i.e. $\widehat{x}(t_c + T2)$). Then the control action for $t_c + T2$ (i.e. $u(t_c + T2)$) is generated so that it can take effect on the controlled process at $t_c + T2$. Applying such action to all time in the ith segment, the time delay can be compensated and (15) can still be used to predict $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$. In other words, $(\widehat{x}(T_i), \widehat{y}(T_i))^T$ can be predicted at $t_c \in [t_i^0, t_i^{M_i}]$ in (16) , which is the bottom blue line in Fig.42 predicted with $\widetilde{x}|t_c$ and $\widetilde{p}|t_c$.

$$(R_i, \widetilde{x}, \widetilde{p}, f, g_i) \rightarrow (\widehat{x}(T_i), \widehat{y}(T_i))^T \text{ where } t_c \in [t_i^0, t_i^{M_i}] \text{ and } T_i = [t_c, t_i^{M_i}] \quad (16)$$

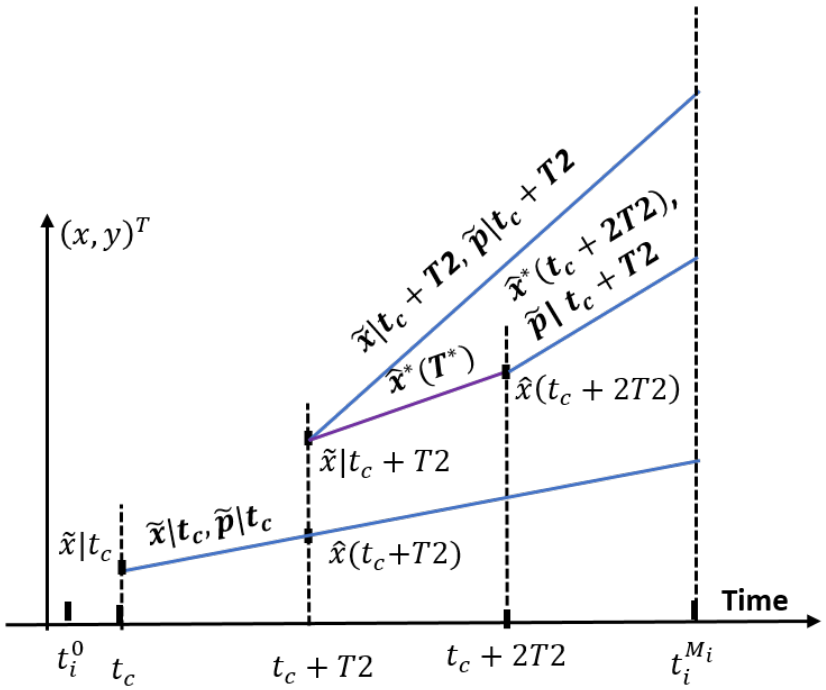


Figure 42: Predicting $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ in the case of time delay $T2$. The blue line is the prediction by adopting (15). Note that $\widehat{x}(t)$ is the prediction made at $t - T2$; $\widetilde{x}|t$ and $\widetilde{p}|t$ are the observed state and parameter the current time t. The bold $\widetilde{x}$ and $\widetilde{p}$ above the blue lines are the state and the parameter used to make the prediction.

"Deviation" in this specific example means the observed states or parameters are different from the previously predicted states or parameters. As shown in Fig.42, $\widehat{x}(t_c + T2)$ is predicted at t_c to generated the control action in order to compensate the time delay $T2$. A deviation is, if at time $t_c + T2$, it is found that real observed state $\widetilde{x}|t_c + T2$ is different from the predicted $\widehat{x}(t_c + T2)$, i.e. $\widetilde{x}|t_c+T2 \notin \widehat{x}(t_c+T2)$. Obviously, (6) is not applicable to $[t_c+T2, t_i^{M_i}]$ as a result of such deviation. Therefore, a new prediction needs to be made for $[t_c + T2, t_i^{M_i}]$. Were (15) still applicable, the prediction can be made in (17), which corresponds to the top blue line in Fig.42.

$$(R_i, \widetilde{x}|t_c + T2, \widetilde{p}|t_c + T2, f, g_i) \rightarrow (\widehat{x}(T_i), \widehat{y}(T_i))^T \text{ where } T_i = [t_c + T2, t_i^{M_i}] \quad (17)$$

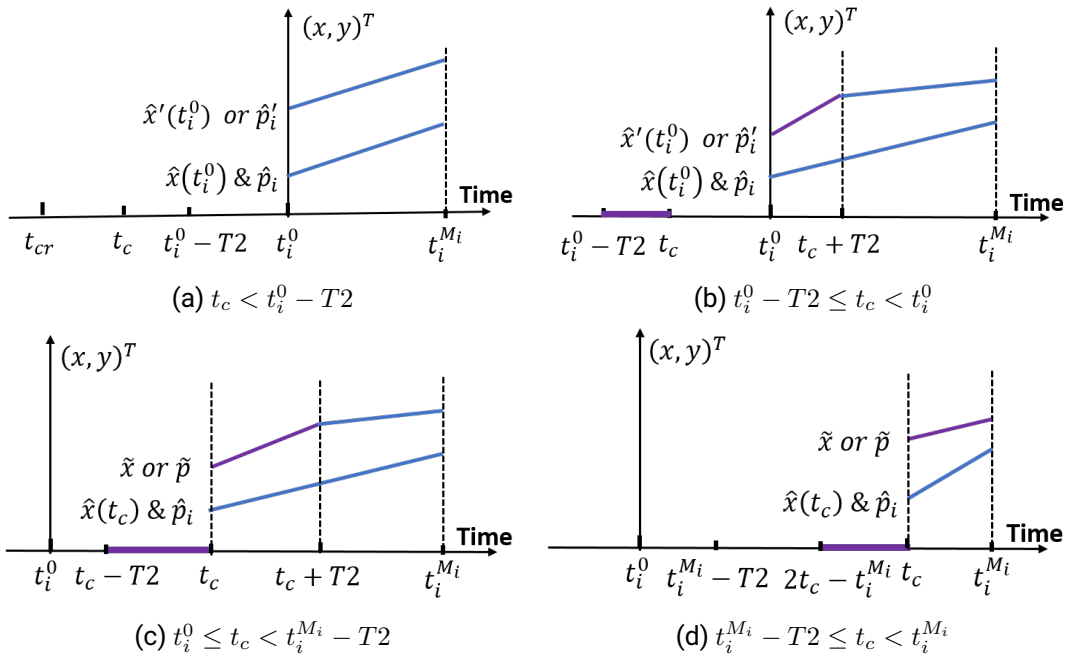
However, (17) is actually incorrect due to the coupling between the deviation and the delay. As shown in Fig.42, all the control actions for $[t_c+T2, t_c+2T2]$ are determined based on the state (and parameter) predictions of $[t_c+T2, t_c+2T2]$ (which is the bottom blue line) made at $[t_c, t_c + T2]$. As a result, when the deviation is detected at $t_c + T2$, the control action for $[t_c + T2, t_c + 2T2]$ based on the outdated predictions of the bottom blue line rather the top blue line. Therefore, (17) i.e. the top blue line, is not applicable to predict $(\widehat{x}(T_i), \widehat{y}(T_i))^T$ for $T_i = [t_c + T2, t_i^{M_i}]$.

As a solution, the prediction of $[t_c + T2, t_c + 2T2]$ must be made based on the all the control actions for $[t_c + T2, t_c + 2T2]$; then the rest of the timeline $[t_c+2T2, t_i^{M_i}]$ can be predicted by applying (15).The prediction of $[t_c+T2, t_c+2T2]$ has to be accomplished by Task 3 as it requires insights about how the control actions are generated. Furthermore, because Task 3 always needs to make such prediction to generate the control action, no extra task is added to Task 3. We denote the prediction made by Task 3 as $\widehat{x}^*(T^*)$. The correct prediction after the deviation should be calculated by (18). As shown in Fig.42, the prediction made by Task 3 is the purple line (where $T^* = [t_c + T2, t_c + 2T2]$). The rest of the prediction is made by applying (15) with the currently observed parameter $\widetilde{p}|t_c + T2$ and the predicted state $\widehat{x}^*(t_c + 2T2)$.

$$(R_i, \widehat{x}^*(T^*), \widehat{x}^*(t_c+2T2), \widetilde{p}|t_c+T2, f, g_i) \rightarrow (\widehat{x}(T_i), \widehat{y}(T_i))^T \text{ where } T_i = [t_c+T2, t_i^{M_i}] \quad (18)$$

In a more general sense, four scenarios (a)∼(d) are developed to make prediction of $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$. As shown in Fig.43, the blue line is the prediction made by (15) and the purple line is the prediction from Task 3. Note that these four scenarios are only applicable to the situation where $T2 < t_i^{M_i} - t_i^0$. The scenarios for $T2 \geq t_i^{M_i} - t_i^0$ will be discussed briefly later.

(a) $t_{cr} \leq t_c < t_i^0 - T2$: Because the ith segment of the desired SEB has not started at this point, $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ are made for the entire $T_i = [t_i^0, t_i^{M_i}]$. Furthermore, deviation in this scenario means the change of the predicted initial state $\widehat{x}(t_i^0)$ or the parameter $\widehat{p}_i$. When the deviation happens, (15) can still be used to update $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ with the updated $\widehat{x}'(t_i^0)$ and/or the parameter $\widehat{p}_i'$ (the upper blue line), because no control action has been determined by Task 3 at this time point. Mathematically, the prediction can



(a) $t_c < t_i^0 - T2$ (b) $t_i^0 - T2 \le t_c < t_i^0$

(c) $t_i^0 \le t_c < t_i^{M_i} - T2$ (d) $t_i^{M_i} - T2 \le t_c < t_i^{M_i}$

Figure 43: The four scenarios to predict $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ for $T2 < t_i^{M_i} - t_i^0$. t_{cr} is the time when the control reference is decided. The blue line is the prediction by (15). The purple line is the prediction from Task 3. t_{cr} is the time when the control reference is issued.

be initially made and then updated (if necessary) by the transformation f_{24}, where $\widehat{x}'(t_i^0)$ and $\widehat{p}_i'$ are the latest predicted value of $x(t_i^0)$ and p_i. Intuitively, as long as $t_{cr} \le t_c < t_s - T2$, f_{24} can always be used to predict the $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ with the latest predicted value of $x(t_i^0)$ and p_i.

$$\begin{cases} (R_i, \widehat{x}(t_i^0), \widehat{p}_i, f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \\ (R_i, \widehat{x}'(t_i^0), \widehat{p}_i', f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \end{cases}$$

where $T = [t_i^0, t_i^{M_i}]$.

(b) $t_i^0 - T2 \le t_c < t_i^0$: Similar to (a), $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ are made for the entire $T_i = [t_i^0, t_i^{M_i}]$ and deviation also means the change of the predicted $\widehat{x}(t_i^0)$ or $\widehat{p}_i$. However, because the states of $[t_i^0, t_c + T2]$ have already been predicted at this point during $[t_i^0 - T2, t_c]$ (the purple segment on the time axis), it is too late to change the control actions (to be) generated based on these predictions. Therefore, the prediction of $[t_i^0, t_c + T2]$ must be made by Task 3 (the purple line, denoted as $\widehat{x}^*(T^*)$) with the new initial state $\widehat{x}'(t_i^0)$ and/or the parameter $\widehat{p}_i'$. Then the rest of the prediction at $[t_c + T2, t_i^{M_i}]$ can be made by (15) with the $\widehat{x}^*(t_c + T2)$ as the initial state and/or the updated parameter $\widehat{p}_i'$ (the upper blue line). Mathematically, the prediction can be initially made by f_{24} and then updated (if necessary) by f_{25}, where $\widehat{x}^*(T^*)$ is the prediction made with $\widehat{x}'(t_i^0)$ and $\widehat{p}_i'$ by Task 3. Intuitively, f_{25} obtains the new prediction by concatenating the prediction of $\widehat{x}^*(T^*)$ from Task 3 with the prediction by f_{24} starting from the time point $t_c + T_2$.

$$\begin{cases}(R_i, \widehat{x}(t_i^0), \widehat{p}_i, f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \\ (R_i, \widehat{x}^*(T^*), \widehat{x}^*(t_c + T2), \widehat{p}_i, f, g_i) \xrightarrow{f_{25}} (\widehat{x}(T_i), \widehat{y}(T_i))^T\end{cases}$$

where $T^* = [t_i^0, t_c + T2]$ and $T_i = [t_i^0, t_i^{M_i}]$.

(c) $t_i^0 \leq t_c < t_e - T2$: Because the desired SEB has started at this point, $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ are made only for time duration $[t_c, t_i^{M_i}]$. Furthermore, deviation in this scenario means the observed system state or the parameter is different than the predicted one, i.e. $\widetilde{x} \notin \widehat{x}(t_c)$ or $\widetilde{p} \notin \widehat{p}_i$. When the deviation happens, because it is too late to change the control action for $[t_c, t_c + T2]$, the prediction of $[t_c, t_c + T2]$ must be made by Task 3 (the purple line) with the new initial state $\widetilde{x}$ and/or the parameter $\widetilde{p}$. Then the rest of the prediction at $[t_c + T2, t_i^{M_i}]$ can be made by (15) with the $\widehat{x}^*(t_c + T2)$ as the initial state and/or the latest parameter $\widetilde{p}$ (the upper blue line). Mathematically, the prediction can be made by f_{24} with the newly observed $\widetilde{x}$ and $\widetilde{p}$, if they do not deviated from the predicted value. Otherwise, f_{25} can be used to make the prediction, where $\widehat{x}^*(T^*)$ is the prediction made with $\widetilde{x}$ and $\widetilde{p}$ by Task 3.

$$\begin{cases}(R_i, \widetilde{x}, \widetilde{p}, f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \\ (R_i, \widehat{x}^*(T^*), \widehat{x}^*(t_c + T2), \widetilde{p}, f, g_i) \xrightarrow{f_{25}} (\widehat{x}(T_i), \widehat{y}(T_i))^T\end{cases}$$

where $T^* = [t_c, t_c + T2]$ and $T_i = [t_c, t_i^{M_i}]$.

(d) $t_i^{M_i} - T2 \leq t_c < t_i^{M_i}$: Similar to (c), $\widehat{x}(T)$ and $\widehat{y}(T)$ are made for $[t_c, t_i^{M_i}]$ and the deviation also means the observed system state or the parameter is different than the predicted one, i.e. $\widetilde{x} \notin \widehat{x}(t_c)$ or $\widetilde{p} \notin \widehat{p}_i$. Furthermore, because there is less than $T2$ time left for the desired SEB, all the control actions for the rest of the time have been determined at $[2t_c - t_i^{M_i}, t_c]$ (the purple segment on the time axis). Therefore, when the deviation happens, the prediction of $[t_c, t_i^{M_i}]$ must be made by Task 3 (the purple line) with the new initial state $\widetilde{x}$ and/or the parameter $\widetilde{p}$. Mathematically, the prediction can be made by f_{24} with the newly observed $\widetilde{x}$ and $\widetilde{p}$, if they do not deviated from the predicted value. Otherwise, f_{26} can be used to make the prediction, where $\widehat{x}^*(T^*)$ is the prediction made with $\widetilde{x}$ and $\widetilde{p}$ by Task 3. Intuitively, $\widehat{x}(T)$ is the same as $\widehat{x}^*(T^*)$; f in the input is to calculate $\widehat{y}(T)$ from $\widehat{x}^*(T^*)$.

$$\begin{cases}(R_i, \widetilde{x}, \widetilde{p}, f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \\ (\widehat{x}^*(T^*), f) \xrightarrow{f_{26}} (\widehat{x}(T_i), \widehat{y}(T_i))^T\end{cases}$$

where $T^* = [t_c, t_i^{M_i}]$ and $T = [t_c, t_i^{M_i}]$.

In summary, when $T2 < T_i^{M_i} - T_i^0$, the workflow to predict $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ is shown in Fig.44.

When $T2 \geq T_i^{M_i} - T_i^0$, a new set of scenarios (a)∼(d) has to be created in a similar way as $T2 < T_i^{M_i} - T_i^0$. We only present the transformations for each

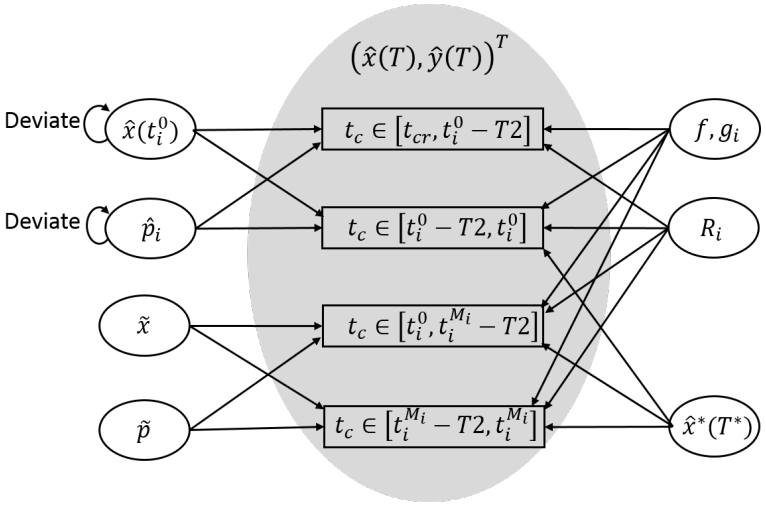


Figure 44: The workflow to predict $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ when $T2 < T_i^{M_i} - T_i^0$.

scenario without detailed explanation (Fig.45). For each scenario, the first transformation is when no deviation happens, and the second is when a deviation is detected.

(a) $t_c < t_i^0 - T2$:

$$\begin{cases} (R_i, \widehat{x}(t_i^0), \widehat{p}_i, f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \\ (R_i, \widehat{x}'(t_i^0), \widehat{p}_i', f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \end{cases}$$

where $T = [t_i^0, t_i^{M_i}]$.

(b) $t_i^0 - T2 \leq t_c < t_i^{M_i} - T2$:

$$\begin{cases} (R_i, \widehat{x}(t_i^0), \widehat{p}_i, f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \\ (R_i, \widehat{x}^*(T^*), \widehat{x}^*(t_c + T2), \widehat{p}_i', f, g_i) \xrightarrow{f_{25}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \end{cases}$$

where $T^* = [t_i^0, t_c + T2]$ and $T_i = [t_i^0, t_i^{M_i}]$.

(c) $t_i^{M_i} - T2 \leq t_c < t_i^0$:

$$\begin{cases} R_i, \widehat{x}(t_i^0), \widehat{p}_i, f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \\ (\widehat{x}^*(T^*), f) \xrightarrow{f_{26}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \end{cases}$$

where $T^* = [t_i^0, t_i^{M_i}]$ and $T_i = [t_i^0, t_i^{M_i}]$.

(d) $t_i^0 < t_c \leq t_i^{M_i}$:

$$\begin{cases} (R_i, \widetilde{x}, \widetilde{p}, f, g_i) \xrightarrow{f_{24}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \\ (\widehat{x}^*(T^*), f) \xrightarrow{f_{26}} (\widehat{x}(T_i), \widehat{y}(T_i))^T \end{cases}$$

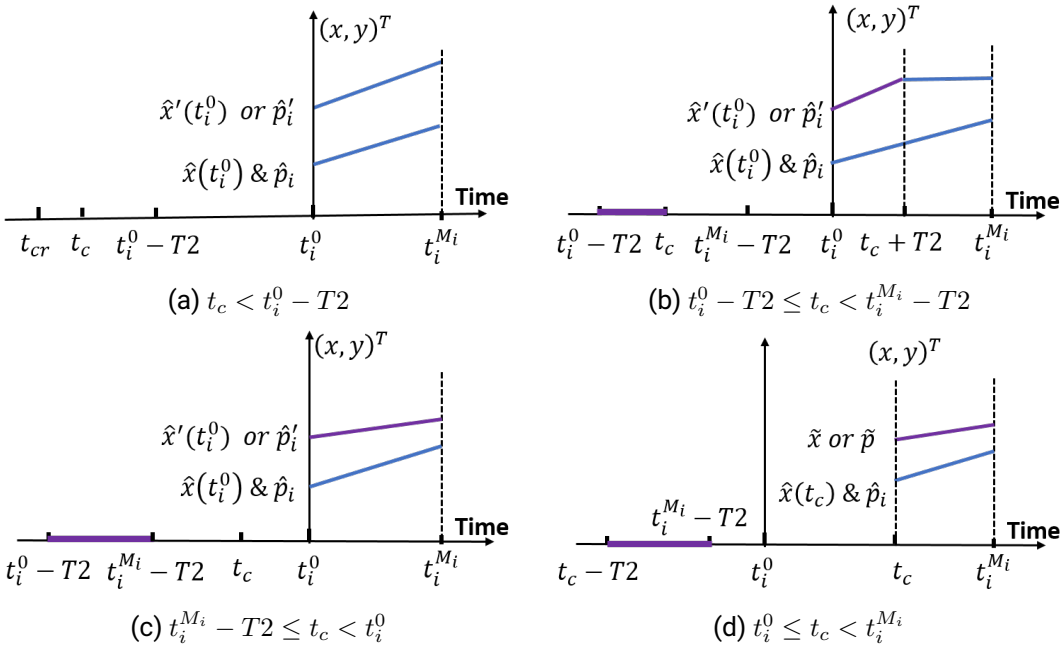


Figure 45: The four scenarios to predict $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ for $T2 \geq t_i^{M_i} - t_i^0$.

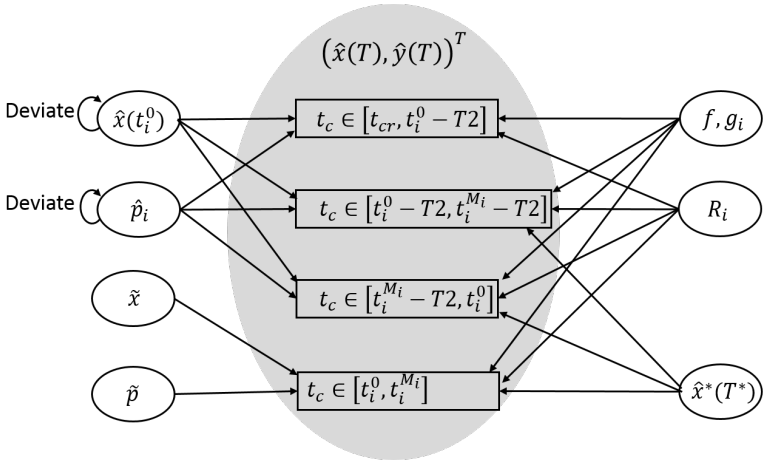


Figure 46: The workflow to predict $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ when $T2 \geq T_i^{M_i} - T_i^0$.

where $T^* = [t_c, t_i^{M_i}]$ and $T_i = [t_c, t_i^{M_i}]$.

In summary, when $T2 \geq T_i^{M_i} - T_i^0$, the workflow to predict $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ is shown in Fig.46.

Finally, the prediction of $\widehat{u}(T_i)$ and $\widehat{\dot{x}}(T_i)$ can be calculated based on $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$. According to (5), $\widehat{u}(T_i)$ can be first calculated from $\widehat{x}(T_i)$ and $\widehat{y}(T_i)$ through g, and then $\widehat{\dot{x}}(T_i)$ can be calculated from $\widehat{x}(T_i)$ and $\widehat{u}(T_i)$ through f. This process is represented in f_{27} below.

$$(\widehat{x}(T_i), \widehat{y}(T_i), R_i, \widehat{p}_i)^T \xrightarrow{f_{27}} (\widehat{u}(T_i), \widehat{\dot{x}}(T_i))^T$$

The monitor theme. This theme is to monitor the *satisfiability* and the *feasibility* of the control references. However, both properties rely on the *validity* of

the process model. Therefore, we first explain how to monitor the validity of the process model, and then the monitoring of satisfiability and the feasibility are addressed.

First, based on Method 2, the process model is comprised of implicit constraints (IC) and explicit constraints $\{V_f, U, X, P, \dot{X}, Y\}$, and the latter is affected by the former, which can be mathematically represented in (19) below.

$$IC \rightarrow \{V_f, U, X, P, \dot{X}, Y\} \tag{19}$$

The validity of the process model means V_f must be true and the explicit constraints must be respected. As a result, the following two transformations are defined to validate the validity of the process model (Fig.49).

$$\begin{cases} IC \xrightarrow{f_{28}} \{V_f, U, X, P, \dot{X}, Y\} \\ (\{X, P, \dot{X}, Y\}, \{\widetilde{x}, \widetilde{p}, \widetilde{\dot{x}}, \widetilde{y}\}) \xrightarrow{f_{29}} Val \end{cases}$$

- f_{28}: This transformation is to generate the process model based on the implicit constraints (IC). V_f denotes the validity of the f. $\{U, X, P, \dot{X}, Y\}$ is the explicit constraints of the process model that the real controlled process must respect. If $\{V_f, U, X, P, \dot{X}, Y\}$ is false, then the process model is invalid and the the controller enters contingency mode immediately.
- f_{29}: This transformation is to make sure the process model is applicable to the real controlled process. Specifically, f_{29} is to make sure $\{x, p, \dot{x}, y\} \in \{X, P, \dot{X}, Y\}$ is true, because if the real controlled process is beyond the state space that the process model depicts, the process model cannot be used to represent and predict the behavior of the real process. If the result turns out to be false, the controller enters contingency mode. Note that u is not included in f_{29} because $u \in U$ will be guaranteed by Task 3.

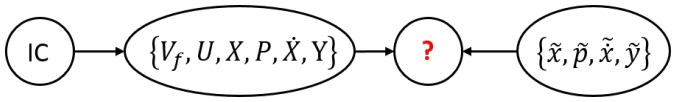


Figure 47: The workflow to monitor the validity of the process model.

Second, the feasibility of the control references means the predicted $\{\widehat{u}(T_i), \widehat{\dot{x}}(T_i), \widehat{x}(T_i), \widehat{y}(T$ will not lead to a violation of the process model. Since we have already established that the process model might also change subject to $\widehat{IC}$, a prediction of the process model also has to be made so that it can be compared with the predicted trajectory of the controlled process.

As a result, the following two transformations are defined to examine the feasibility of the control references (Fig.48).

$$\begin{cases} \widehat{IC} \xrightarrow{f_{30}} \{\widehat{V_f}, \widehat{X}, \widehat{U}, \widehat{P}, \widehat{\dot{X}}, \widehat{Y}\} \\ (\{\widehat{X}(T_i), \widehat{Y}(T_i), \widehat{U}(T_i), \widehat{\dot{X}}(T_i), \widehat{P}_i\}, \{\widehat{x}(T_i), \widehat{y}(T_i), \widehat{u}(T_i), \widehat{\dot{x}}(T_i), \widehat{p}_i\}) \xrightarrow{f_{31}} \{Fea, t_{d1}\} \end{cases}$$

- f_{30}: This transformation is to generate the predicted explicit constraints for the process model, which will be used in f_{31} to examine the feasibility of the control references. If $\{\widehat{V_f}, \widehat{X}, \widehat{U}, \widehat{P}, \widehat{\dot{X}}, \widehat{Y}\}$ is $false$, meaning the process model will become invalid at some point during the time period of interest, then the controller has to enter the contingency mode before $\{\widehat{V_f}, \widehat{X}, \widehat{U}, \widehat{P}, \widehat{\dot{X}}, \widehat{Y}\}$ becomes false.
- f_{31} is defined to examine the feasibility of the control references. If no violation is detected (and hence feasible), Fea is assigned true; otherwise Fea is assigned false and the deviation point t_d (i.e. the earliest time that the violation happens) is returned. Note that if $\widehat{p}_i \not\subseteq \widehat{P}_i$ is detected, the controller has to enter the contingency mode before $\widetilde{p} \notin P$ happens.

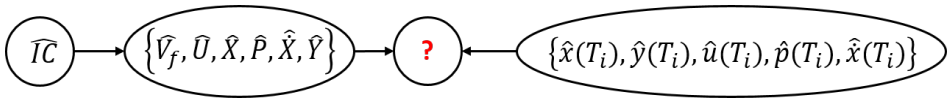


Figure 48: The workflow to validate the feasibility of the control references.

Third, after the validity of the process model and the applicability of the control algorithm is validated, we can simply check the satisfiability (i.e. $\widehat{y}(T_i) \subseteq y(T_i)$) of the control references by applying the transformation f_{32}. If $\widehat{y}(T_i) \subseteq y(T_i)$ is true, then assigned true to Sat; otherwise assign false to Sat and the deviation point t_d (i.e. the time that $\widehat{y}(T_i)$ deviates from $y(T_i)$) is returned.

$$(y(T_i), \widehat{y}(T_i)) \xrightarrow{f_{32}} \{Sat, t_{d2}\}$$

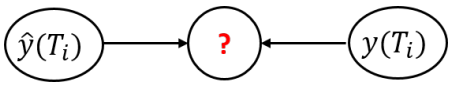


Figure 49: The workflow to validate the satisfiability of the control references.

3.2 The enabling action of Task 2.

Predicting the parameter (EA_6). Both the generate theme and the predict theme requires prediction of the parameters for each segment. We define EA_6 as a general placeholder for the specific prediction activities for specific design applications.

Therefore, EA_6 can be defined as below programmatically:

- Trigger event: NA.
- Guard condition: $stage! = (preStage \vee postStage)$.

- Input: E_p and CP_p represents the set of information necessary to make such prediction from the environment and the controlled process respectively; $y(T)$ contains the information about the time period of the desired SEB.
- Output: $\widehat{p}(T)$ is the predicted parameter value for the time period of the desired SEB.
- Transformation: Execute f_{33}.

Graphically, Action EA_6 can be represented in Fig.50.

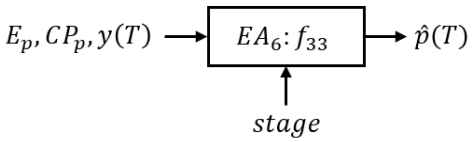


Figure 50: Graphical representation of EA_6.

Segmenting the SEB (EA_7). As explained before, the control references are generated segment by segment. The segment can start/end with time intervals. The time span of each segment is subject to (21). The parameter in each segment must be constant value or set. We define EA_7 as a general placeholder and **assume** that it can always determine the segments in a timely manner.

Therefore, EA_7 can be defined as below programmatically:

- Trigger event: NA.
- Guard condition: $stage! = (preStage \vee postStage)$.
- Input: $y(T)$ contains the information about the time period of the desired SEB; $\widehat{p}(T)$ is used to make sure the parameter value of each segment is the same, which is required by the problem formulation.
- Output: $y(\overline{T})$ is the segmented SEB and $\widehat{p}(\overline{T})$ is the set of the predicted values for the segmented SEB.
- Transformation: Execute f_{34}.

Graphically, Action EA_7 can be represented in Fig.51.

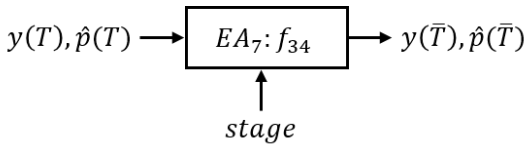


Figure 51: Graphical representation of EA_7.

Determining the scenario(EA_8). EA_8 is an instantaneous reflection of the scenario (for the generate theme) and t_d based on the outputs of the monitor theme. Therefore, EA_8 can be defined as below programmatically:

- Trigger event: NA.
- Guard condition: NA.
- Input: Sat and t_{d1} are the value and the associated deviation point of satisfiability; Fea and t_{d2} are the value and the associated deviation point of the feasibility. They are all outputs of the monitor theme.
- Output: $Scea$ is the specific scenario of the generate theme based on $\{Sat, Fea\}$; t_d is the associated deviation point that will go into the generate theme, specifically f_{23}.
- Transformation: f_{35} is defined as below.

$$\begin{cases} Scea = 1 \text{ and } t_d = false, if\ Sat = true \wedge Fea = true \\ Scea = 2 \text{ and } t_d = t_{d2}, if\ Sat = false \wedge Fea = true \\ Scea = 3 \text{ and } t_d = t_{d1}, if\ Sat = true \wedge Fea = false \\ Scea = 4 \text{ and } t_d = min(t_{d1}, t_{d2}), if\ Sat = false \wedge Fea = false \end{cases}$$

Graphically, Action EA_8 can be represented in Fig.52.

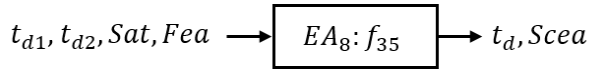


Figure 52: Graphical representation of EA_8.

Setting the priority(EA_9). This action implements the priority of the three themes (Fig.54). Because the predict theme has the highest priority, if its inputs change, EA_9 pauses the monitor theme and the generate theme by setting mon and gen to $false$; after the predicted theme is updated, EA_9 frees the monitor theme by setting mon to $true$; after the monitor theme is updated, EA_9 frees the generate theme by setting gen to be $true$. Similarly, if the inputs of the monitor theme change, EA_9 pauses the generate theme by setting gen to $false$; after the monitor theme is updated, EA_9 frees the generate theme by setting gen to true. If when the monitor theme is updating itself the inputs of the predict theme change, the monitor theme is paused and the loop starts from the predict theme, as the predict theme has the higher priority.

Therefore, EA_9 can be defined as below programmatically:

- Trigger event: NA.
- Guard condition: NA.
- Input: $\{\widehat{x}^*(T^*), \widehat{p}(\overline{T}), \widetilde{x}, \widetilde{p}\}$ and $\{y(\overline{T}, IC)\}$.
- Output: gen, mon.

- Transformation f_{36}: If $\{\widehat{x}^*(T^*), \widehat{p}(\overline{T}), \widetilde{x}, \widetilde{p}\}$ changes, $mon \leftarrow false$ and $gen \leftarrow false$; if the predict theme is updated, $mon \leftarrow true$; if the monitor theme is updated, $gen \leftarrow true$. If $\{y(\overline{T}, IC)\}$ changes, $gen \leftarrow false$; if the monitor theme is updated, $gen \leftarrow true$.

Graphically, Action EA_9 can be represented in Fig.53.

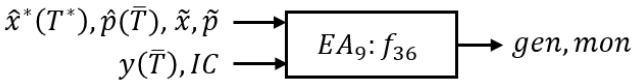


Figure 53: Graphical representation of EA_9.

Finally, **we assume** an underlying function that automatically updates the current time stamps of $\widehat{p}(\overline{T}), \widehat{x}(\overline{T}), \widehat{y}(\overline{T}), \widehat{u}(\overline{T}), \widehat{\dot{x}}(\overline{T})$ and $y(\overline{T})$, and make sure the current time in all of them is consistent with the real current time stamp.

3.3 The main actions of Task 2.

3.3.1 The overall description

The overall information flow of the three themes of Task 2 can be summarized in the left of Fig.54. Task 2 starts with the generate theme receiving $y(\overline{T})$ with Sat and Fea as $true$ by default. The generate theme creates the control references $R(T_1)$ for the first segment, which will then be used to predict the dynamic trajectory of the controlled process $\{\widehat{x}(T_1), \widehat{y}(T_1), \widehat{u}(T_1), \widehat{\dot{x}}(T_1)\}$. Such a prediction is then used to monitor the feasibility and the satisfiability of $R(T_1)$, which will then be used to update the scenario for the generate theme for the second segment. This process is then used repeatedly for all the remaining segments.

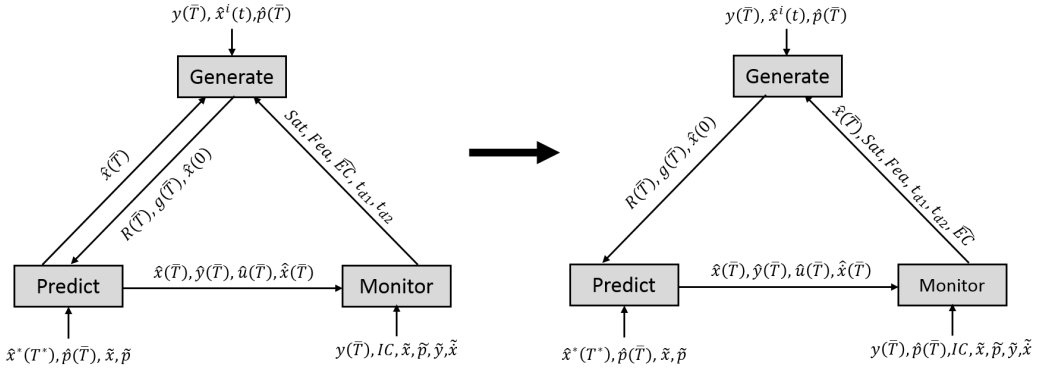


Figure 54: The counter-clockwise overall information flow of Task 2, where $\widehat{EC} = \{\widehat{X}, \widehat{U}, \widehat{P}, \widehat{\dot{X}}, \widehat{Y}\}$ and $\widehat{x}^i(t)$ is the predicted dynamic trajectory of the in-behavior. Left is the original information flow; right is the adjusted information flow to address the synchronization problem.

However, such information flow (left of Fig.54) potentially has a problem of synchronization. When $\widehat{x}(\overline{T})$ changes, the generate theme can receive the latest value immediately. However, such a change in $\widehat{x}(\overline{T})$ may also affect the value of Fea, which is likely to take a longer time to reach the generate theme. The problem is that the generate theme has no idea whether Fea will be affected by the change and hence may have already taken actions to address the change of $\widehat{x}(\overline{T})$ before a new Fea is reached. This inconsistency may be propagated through the information loop and eventually lead to erratic behaviors. As a solution, $\widehat{x}(\overline{T})$ is passed from the predict theme to the monitor theme, and then is passed to the generate theme together with the updated results of the monitor theme (right of Fig.54). In this way, the generate theme is always updated with the same $\widehat{x}(\overline{T})$ used to update the monitor theme.

Furthermore, the loop in Fig.54 only works when each theme updates one at a time. When two themes update at the same time due to a new input value, it may cause problems. For example, when the monitor theme and the generate theme update at the same time due to a new $y(\overline{T})$, the new control references may be generated based on the current $Scea$ or t_d while the new $Scea$ or t_d is still being calculated by the monitor theme. The (incorrect) result of the generate theme will then trigger the predict theme and further propagate through the loop, causing erratic behaviors. It is the same case when the generate theme and the predict theme update at the same time, and the predict theme and the monitor theme update at the same time. Therefore, there must be a way to decide which theme to update the first when two or more themes update at the same time.

As a solution, we assign priorities to the three themes. First, the generate theme always need the results of the predict theme and the monitor theme to generate/update the control references. For example, if the inputs of the predict theme change while the generate theme is updating the control references, the generate theme must stop until the new $scea$ and t_d are decided based on the new prediction. Therefore, the generate theme has the lowest priority among the three themes. Similarly, the predict theme has a higher priority over the monitor theme. Therefore, the priority among the three themes are "the predict theme $>$ the monitor theme $>$ the generate theme". Each theme may start a new loop by the change of the its input. When the inputs of more than one theme change, the loop starts from the theme with the higher priority, and pauses the themes with the lower priority until the the loop progress to the paused theme. In this way, we ensure a single start point of the loop while more than one theme needs to be updated. Refer to EA_9 for the specific implementation.

Therefore, the overall workflow of Task 2 is presented in Fig.55. On "$y(\overline{T})$ transitioning from $false$ to $true$", the generate theme assign $true$ to $g(\overline{T})$ and $R(\overline{T})$ to initiate the whole Task 2. The predict theme passes the observed information to the monitor theme. The monitor theme examine the feasibility based $\widehat{x}(\overline{T})$ and $\widehat{X}$. Once the generate theme receives the output of the monitor theme, it starts to generate $g(\overline{T})$ and $R(\overline{T})$. Furthermore, $gen = true$ and $mon = true$ are the guard conditions for the generate theme and the monitor theme respectively. Finally, all the three themes also have a guard condition "$stage! = (preStage \vee postStage)$", meaning that when the desired SEB is

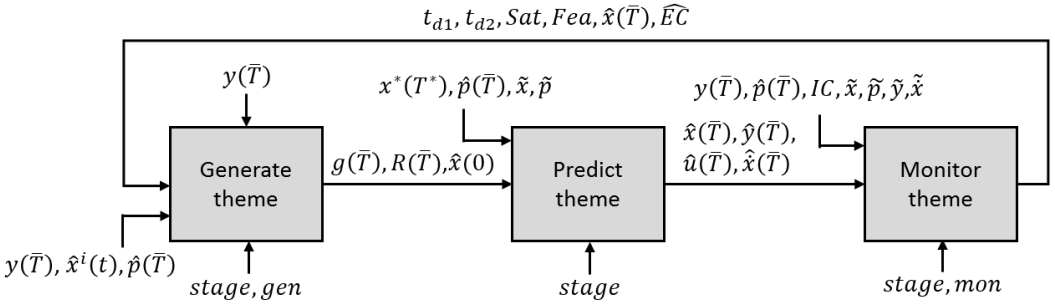


Figure 55: The overall workflow of Task 2. $\widehat{EC} = \{\widehat{X}, \widehat{U}, \widehat{P}, \widehat{\dot{X}}, \widehat{Y}\}$

achieved, Task 2 needs to be terminated.

3.3.2 The generate theme

Three main actions are defined for the generate theme (Fig.56): MA_{21} and MA_{22}, where the first is to select $t_{d'}$ for Scenario $2 \sim 4$, the second is to generate the control references, and the third is to retain the current value of $g(\bar{T})$ and $R(\bar{T})$ while the input values change. We now explain each main action in detail.

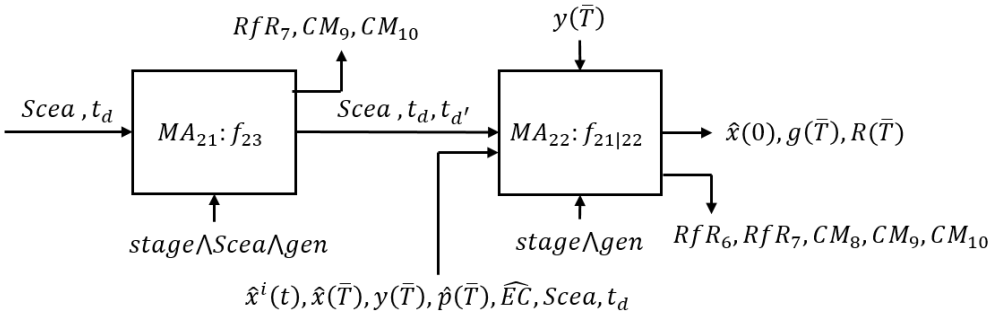


Figure 56: The main actions of the generate theme of Task 2. $\widehat{EC} = \{\widehat{X}, \widehat{U}, \widehat{P}, \widehat{\dot{X}}, \widehat{Y}\}$

- $\boldsymbol{MA_{21}}$:
  - Transformation: f_{23} is to find the time point $t_{d'}$ to start change the dynamic trajectory of the controlled process. The mathematical expression and the associated constraints (14) have been explained previously. However, f_{23} only describes how the $t_{d'}$ is to be found, but not what to do if $t_{d'}$ cannot be found. Hence, according to the specific scenarios, different responses will be issued if $t_{d'}$ cannot be found:
    * Scenario 2: If $t_{d'}$ cannot be found before $t_d - T1$, then RfR_7 is sent for a new SEB to make the control references satisfactory before t_d. The reason that no contingency mode after $t_d - T1$ is because a new SEB may de-conflict the whole situation, as long as the new SEB comes before t_d.

* Scenario 3: If $t_{d'}$ cannot be found before $t_d - T1$, CM_9 is issued at $t_d - T1$ to enter the contingency mode, because a new SEB cannot solve the feasibility problem *directly*.
* Scenario 4: If $t_{d'}$ cannot be found before $t_d - T1$, CM_{10} is issued at $t_d - T1$ to enter the contingency mode for the same reason of Scenario 3.

– Duration: None.
– Output: $t_{d'}, t_d, Scea, RfR_7, CM_9$ and CM_{10}.
– Input: $Scea$ and t_d are the scenario and the deviation point decided by EA_8.
– Trigger: None.
– Guard: $stage \wedge Scea \wedge gen$, where $stage! = (preStage \vee postStage)$, $Scea = \{2, 3, 4\}$ and $gen = true$. Intuitively, this main action cannot happen if the functional goal is issued or the functional goal has achieved, and t_d is not defined if it is currently in Scenario 1.

• $\boldsymbol{MA_{22}}$

– Transformation: When the SEB $y(T)$ transitions from $false$ to $true$, the action first choose the start time along the predicted trajectory $\widehat{x}^i(t)$ of the in-behavior, and the initial condition is $\widehat{x}(0)$.

When $Scea$ and t_d is received from EA_8, this action addresses the synchronization problems by comparing $Scea$ and t_d from MA_{21} and EA_8 to make sure it is synchronized with MA_{21}.

After the synchronization is confirmed, the action precedes to execute $f_{21|22}$. $f_{21|22}$ is to calculate the control algorithm and the control references based on the current scenario. RfR and CM are issued differently in different scenarios:

* Scenario 1: If it is determined before $t_i^{M_i} - T1$ that the control algorithm or the control references cannot be found, then send RfR_6 to Task 1; otherwise enter contingency mode by issuing CM_8. If no control reference is found after $t_i^{M_i} - T2$, then enter contingency mode by issuing CM_8.
* Scenario 2: If it is determined before $t_{d'} - T2$ that the control algorithm or the control references cannot be found, then send RfR_7 to Task 1 for a new SEB to make the control references satisfactory before t_d. If the control algorithm or the control references cannot be found before $t_{d'} - T2$, then RfR_7 is sent to Task 1 immediately at $t_{d'} - T2$.
* Scenario 3: If it is determined before $t_{d'} - T2$ that the control algorithm or the control references cannot be found, then enter the contingency mode by issuing CM_9. If the control algorithm or the control references cannot be found before $t_{d'} - T2$, then the controller enters the contingency mode immediately by issuing CM_9.

* Scenario 4: If it is determined before $t_{d'} - T2$ that the control algorithm or the control references cannot be found, then enter the contingency mode by issuing CM_{10}. If the control algorithm or the control references cannot be found before $t_d - T2$, then the controller enters the contingency mode immediately by issuing CM_{10}.

- Duration: None.
- Output: $g(\overline{T})$ and $R(\overline{T})$ are the generated control algorithm and the control references. $\widehat{x}(0)$ is passed to the predict theme. If $g(\overline{T})$ and $R(\overline{T})$ cannot be found in time, RfR_6 and CM_8 are the response from Scenario 1; RfR_7 is from Scenario 2; CM_9 is from Scenario 3; CM_{10} is from Scenario 4. The reason that no contingency mode for Scenario 2 is same as MA_{21}.
- Input: $\widehat{x}^i(t)$ is from the controller of the in-behavior, and is used to generate the controller references for the first segment. $\widehat{x}(\overline{T})$ is the prediction of the state of the controlled process. $\widehat{p}(\overline{T})$ is the prediction of the parameter; $y(\overline{T})$ is the desired SEB; they come from EA_7. $\{\widehat{EC}, \widehat{x}(\overline{T})\}$ is from the monitor theme and $\{Scea, t_d, t_{d'}\}$ is passed from MA_{21}.
- Trigger: $y(\overline{T}) : false \rightarrow true$, where $y(\overline{T})$ comes from EA_7.
- Guard: $stage \wedge gen$, where $stage! = (preStage \vee postStage)$ and $gen = true$.

In summary, the generate theme of Task 2 can be represented as an action in Fig.57. The sources of the all the external interactions with other themes, the enabling actions and the elements out of the design scope are specified in Fig.58.

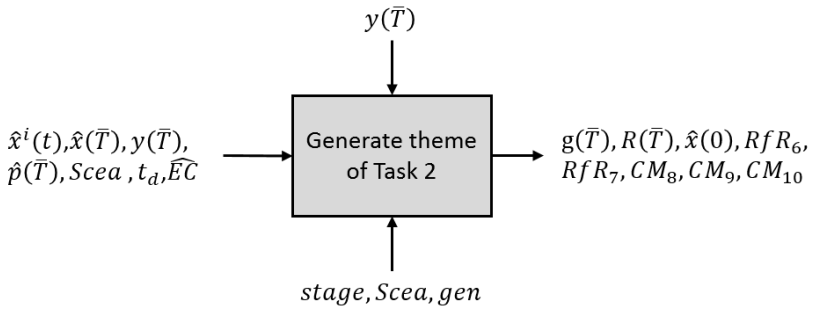


Figure 57: The generate theme of Task 2 represented as an action. The internal interactions are hidden within the grey box.

3.3.3 The predict theme

Two main actions are defined for the predict theme (Fig.59): MA_{23} and MA_{24}, where the former is to calculate the prediction of $\widehat{x}(\overline{T})$ and $\widehat{y}(\overline{T})$, and the latter is to calculate $\widehat{u}(\overline{T})$ and $\widehat{\dot{x}}(\overline{T})$. We now explain each main action in detail.

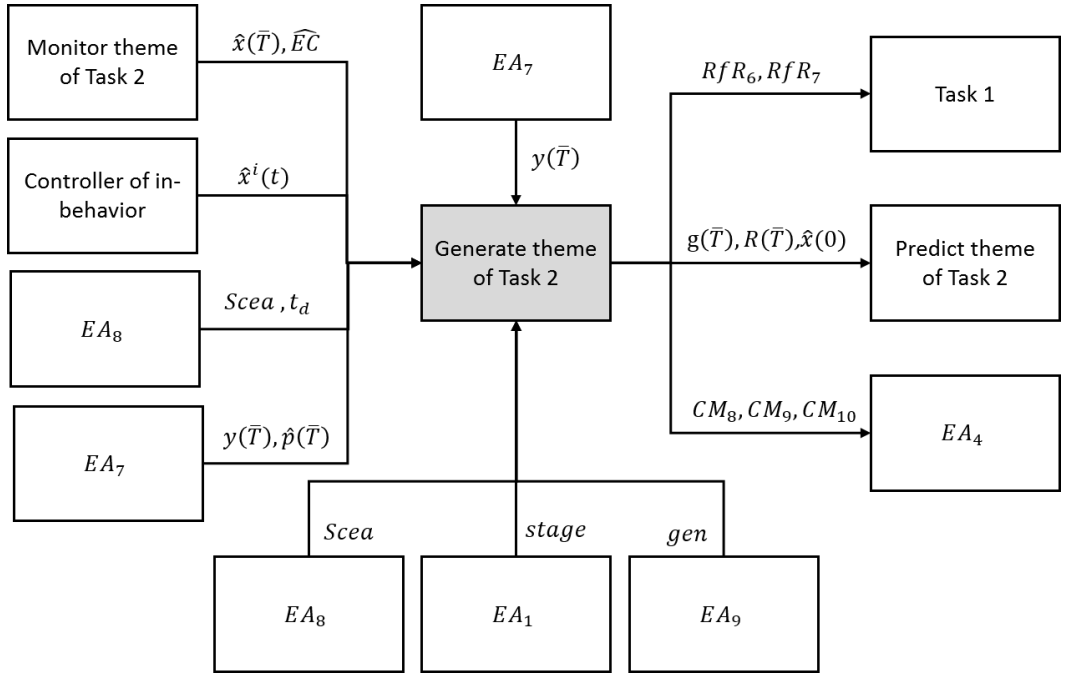


Figure 58: The external interactions of the generate theme of Task 2.

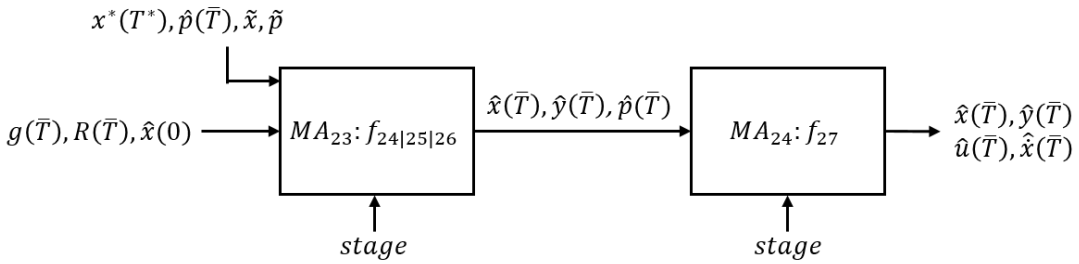


Figure 59: The main actions of the predict theme of Task 2.

- $\boldsymbol{MA_{23}}$
  - Transformation: $f_{24|25|26}$ is a composition of f_{24}, f_{25} and f_{26}, which as explained before, are to predict the dynamic trajectory of x and y, i.e. $\widehat{x}(\overline{T})$ and $\widehat{y}(\overline{T})$.
  - Duration: The duration of this action must be as short as possible. The first e_{23} time must be removed from the output prediction.
  - Output: $\{\widehat{x}(\overline{T}), \widehat{y}(\overline{T}), \widehat{p}(\overline{T})\}$ is for the calculation of $\widehat{u}(\overline{T})$ and $\widehat{\dot{x}}(\overline{T})$; $g(\overline{T})$ and $\widehat{R}(\overline{T})$ are the same as the input. They go into the monitor theme and will be eventually used by the generate theme, so that they are consistent in all three themes.
  - Input: $g(\overline{T}), R(\overline{T})$ and $\widehat{x}(0)$ come from the generate theme; $x^*(T^*)$ comes from Task 3; $\widehat{p}(\overline{T})$ comes from EA_7; $\widetilde{x}$ and $\widetilde{p}$ are the observed current value of the system state and the parameter.
  - Trigger: None.
  - Guard: $stage$, where $stage! = (preStage \vee postStage)$.

- $\boldsymbol{MA_{24}}$
  - Transformation: f_{27} is to calculate $\widehat{u}(\overline{T})$ and $\widehat{\dot{x}}(\overline{T})$ based on the process model (5).
  - Duration: The duration of this action must be as short as possible. The first e_{24} time must be removed from the output prediction.
  - Output: $\{\widehat{u}(\overline{T}), \widehat{\dot{x}}(\overline{T})\}$ is the predicted dynamic trajectory of the controlled process; $\{\widehat{x}(\overline{T}), \widehat{y}(\overline{T})\}$ is passed directly from the input.
  - Input: $\{\widehat{x}(\overline{T}), \widehat{y}(\overline{T}), \widehat{p}(\overline{T})\}$ comes from MA_{23}.
  - Trigger: None.
  - Guard: $stage$, where $stage! = (preStage \vee postStage)$.

In summary, the generate theme of Task 2 can be represented as an action in Fig.57. The sources of the all the external interactions with other themes, the enabling actions and the elements out of the design scope are specified in Fig.58.

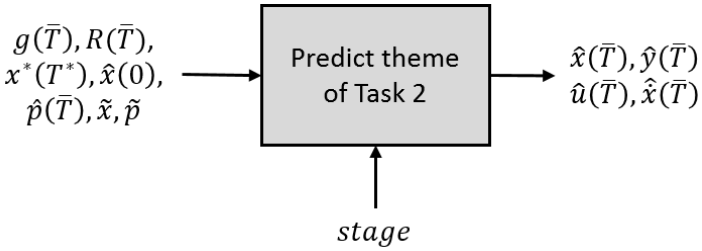


Figure 60: The predict theme of Task 2 represented as an action. The internal interactions are hidden within the grey box.

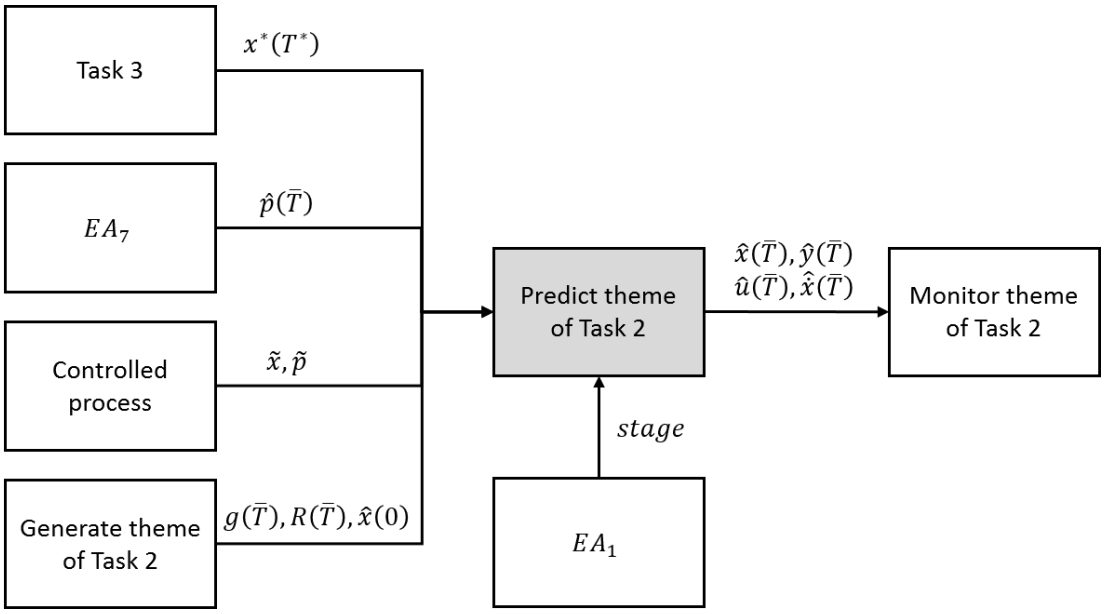


Figure 61: The external interactions of the predict theme of Task 2.

3.3.4 *The monitor theme*

Five main actions are defined for the monitor theme (Fig.62), where MA_{25} and MA_{26} are to make sure that the process model is applicable in real time; MA_{27} and MA_{28} are to decide the feasibility of the control references; MA_{29} is the examine the satisfiability of the control references. We now explain each main action in detail.

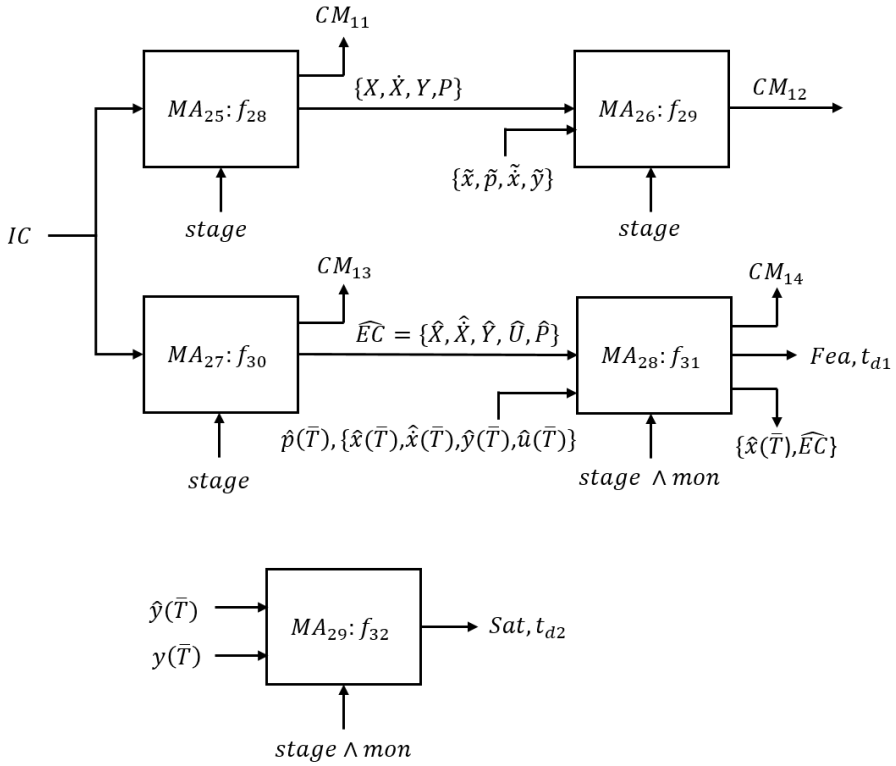


Figure 62: The main actions of the monitor theme of Task 2.

- $\boldsymbol{MA_{25}}$
  - Transformation: f_{28} is to calculate and update the process model in the real time[8]. The output of f_{28} is $\{V_f, X, \dot{X}, U, Y, P\}$. If V_f if $false$, then CM_{11} is issued; otherwise the explicit constraints $\{X, \dot{X}, Y, P\}$ are passed to MA_{26}.
  - Duration: The duration of this action must be as short as possible.
  - Output: CM_{11} and $\{X, \dot{X}, Y, P\}$. Note that the reason that U is not included in the output is because $u \in U$ will be guaranteed by the control algorithm in Task 3.
  - Input: The implicit constraints IC from the environment (denoted as IC_1) or the controlled process (denoted as IC_2). What is included in IC is determined at the design time.
  - Trigger: NA.

[8]Refer Method 2 to see how the process is derived.

- Guard: $stage$ where $stage = (Stage2 \vee Stage3)$. As explained above, the process model is only to be applied in $Stage2$ and $Stage3$.

- $\boldsymbol{MA_{26}}$
  - Transformation: f_{29} is to compare the $\{\widetilde{x}, \widetilde{\dot{x}}, \widetilde{y}, \widetilde{p}\}$ with the process model $\{X, \dot{X}, Y, P\}$. If one element of the former set is beyond the latter set, the process model is not applicable at the current time, and hence the controller must enter the contingency mode by issuing CM_{12}.
  - Duration: The duration of this action must be as short as possible.
  - Output: CM_{12}. If the process model is applicable, then no output comes out of this action.
  - Input: $\{\widetilde{x}, \widetilde{p}, \widetilde{\dot{x}}, \widetilde{y}\}$ observed from the real controlled process, and $\{X, \dot{X}, Y, P\}$ from MA_{25}.
  - Trigger: None.
  - Guard: $stage = (Stage2 \vee Stage3)$.

- $\boldsymbol{MA_{27}}$
  - Transformation: f_{30} is to predict $\widehat{V_f}$ and the explicit constraints $\widehat{EC} = \{\widehat{X}, \widehat{\dot{X}}, \widehat{Y}, \widehat{U}, \widehat{P}\}$. If either of them is $false$, then the controller must enter the contingency mode before $\widehat{V_f}$ or $\widehat{EC}$ becomes false by issuing CM_{13}.
  - Duration: The duration of this action must be as short as possible. The first e_{27} time must be removed from the output prediction.
  - Output: $\{\widehat{X}, \widehat{\dot{X}}, \widehat{Y}, \widehat{U}, \widehat{P}\}$.
  - Input: The implicit constraints IC from the environment (denoted as $\overline{IC_3}$) or the controlled process (denoted as IC_4). The implicit constraints IC that are determined at the design time.
  - Trigger: NA.
  - Guard: $stage = (Stage2 \vee Stage3)$.

- $\boldsymbol{MA_{28}}$
  - Transformation: f_{31} is to compare the predicted dynamic trajectory of the controlled process $\{\widehat{x}(\overline{T}), \widehat{\dot{x}}(\overline{T}), \widehat{y}(\overline{T}), \widehat{u}(\overline{T}), \widehat{p}(\overline{T})\}$ with predicted process model $\{\widehat{X}, \widehat{\dot{X}}, \widehat{Y}, \widehat{U}, \widehat{P}\}$. If $\widehat{p}(\overline{T}) \not\subseteq \widehat{P}$, the controller must enter the contingency mode before $\widetilde{p} \notin P$ by issuing CM_{14}, as the controller has no impact over the parameter. For the rest of the set, if a violation is detected, then the control references are infeasible (represented by Fea) and the deviation point (t_{d1}) is decided.
  - Duration: The duration of this action must be as short as possible.
  - Output: CM_{14}, Fea and t_{d1} are explained above. $\widehat{EC}$ and $\widehat{x}(\overline{T})$ are faithfully passed from the input for the reason of synchronization.

- Input: $\widehat{EC} = \{\widehat{X}, \widehat{\dot{X}}, \widehat{Y}, \widehat{U}, \widehat{P}\}$ from MA_{27}; $\{\widehat{x}(\overline{T}), \widehat{\dot{x}}(\overline{T}), \widehat{y}(\overline{T}), \widehat{u}(\overline{T})\}$ from the predict theme; $\widehat{p}(\overline{T})$ from EA_7.
- Trigger: None.
- Guard: $stage \wedge mon$ where $stage = (Stage2 \vee Stage3)$ and $mon = true$.

- $\boldsymbol{MA_{29}}$
  - Transformation: f_{32} is to examine the satisfiability of the control references by comparing $\widehat{y}(\overline{T})$ and $y(\overline{T})$.
  - Duration: The duration of this action must be as short as possible.
  - Output: Sat represents the satisfiability, and t_{d2} is the deviation point if $Sat = false$; $y(\overline{T})$ is faithfully passed from the input for the reason of synchronization. In additional, the output of this action must be synchronized with the outputs of MA_{28}.
  - Input: $y(\overline{T})$ from EA_7, and $\widehat{y}(\overline{T})$ from the predict theme.
  - Trigger: None.
  - Guard: $stage \wedge mon$ where $stage = (Stage2 \vee Stage3)$ and $mon = true$.

In summary, the monitor theme of Task 2 can be represented as an action in Fig.63, where $IC = \{IC_1, IC_2, IC_3, IC_4\}$. The sources of the all the external interactions with other themes, the enabling actions and the elements out of the design scope are specified in Fig.64.

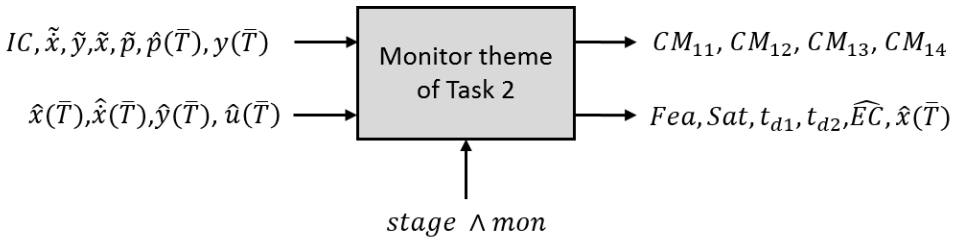


Figure 63: The monitor theme of Task 2 represented as an action. The internal interactions are hidden within the grey box.

4 Task 3: Generate control action

4.1 Fundamentals

4.1.1 Problem formulation

Given the control references r, the next task of a controller is to generate the control action to achieve the issued control references. At the **value aspect**, a

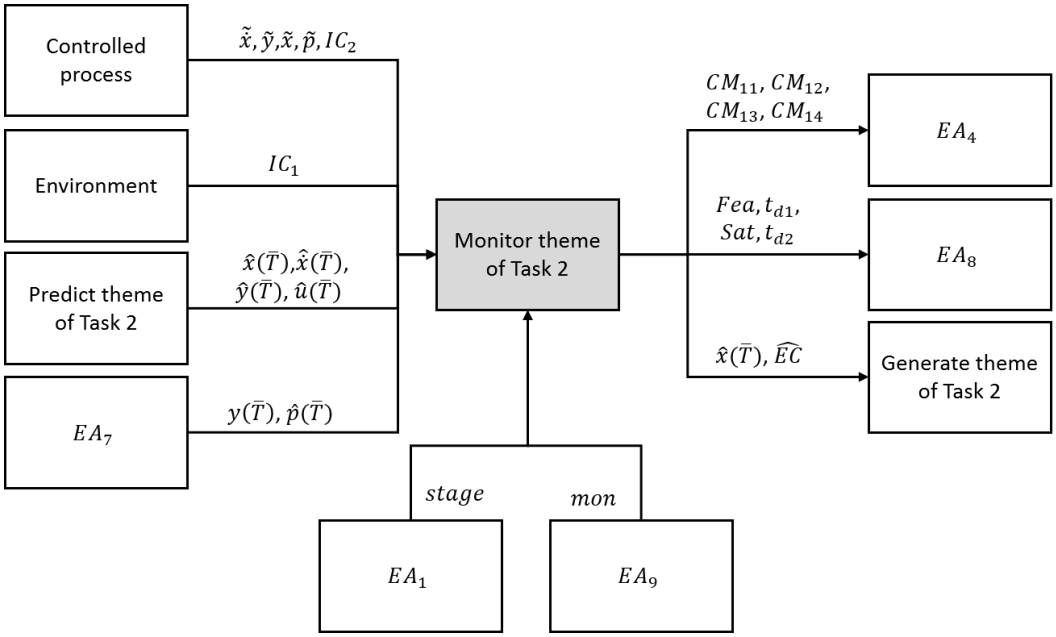


Figure 64: The external interactions of the monitor theme of Task 2.

control algorithm is to generate the control action following (20) below.

$$\begin{aligned} &(r, x(t), p) \xrightarrow{g} u(t) \\ &s.t.\ \{x(t), u(t)\}^T \in \{X, U\}^T \end{aligned} \tag{20}$$

where X and U are the explicit constraints defined in Task 2.

At the **timing aspect**, two general time delays must be considered when defining g (Fig.65). One is $T2$ which represents the duration of the control structure from the system state is observed to the generated control action fully takes effect on the controlled process. But because of $T2$, the system state when the control action takes effect might be different from the state observed to generate the control action in the first place. Therefore, a general approach is to use a predicted state ($T2$ after the current time stamp) to calculate the control action so that the control action takes effect on the right system state. Therefore, if the control action is supposed to take effect at time t, then the generation of the control action must be started no later than $t - T2$.

The other time delay is $T3$, which represents the duration of the controlled process from the time when the control action is applied to the time when the control action fully takes effect. Therefore, if the control action is supposed to take effect at time t, then the control action must be applied before $t - T3$.

Furthermore, there is possibly another type of time delay (denoted as Δ_{T2}), which is the ahead-of time to generate the control action. Specifically, if the control action is supposed to take effect at time t, then the generation of the control action is started no later than $t - T2 - \Delta_{T2}$. Practically, Δ_{T2} creates the time window to send out RfR and awaits resolution. For example, if the air traffic controller waits until the last minute to generate the command for the airplanes that he/she is responsible of, it will have no room to adjust the target

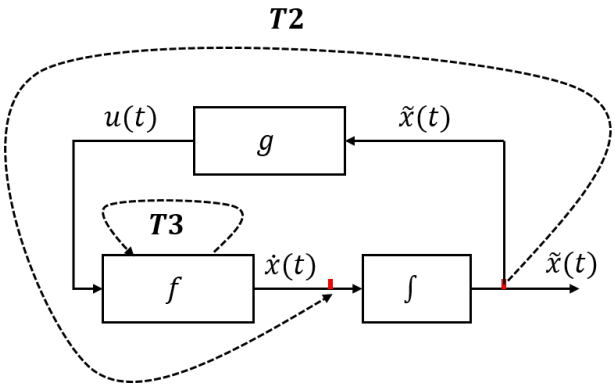


Figure 65: The time delays relevant to Task 3.

waypoint should any unexpected event happens to the airplane. More usually, the command is issued a certain time in advance for the possible "negotiation" between the air traffic controller and the pilot for specific reasons. If a system (especially those operating at a faster pace) does not have such ahead-of time, Δ_{T2} can be simply assigned to 0 without losing the generality of such construct.

In addition, ideally the control action can be applied to the controlled process as a continuous flow. However, this is not realistic. In real practice, the control action is usually applied to controlled process in a discrete manner. For example in Fig.66, the continuous evolution of $x(t)$ is decomposed into four segments, and each segment is applied to a constant control action $u_1 \sim u_4$ respectively. The duration of each segment depends on the specific controlled process and the desired performance, and hence can only be determined in a case-by-case manner. It is worth mentioning that the segment here is different from the segment used in Task 2, where the former is to generate the control references and the latter is to generate the control actions to achieve the individual control references. Because of this difference, the length of the segments in Task 3 is in general shorter than the segments in Task 2.

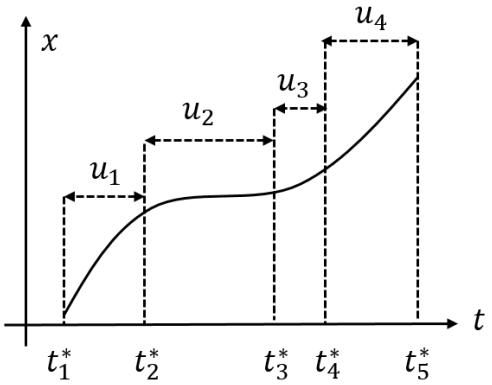


Figure 66: In real practice, an ideal flow of control action is discretized into segments, each of which is applied to the same magnitude of the control action.

Therefore, Task 3 can be described in Fig.67. For the ith segment, when the time stamp comes to $t_c = t_i^* - T2 - \Delta_{T2}$, the controller starts to generate the

control action for t_i^*. First, the system state $\widetilde{x}$ is observed at current time t_c, based on which the system state at $t_i^* = t_c + T2 + \Delta_{T2}$ (denoted as $\widehat{x}^*(t_i^*)$) is predicted. Then the control action u_i is generated and applied to the controlled process before $t_i^* - T3$ so that there will be enough time for the applied control action to take full effect on the controlled process at t_i^*. Moreover, as long as t_i^* has not been reached, no matter whether u_i is generated, if any of the input $\{r, \widehat{x}^*(t^*), \widehat{p}(t^*)\}$ change, the change must be reflected into u_i in a timely manner. However, if the change happens after $t_i^* - T2$, there will not be enough time to calculate a new u_i, and appropriate measures must taken accordingly. Furthermore, if u_i does not need to change, the controller starts to generate the control action for $[t_{i+1}^*, t_{i+2}^*]$ when the timeline comes to $t_{i+1}^* - T2 - \Delta_{T2}$. As shown in the figure, the latest possible time for u_i to be generated is $t_i^* - T3$, and there must at least be $T2$ time interval till t_{i+1}^* so that there will be enough time for the controller to generate u_{i+1}. Therefore, the interval between t_i^* and t_{i+1}^* is subject the constraint in (21) below.

$$t_{i+1}^* - t_i^* \geq T2 - T3 \tag{21}$$

where $T2 - T3$ in fact represents the expected time for the controller to generate the control action. Therefore, this constraint intuitively means that the controller must calculate the control action faster than the control action is required to be applied to the controlled process. Note that (21) in fact has a more general implication, which will be explained momentarily.

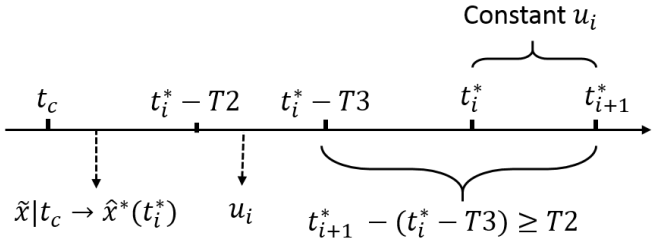


Figure 67: The problem formulation of Task 3, where $t_c = t_i^* - T2 - \Delta_{T2}$. Note that u_i is not necessarily generated after $t_c + \Delta_{T2}$, although it is the case in the figure.

Finally, (22) is a general formulation to generate the control action for the ith segment. In (22), ${}^ir$ is the control reference that the u_i is to achieve; ${}^ig$ is the control algorithm that is decided for the ${}^ir$ in Task 2; $\widehat{x}^*(t_i^*)$ is the predicted system state at t_i^*. Note that $\widehat{x}^*(t_i^*)$ does not come from the predict theme of Task 2, but the input to the predict theme of Task 2. $\widehat{p}(t_i^*)$ is the predicted value of p at time t_i^*, which comes from EA_6 of Task 2. u_i is the control action that is supposed to take effect (rather than "applied to") on the controlled process during T_i^*. $\widehat{x}^*(T_i^*)$ is the predicted system state during T_i^* that can be calculated once u_i is determined. Both u_i and $\widehat{x}^*(T_i^*)$ have to satisfy the predicted explicit constraints of the process model $\{\widehat{X}(T_i^*), \widehat{U}(T_i^*)\}^T$ that comes from MA_{27} of Task 2. Actually, the $\{X, U\}^T$ used in (20) is less general than the $\{\widehat{X}(T_i^*), \widehat{U}(T_i^*)\}^T$ here, because

the former is only correct if the process model does not change over time.

$$\begin{aligned} ({}^{i}r, \widehat{x}^*(t_i^*), \widehat{p}(t_i^*)) &\xrightarrow{{}^{i}g} \{u_i, \widehat{x}^*(T_i^*)\} \\ s.t.\ \{\widehat{x}^*(T_i^*), u_i\}^T &\in \{\widehat{X}(T_i^*), \widehat{U}(T_i^*)\}^T \end{aligned} \tag{22}$$

where $T_i^* = [t_i^*, t_{i+1}^*]$.

4.1.2 The workflow of Task 3

Three themes are defined for Task 3. The generate theme is to decide the control action for each individual time interval; the predict theme is to predict the initial states for the time interval under study; the monitor theme is to decide the time interval to update in the case of change.

The generate theme. The generate theme is to generate the control actions. But before the control actions can be generated, the temporal segments for each control action must be generated first, where the ith segment is the time between $[t_i^*, t_{i+1}^*]$. Note that each segment is the time interval that a constant control action take effects on the controlled process, rather than the time that the control action is applied due the time delay $T3$. Furthermore, the determination of the segments depends on the specific controlled process, the desired performance and the control references. Hence, it can only be decided on a case-by-case basis. For this reason, we take the segments (defined as below) as given in this work as long as they satisfy the constraint in (21).

$$t^* = \{t_1^*, t_2^*, ..., t_k^*\} \text{ and } T_i^* = [t_i^*, t_{i+1}^*].$$

For T_i^*, the timeline of the generate theme is shown in Fig.68. When the timeline comes to $t_c = t_i^* - T2 - \Delta_{T2}$, the controller is triggered to follow (22) to generate the control action that is supposed to take effect at t_i^*. If $u(t_i^*)$ cannot be generated before $t_i^* - T3$, then there will be no time for the control action, even if successfully generated later, to take effect at t_i^*. Therefore, once the timeline passes $t_i^* - T3$ without a control action, the controller applies a default action u_d [9] to the controlled process immediately. After that, when the timeline comes to $t_{i+1}^* - T2 - \Delta_{T2}$, the controller follows (22) to generate the control action $u(t_{i+1}^*)$ for t_{i+1}^*.

Therefore, one transformation f_{38} is defined for the generate theme, which is basically a summary of (22).

$$({}^{i}r, {}^{i}g, \widehat{x}^*(t_i^*), \widehat{p}(t_i^*), \widehat{X}(T_i^*), \widehat{U}(T_i^*), T_i^*) \xrightarrow{f_{38}} u_i/u_d$$

where ${}^{i}g$ is the control algorithm selected by Task 2 that is used to generate u_i, u_d is the default control action when u_i cannot be found before $t_i^* - T2$, and $\widehat{x}^*(t_1^*) = \widehat{x}(0)$ where $\widehat{x}(0)$ comes from the controller of the in-behavior.

[9] No control action is also a default action.

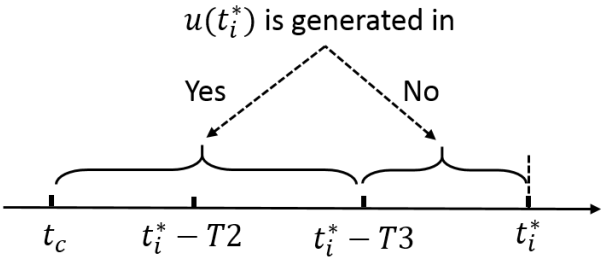


Figure 68: The timeline of the generate theme for the ith impact time point, where t_c in the figure is the start time to generate the control action for t_i^*, thus $t_c = t_i^* - T2 - \Delta_{T2}$. Note that t_i^* can be a time interval as well.

The predict theme. Among all the inputs of f_{38}, ${}^ir, \widehat{p}(t_i^*), \widehat{X}(T_i^*), \widehat{U}(T_i^*)$ and ${}^ig$ are all passed from Task 2; T_i^* is taken as given. Only $\widehat{x}^*(t_i^*)$ must be predicted by this task. Although $\widehat{x}^*(T_{i-1}^*)$ contains information about $\widehat{x}^*(t_i^*)$, such information is only valid when u_{i-1} is being generated. If the value of $\widehat{x}^*(t_i^*)$ changes afterward, it needs to be updated accordingly. Therefore, the predict theme of Task 3 is to predict $\widehat{x}^*(t_i^*)$ after u_{i-1} is generated.

Intuitively, the future state of the controlled process is determined by the control actions to be applied, the initial state and the parameter values from the current time to the specific future time under study. Therefore, $\widehat{x}^*(t_i^*)$ in general can calculated in (23) below.

$$(u[t_i^*], \widetilde{x}, \widehat{p}[t_i^*]) \xrightarrow{f} (\widehat{x}^*(t_i^*), \widehat{x}^*[t_i^*])^T \tag{23}$$

where $u[t_i^*] = \{u(t)|t \in [t_c, t_i^*]\}$, $\widehat{p}[t_i^*] = \{\widehat{p}(t)|t \in [t_c, t_i^*]\}$ and $\widehat{x}^*[t_i^*] = \{\widehat{x}^*(t)|t \in [t_c, t_i^*]\}$. Note that the definition of $u[t_i^*]$ assumes that the control actions till the $(i-1)$th segment are all defined.

However, depending on the temporal relationship between t_c and st [10], (23) takes different forms.

Scenario 1: $t_c < st$. As shown in Fig.69, when the current time is before st, the time duration under study is $[st, t_i^*]$ (rather than $[t_c, t_i^*]$). All the inputs of (23) are redefined as below.

- $u[t_i^*] = \{u(t)|t \in [st, t_i^*]\}$.
- $\widehat{p}[t_i^*] = \{\widehat{p}(t)|t \in [st, t_i^*]\}$.
- $\widetilde{x} = \widehat{x}(st)$. Because SEB has not started at the current time t_c, the prediction can only starts from the initial state of the desired SEB, which can only be predicted by the controller of the the in-behavior.

Therefore, the transformation for this scenario can be summarized by f_{39} below.

$$(u[t_i^*], \widehat{x}(st), \widehat{p}[t_i^*], f) \xrightarrow{f_{39}} (\widehat{x}^*(t_i^*), \widehat{x}^*[t_i^*])^T$$

[10] st is the start time of the entire SEB. It is defined in Task 1.

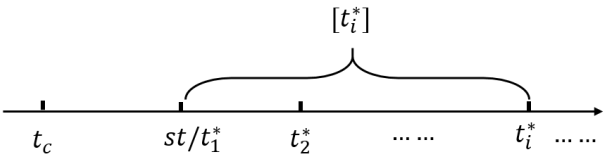


Figure 69: The predict theme when $t_c < st$.

where f is the transformation of the process model and $\widehat{x}^*[t_i^*] = \{\widehat{x}^*(t)|t \in [st, t_i^*]\}$.

Scenario 2: $t_c \geq st$. As shown in Fig.70, when the current time is after st, the time duration under study is $[t_c, t_i^*]$. The definition of (23) can be applied directly.

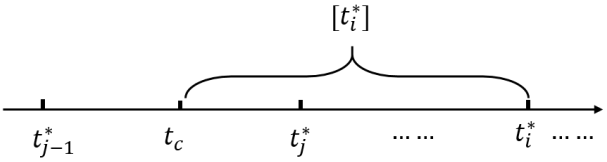


Figure 70: The predict theme when $t_c \geq st$.

Therefore, the transformation for this scenario can be summarized by f_{40} below.

$$(u[t_i^*], \widetilde{x}, \widehat{p}[t_i^*], f) \xrightarrow{f_{40}} (\widehat{x}^*(t_i^*), \widehat{x}^*[t_i^*])^T$$

where f is the transformation of the process model and $\widehat{x}^*[t_i^*] = \{\widehat{x}^*(t)|t \in [t_c, t_i^*]\}$.

Finally, among the two outputs of the predict theme, the former $\widehat{x}^*(t_i^*)$ is used in the generate theme to decide the control action and the monitor theme below to update the generated control action, and the latter $\widehat{x}^*[t_i^*]$ is used in the predict theme of Task 2.

The monitor theme. Because the control action is subject to change, the monitor theme is to update the control actions in time.

First, for a single control action u_i, if any of the inputs of f_{38} is changed, u_i needs to be updated. However, depending on whether u_i is already generated or being generated, it is updated differently.

- u_i is already generated: the inputs of f_{38} can be divided into two groups: $\{^ir, \widehat{x}^*(t_i^*), \widehat{p}(t_i^*), T_i^*, ^ig\}$ and $\{\widehat{X}(T_i^*), \widehat{U}(T_i^*)\}$. If any element of the first group changes, then recalculate u_i by following f_{38}. If any element of the second group changes, then no action is needed, because the unsatisfiability of $\{\widehat{X}(T_i^*), \widehat{U}(T_i^*)\}$ has been addressed by the "feasibility" of Task 2.
- u_i is being generated: if u_i is being calculated but has not accomplished, then restart the action by using the most up-to-date input values.

Second, the next question is how many u_i needs to be updated. Only those control actions that are generated or being generated but have not been applied to the controlled process can be updated. As shown in Fig.71, because of the time delay $T3$ for an applied control action to take full effect at the controlled process, it is impossible for a generated control action not being applied during $[t_c, t_c + T3]$. Moreover, the controller only starts generating a new control action when the current time t_c is $\Delta_{T2} + T2$ closer to the start of a new segment. Therefore, the relevant segments to be monitored are those within $[t_c + T3, t_c + \Delta_{T2} + T2]$, i.e. the segments within the bracket in Fig.71. Furthermore, within all the relevant segments, if one segment that starts before $t_c + T2$ (such as T_j^* in Fig.71) needs to be updated, there will not be enough time for the controller to generate a new control action for T_j^*. As a result, the default control action u_d is applied. For the segments that are after $t_c + T2$ (such as T_i^* in Fig.71), new control actions can be calculated by following f_{38}.

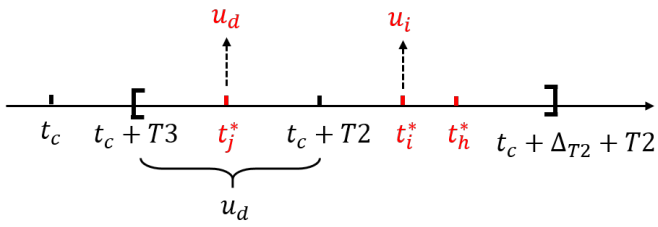


Figure 71: The timeline of the monitor theme of Task 3. The segments within the bracket are those being monitored, where the first segment is denoted as T_j^* and the last is denoted as T_h^*.

Finally, not all the segments within $[t_c + T3, t_c + \Delta_{T2} + T2]$ is to be updated. For example, if the first control action that needs to be updated is u_i, then only the control actions that are after u_i have to be updated, because the change of u_i will change $\widehat{x}^*(t_{i+1}^*)$, which will in turn change u_{i+1}.

Therefore, as shown in Fig.71, the segments to be monitored is defined as $T_{mon}^* = \{T_j^* ... T_i^* ... T_h^*\}$. The monitor theme is to find the first segment (denoted as T_{upt}^*) that needs to be updated by constantly monitoring $\{{}^ir, \widehat{x}^*(t_i^*), \widehat{p}(t_i^*), \widehat{X}(T_i^*), \widehat{U}(T_i^*), T_i^*, {}^ig$ for each segment in T_{mon}^*. Once T_{upt}^* is found, all the segments after T_{upt}^* also need to be updated accordingly. In summary, the monitor theme can be presented as the transformation f_{41} below.

$$(T_{mon}^*, \{I_i | i = (j...h)\}) \xrightarrow{f_{41}} T_{upt}^*$$

where $I_i = \{{}^ir, \widehat{x}^*(t_i^*), \widehat{p}(t_i^*), \widehat{X}(T_i^*), \widehat{U}(T_i^*), T_i^*, {}^ig\}$ and $\widehat{x}^*(t_1^*) = \widehat{x}(ts)$.

Once T_{upt}^* is found, it is passed to the generate theme to update the control actions.

4.2 The enabling action of Task 3

Three enabling actions are identified for Task 3.

Segmenting the control action (EA_{10}). As explained before, the control action is usually applied segment by segment for a specific control reference with a constant magnitude for each segment. Because the segments can only be determined on a case-by-case basis, we only define EA_{10} as a placeholder without specifying the details to signify a certain action to generate the segments. EA_{10} can be defined as below programmatically:

- Trigger event: NA.
- Guard condition: NA.
- Input: $R(\overline{T})$ from Task 2. **We assume** an underlying function that automatically updates the current time stamps of $R(\overline{T})$ so that $R(\overline{T})$ only contains the control references that have been generated but have not been achieved.
- Output: T^* is the set of segments for the control action to achieve the generated the control references.
- Transformation: Execute f_{42}.

Graphically, Action EA_{10} can be represented in Fig.72.

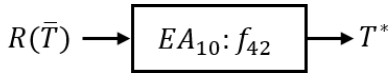


Figure 72: Graphical representation of EA_{10}.

Determining the control actions to monitored (EA_{11}). All the control actions within $[t_c + T3, t_c + \Delta_{T2} + T2]$ that have not been applied must be monitored. EA_{11} is to determine the segments of those control actions.

Therefore, EA_{11} can be defined as below programmatically:

- Trigger event: NA.
- Guard condition: NA.
- Input: T^* from EA_{10}.
- Output: T^*_{mon} is the set of segments of the control actions to be monitored.
- Transformation: f_{43} selects the segments within $[t_c + T3, t_c + \Delta_{T2} + T2]$ where the control action has been generated but has not been applied.

Graphically, Action EA_{11} can be represented in Fig.73.

Setting the priority(EA_{12}). This action implements the priority of the three themes (Fig.75). It is similar to EA_9, and can be defined as below programmatically:

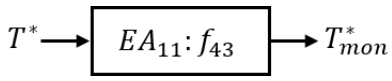


Figure 73: Graphical representation of EA_{11}.

- Trigger event: NA.
- Guard condition: NA.
- Input: $\{\widehat{p}(\overline{T}), \widetilde{x}, \widehat{x}(0)\}$ and $\{R(\overline{T}), \widehat{X}(T), \widehat{U}(T), g(\overline{T})\}$.
- Output: $gen1, mon1$.
- Transformation f_{37}: If $\{\widehat{p}(\overline{T}), \widetilde{x}, \widehat{x}(0)\}$ changes, $mon1 \leftarrow false$ and $gen1 \leftarrow false$; if the predict theme is updated, $mon1 \leftarrow true$; if the monitor theme is updated, $gen1 \leftarrow true$. If $\{R(\overline{T}), \widehat{X}(T), \widehat{U}(T), g(\overline{T})\}$ changes, $gen1 \leftarrow false$; if the monitor theme is updated, $gen1 \leftarrow true$.

Graphically, Action EA_{12} can be represented in Fig.74.

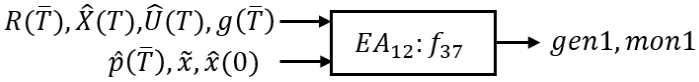


Figure 74: Graphical representation of EA_{12}.

4.3 The main actions of Task 3

4.3.1 The overall description

The overall information flow of Task 3 is shown in Fig.75. The generate theme creates/ updates the control action $u(T^*)$ following f_{38}. Then the generated $u(T^*)$ is input into the predict theme to predict and update the system states $\widehat{x}^*(T^*)$ by following f_{39} or f_{40} depending on the specific scenario. The system states are then input into the generate theme and the monitor theme because the generate theme needs $\widehat{x}^*(t_i^*)$ to generate $u(T_i^*)$; the monitor theme monitors the change of $\widehat{x}^*(T^*)$ to decide from which segment the control actions should be updated (i.e. T^*_{upt}).

The sources of the inputs from the external of the three themes are as following: $\{R(\overline{T}), \widehat{p}(\overline{T}), \widehat{X}, \widehat{U}, g(\overline{T})\}$ is passed from Task 2; T^* is from EA_{10}; $\widetilde{x}$ is observed from the real controlled process; $\widehat{x}(0)$ is from the controller of the in-behavior; T^*_{mon} is from EA_{11}.

The overall workflow of Task 3 is shown in Fig.75 (left). The initial condition of T^*_{upt} is T_1^*, and the initial condition of $\widehat{x}^*(t^*)$ is $\widehat{x}(0)$. Once the inputs of the generate theme are ready, the generate theme starts to generate the control action $u(T^*)$. The predict theme is then to compute $\widehat{x}^*(T^*)$ using the generated

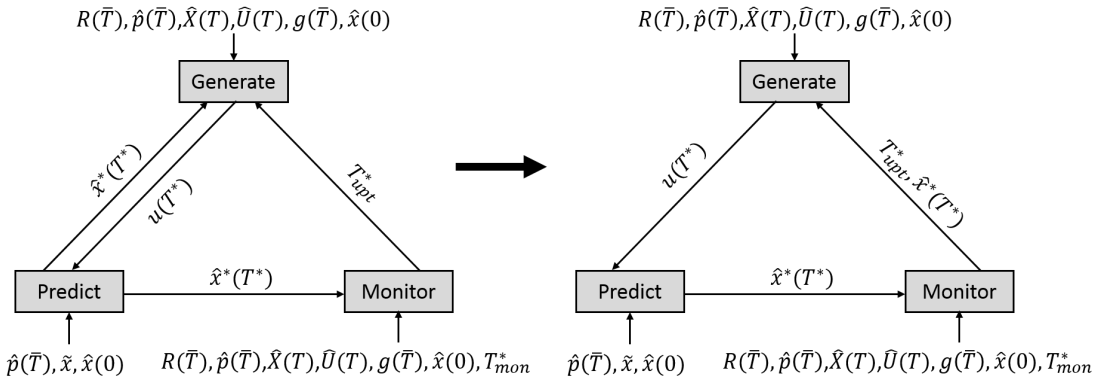


Figure 75: The overall information flow of Task 3.

$u(T^*)$. The monitor theme watches out for the changes that may affect the validity of the generated control actions. Once T^*_{upt} is computed, the generate theme is reactivated to create control actions starting the T^*_{upt} segment. Repeat the process above iteratively until the inputs from Task 2 become invalid.

However, similar to Task 2, there are two problems for this loop. One is that the synchronization of the monitor theme and the generate theme. Specifically, when new $\hat{x}(T^*)$ is generated, the generate theme must wait after the monitor theme is updated to update itself. For this reason, we replace it with the Fig.75 (right) as a solution, where $\hat{x}(T^*)$ goes first into the monitor theme. After the monitor theme is updated, $\hat{x}(T^*)$ flows into the generate theme together with T^*_{upt}.

The other problem is that when more than one of the three themes have to update at the same time due to the change of the respective inputs. Same as Task 2, the concept of the priority is also introduced to solve this problem. The priority of the predict theme is higher than the monitor theme, and the monitor theme is higher than the generate theme. This idea is implemented in EA_{12}.

Therefore, the overall workflow of Task can summarized in Fig.76.

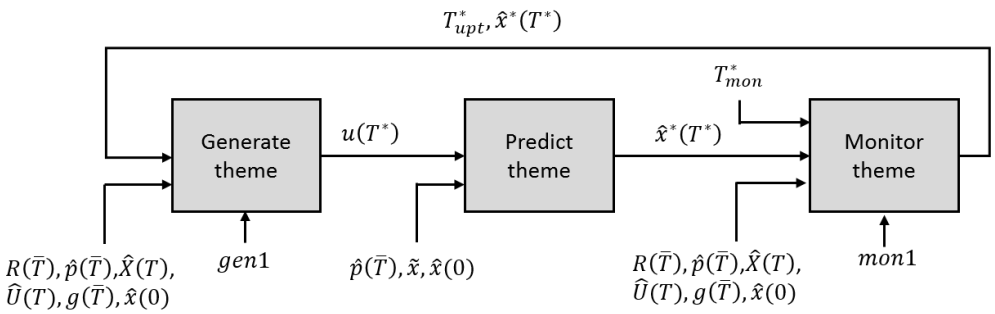


Figure 76: The overall workflow of Task 3.

4.3.2 The generate theme

Two main actions MA_{30} and MA_{31} are defined for the generate theme (Fig.77).

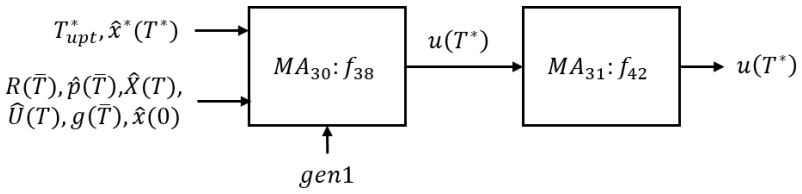


Figure 77: The main action of the generate theme of Task 3.

- $\boldsymbol{MA_{30}}$
  - Transformation: f_{38} is to calculate the control actions. If the control action cannot be found, the controller issues the default action u_d. Furthermore, f_{38} only depicts the value aspect of MA_{30}, but MA_{30} is more than f_{38} especially the timing aspect: when to start/stop/restart f_{38}. On one hand, when no change is detected by the monitor theme, the controller starts to generate the control action for T^*_i when the timeline comes to $t^*_i - \Delta_{T2} - T2$. If no control action can be found before $t^*_i - T2$, then the controller applies u_d before $t^*_i - T3$. (21) guarantees the control action for each segment can be generated in the same way iteratively from the first segment to the last. On the other hand, when a change is detected and a T^*_{upt} is received from the monitor theme, the process described above is also applicable. First, the monitor theme and (21) guarantee the two inequalities below respectively:

    $$t^*_{upt} \geq t_c + T3 \text{ and } t^*_{upt+1} - t^*_{upt} \geq T2 - T3.$$

    Plug the former into the latter, we get a new inequality as below, which means there is always enough time to generate the control action for the segments after t^*_{upt}.

    $$t^*_{upt+1} - t_c \geq T2.$$

    Therefore, the problem is boiled down to generating the control action for T^*_{upt}. If $T2 + t_c \leq t^*_{upt} \leq t_c + \Delta_{T2} + T2$, the controller update the control actions for the segments after T^*_{upt}; if $t^*_{upt} < t_c + T2$, then the controller issues the default control action u_d for $[t_c + T3, t_c + T2]$, and then follows the process in the same way as no change is detected to generate the new control actions for the segments after.
  - Duration: The duration of this action must be as short as possible.
  - Output: $u(T^*)$ is the set of generated control actions for T^*.
  - Input: $\{R(\overline{T}), \widehat{p}(\overline{T}), \widehat{X}, \widehat{U}, g(\overline{T})\}$ from Task 2, because of which the start/stop time of Task 3 is synchronized with Task 2; $T^*_{upt}, \widehat{x}^*(T^*)$ from the monitor theme, and its initial condition is $\widehat{x}(0)$ from Task 2. The initial condition of T^*_{upt} is $T^*_{upt} = T^*_1$.
  - Trigger: NA.
  - Guard: $gen1 = true$.
- $\boldsymbol{MA_{31}}$

- Transformation: f_{42} is to pass the input $u(T^*)$ to the output. However, when $gen1 = false$, no new control action is generated. This action is to assign the default control action u_d to $u(T^*)$ when $u(t_c + T3) = false$.
- Duration: The action is considered instantaneous.
- Output: $u(T^*)$.
- Input: $u(T^*)$.
- Trigger: NA.
- Guard: NA.

In summary, the external interactions of the generate theme of Task 3 can be specified in Fig.78.

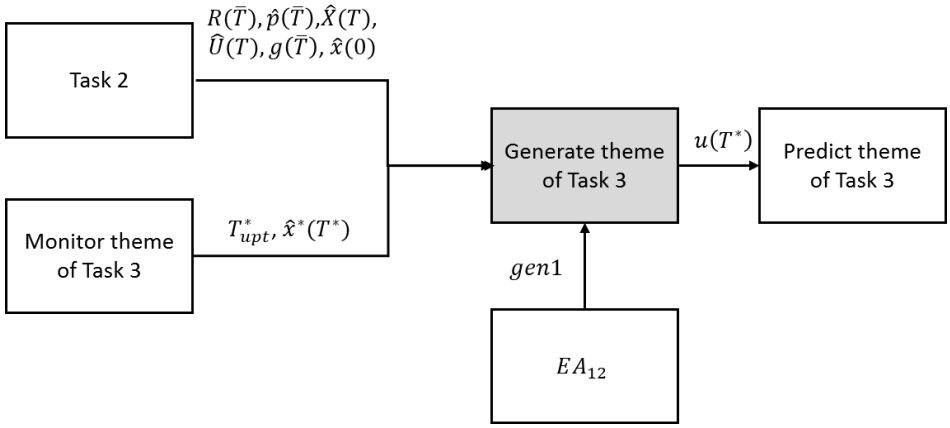


Figure 78: The external interactions of the generate theme of Task 3.

4.3.3 The predict theme

Only one main action MA_{32} is defined for the predict theme (Fig.79).

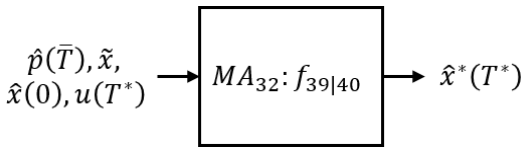


Figure 79: The main action of the predict theme of Task 3.

- $\boldsymbol{MA_{32}}$
  - Transformation: $f_{39|40}$ is to predict and update in real time the system states of the segments where the control actions are defined. If $t_c < st$, then f_{39} is applied; otherwise f_{40} is applied.
  - Duration: The duration of this action must be as short as possible. The first e_{31} time must be removed from the output prediction.

- Output: $\widehat{x}^*(T^*)$ is the prediction of the system states for all the segments that have the control actions defined.
- Input: $\widehat{x}(0)$ is the predicted initial state that comes from Task 2; $\widehat{p}(\overline{T})$ is the predicted value of the parameter that comes from Task 2; $\widetilde{x}$ is the real system state that is observed from the real controlled process; $u(T^*)$ is a set of generated control actions that comes from the generate theme.
- Trigger: NA.
- Guard: NA.

In summary, the external interactions of the predict theme of Task 3 can be specified in Fig.80.

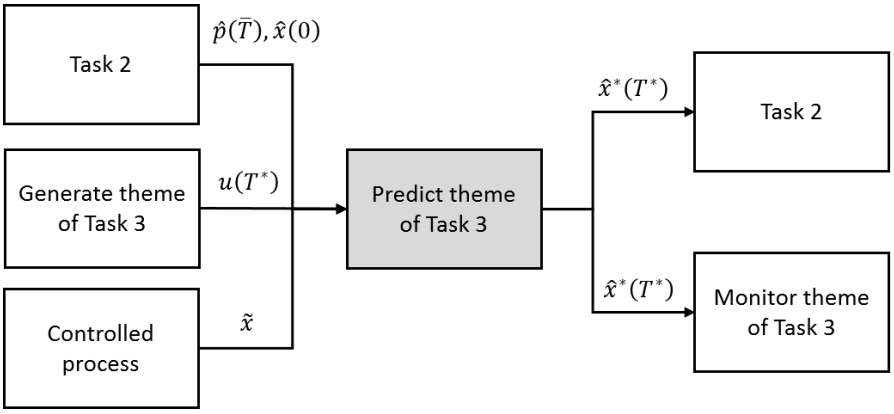


Figure 80: The external interactions of the predict theme of Task 3.

4.3.4 The monitor theme

Only one main action MA_{33} is defined for the monitor theme (Fig.81).

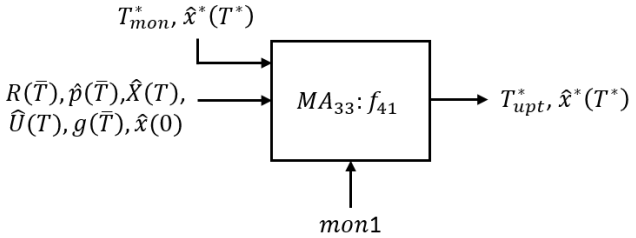


Figure 81: The main action of the monitor theme of Task 3.

- $\boldsymbol{MA_{33}}$
  - Transformation: f_{41} is to monitor the change of the inputs used to generate the control actions, and pinpoint the segments that the respective control actions need to be updated.

- Duration: The duration of this action must be as short as possible.
- Output: T^*_{upt} is the segment from which the control actions must be updated; $\widehat{x}^*(T^*)$ is passed from the input.
- Input: T^*_{mon} is the set of segments whose potential change is monitored; $\widehat{x}^*(T^*)$ is the prediction of the system states for all the segments that have the control actions defined, and is from the predict theme; $\{R(\overline{T}), \widehat{p}(\overline{T}), \widehat{X}, \widehat{U}, g(\overline{T})\}$ is from Task 2; $\widehat{x}(0)$ is from Task 2.
- Trigger: NA.
- Guard: $mon1 = true$.

In summary, the external interactions of the monitor theme of Task 3 can be specified in Fig.82.

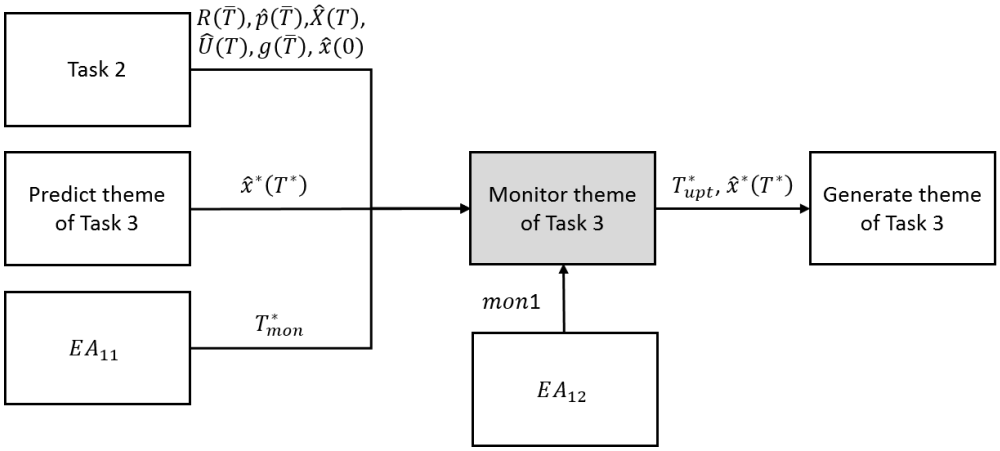


Figure 82: The external interactions of the monitor theme of Task 3.

Appendix D: The reference architecture in a N^2 diagram

	Controlled process	Environment	Higher level of control	Controller of in-behavior	MA1	MA2
MA1	CP_1	E_1	FG			
MA2	$CP_2, \tilde{y}$	E_2		$In-B$	pc^i, pc, pc^o	
MA3	$CP_3, \tilde{y}$	E_3		$In-B$		mst_T,cst_T,nst_T
MA4	CP_1	E_1	FG			
MA5	$CP_2, \tilde{y}$	E_2		$In-B$		
MA6						
MA7	$CP_3, \tilde{y}$	E_3		$In-B$		
MA8		E_3				
MA9	CP_1	E_1	FG			
MA10	CP_4	E_4				
MA11						
MA12	CP_1	E_1	FG			
MA13	CP_4	E_4				
MA14	CP_3	E_3				
MA15	CP_1	E_1				
MA16	CP_4	E_4				
MA17	CP_3					
MA18	CP_1	E_1				
MA19	CP_4	E_4				
MA20						
MA21						
MA22				$\hat{x}^i(t)$		
MA23	$\tilde{x}, \tilde{p}$					
MA24						
MA25	IC_2	IC_1				
MA26	$\tilde{x}, \tilde{p}, \tilde{\dot{x}}, \tilde{y}$					
MA27	IC_4	IC_3				
MA28						
MA29						
MA30						
MA31						
MA32	$\tilde{x}$					
MA33						

	MA3	MA4	MA5	MA6	MA7	MA8
MA1						
MA2						
MA3						
MA4						
MA5	st	pc^i, pc				
MA6			mst_T,cst_T,nst_T, st			
MA7	$y(t), st$	pc				
MA8	$y(sp), st$	pc^o				
MA9						
MA10	$y(t)$					
MA11	sp					
MA12						
MA13	$y(t)$					
MA14	$y(t), sp$					
MA15						
MA16						
MA17						
MA18						
MA19						
MA20						
MA21						
MA22						
MA23						
MA24						
MA25						
MA26						
MA27						
MA28						
MA29						
MA30						
MA31						
MA32						
MA33						

	MA9	MA10	MA11	MA12	MA13	MA14	MA15
MA1							
MA2							
MA3							
MA4							
MA5							
MA6							
MA7							
MA8							
MA9							
MA10	pc, pc^o					$y(t)$	
MA11		msp_T,csp_T,nsp_T				sp	
MA12							
MA13				pc		$y(t)$	
MA14				pc, pc^o	msp_T	$y(t), sp$	
MA15							
MA16						$y(t)$	pc
MA17						$y(t), sp$	pc, pc^o
MA18							
MA19						$y(t)$	
MA20						sp	
MA21							
MA22							
MA23							
MA24							
MA25							
MA26							
MA27							
MA28							
MA29							
MA30							
MA31							
MA32							
MA33							

	MA16	MA17	MA18	MA19	MA20	MA21
MA1						
MA2						
MA3						
MA4						
MA5						
MA6						
MA7						
MA8						
MA9						
MA10						
MA11						
MA12						
MA13						
MA14						
MA15						
MA16		$y(t)$				
MA17	msp_T	$y(t), sp$				
MA18						
MA19		$y(t)$	pc, pc^o			
MA20		sp		msp_T,csp_T,nsp_T		
MA21						
MA22						$Scea, t_d, t_d'$
MA23						
MA24						
MA25						
MA26						
MA27						
MA28						
MA29						
MA30						
MA31						
MA32						
MA33						

	MA22	MA23	MA24	MA25
MA1				
MA2				
MA3				
MA4				
MA5				
MA6				
MA7				
MA8				
MA9				
MA10				
MA11				
MA12				
MA13				
MA14				
MA15				
MA16				
MA17				
MA18				
MA19				
MA20				
MA21				
MA22				
MA23	$g(\bar{T}), R(\bar{T}), \hat{x}(0)$			
MA24		$\hat{x}(\bar{T}),\hat{y}(\bar{T}),\hat{p}(\bar{T})$		
MA25				
MA26				$X, \dot{X}, Y, P$
MA27				
MA28			$\hat{x}(\bar{T}),\hat{y}(\bar{T}),\hat{u}(\bar{T}),\hat{\dot{x}}(\bar{T})$	
MA29			$\hat{y}(\bar{T})$	
MA30	$g(\bar{T}), R(\bar{T}), \hat{x}(0)$			
MA31				
MA32	$\hat{x}(0)$			
MA33	$g(\bar{T}), R(\bar{T}), \hat{x}(0)$			

	MA26	MA27	MA28	MA29	MA30	MA31	MA32
MA1							
MA2							
MA3							
MA4							
MA5							
MA6							
MA7							
MA8							
MA9							
MA10							
MA11							
MA12							
MA13							
MA14							
MA15							
MA16							
MA17							
MA18							
MA19							
MA20							
MA21							
MA22			$\hat{x}(\bar{T}),\widehat{EC}$				
MA23							$x^*(T^*)$
MA24							
MA25							
MA26							
MA27							
MA28		$\widehat{EC}$					
MA29							
MA30		$\hat{X}, \hat{U}$					
MA31					$u(T^*)$		
MA32						$u(T^*)$	
MA33		$\hat{X}, \hat{U}$					$x^*(T^*)$

	MA33	EA1	EA2	EA3	EA4	EA5	EA6
MA1		$stage$				$sebStatus$	
MA2		$stage$				$sebStatus$	
MA3		$stage$				$sebStatus$	
MA4		$stage$				$sebStatus$	
MA5		$stage$				$sebStatus$	
MA6		$stage$				$sebStatus$	
MA7		$stage$				$sebStatus$	
MA8		$stage$				$sebStatus$	
MA9		$stage$				$sebStatus$	
MA10		$stage$				$sebStatus$	
MA11		$stage$				$sebStatus$	
MA12		$stage$				$sebStatus$	
MA13		$stage$				$sebStatus$	
MA14		$stage$				$sebStatus$	
MA15		$stage$				$sebStatus$	
MA16		$stage$				$sebStatus$	
MA17		$stage$				$sebStatus$	
MA18		$stage$				$sebStatus$	
MA19		$stage$				$sebStatus$	
MA20		$stage$				$sebStatus$	
MA21		$stage$					
MA22		$stage$					
MA23		$stage$					
MA24		$stage$					
MA25		$stage$					
MA26		$stage$					
MA27		$stage$					
MA28		$stage$					
MA29		$stage$					
MA30	$x^*(T^*), T^*_{upt}$						
MA31							
MA32							
MA33							

	EA7	EA8	EA9	EA10	EA11	EA12
MA1						
MA2						
MA3						
MA4						
MA5						
MA6						
MA7						
MA8						
MA9						
MA10						
MA11						
MA12						
MA13						
MA14						
MA15						
MA16						
MA17						
MA18						
MA19						
MA20						
MA21		$Scea, t_d$	gen			
MA22	$y(\bar{T}), \hat{p}(\bar{T})$	$Scea, t_d$	gen			
MA23	$\hat{p}(\bar{T})$					
MA24						
MA25						
MA26						
MA27						
MA28	$\hat{p}(\bar{T})$		mon			
MA29	$y(\bar{T})$		mon			
MA30	$\hat{p}(\bar{T})$					$gen1$
MA31						
MA32	$\hat{p}(\bar{T})$					
MA33	$\hat{p}(\bar{T})$				T^*_{mon}	$mon1$

Appendix E: The UAM controller in a N^2 diagram

	Weather service provider	The vehicle	Higher level of control	Traffic surveillance
MA1	$\widehat{w}(t)$	pl, bl	(x_B, h_B, rta)	vt
MA2	$cd, vb, \widehat{w}(t), \widehat{wa}, \widetilde{wa}$	$vm, \tilde{s}, \tilde{v}$		$\hat{s}_{tf}, \hat{v}_{tf}, \tilde{s}_{tf}$
MA3	$\widehat{wa}$	$\tilde{s}, \tilde{v}$		$\hat{s}_{tf}, \hat{v}_{tf}$
MA4	$\widehat{w}(t)$	pl, bl	(x_B, h_B, rta)	vt
MA5	$cd, vb, \widehat{w}(t), \widehat{wa}, \widetilde{wa}$	$vm, \tilde{s}$		$\hat{s}_{tf}, \hat{v}_{tf}, \tilde{s}_{tf}$
MA6				
MA7	$\widehat{wa}$	$\tilde{s}, \tilde{v}$		$\hat{s}_{tf}, \hat{v}_{tf}$
MA9	$\widehat{w}(t)$	pl, bl	(x_B, h_B, rta)	vt
MA10	$\widehat{wa}$			$\hat{s}_{tf}, \hat{v}_{tf}$
MA11				
MA12	$\widehat{w}(t)$	pl, bl	(x_B, h_B, rta)	vt
MA13	$\widehat{wa}$			$\hat{s}_{tf}, \hat{v}_{tf}$
MA14	$\widehat{wa}$			$\hat{s}_{tf}, \hat{v}_{tf}$
MA21				
MA22				
MA23	$\widehat{w}(t), \widetilde{w}$	$\tilde{s}(t), \tilde{\gamma}, \tilde{v}$		
MA24				
MA25	wc, swp	cw, bl, tem, pt		
MA26		$\tilde{v}, \tilde{\gamma}, \tilde{s}, \widetilde{w}$		
MA27	$\widehat{wc}, \widehat{cw}, \widehat{tem}, \widehat{swp}$	$\widehat{bl}$		
MA28				
MA29				
MA30	$\widehat{w}(t)$			
MA31				
MA32	$\widehat{w}(t)$	$\tilde{v}, \tilde{\gamma}, \tilde{s}$		
MA33	$\widehat{w}(t)$			

	Terrain map provider	Airspace info provider	The controller for cruise	Vertiport	MA1
MA1	gh	cat			
MA2	lt, et	$\widehat{ab}, \widetilde{ab}$	$\hat{s}^i(t), \hat{v}^i(t)$		$pc_0^1, pc_0^2, pc_1^i, pc_1, pc_2^i, pc_2, pc_3^i, pc_3, pc_4^i, pc_4$
MA3	lt, et	$\widehat{ab}$	$\hat{s}^i(t), \hat{v}^i(t)$	dc	$pc_0^1, pc_0^2, pc_1^i, pc_1, pc_2, pc_3, pc_4$
MA4	gh	cat		dc	
MA5	lt, et	$\widehat{ab}, \widetilde{ab}$	$\hat{s}^i(t), \hat{v}^i(t)$		
MA6					
MA7	lt, et	$\widehat{ab}$	$\hat{s}^i(t), \hat{v}^i(t)$		
MA9	gh	cat		dc	
MA10		$\widehat{ab}$			
MA11					
MA12	gh	cat		dc	
MA13		$\widehat{ab}$			
MA14	lt, et	$\widehat{ab}$			
MA21					
MA22			$\hat{s}^i(t), \hat{v}^i(t)$		
MA23					
MA24					
MA25	ge, gh			tcl, dp, ttp, vl	
MA26					
MA27				$\widehat{tcl}, \widehat{dp}, \widehat{ttp}, \widehat{vl}$	
MA28					
MA29					
MA30			$\hat{s}^i(t), \hat{v}^i(t)$		
MA31					
MA32			$\hat{s}^i(t), \hat{v}^i(t)$		
MA33			$\hat{s}^i(t), \hat{v}^i(t)$		

	MA2	MA3	MA4	MA5
MA1				
MA2				
MA3	mst_T,nst_T,cst_T			
MA4				
MA5			pc_0^1,pc_0^2,pc_1^i,pc_1, pc_2^i,pc_2,pc_3^i,pc_3, pc_4^i,pc_4	
MA6		st		mst_T,nst_T,cst_T
MA7		$s(T), st$	$pc_0^1, pc_0^2, pc_1, pc_2, pc_3, pc_4$	
MA9				
MA10		$s(T)$		
MA11		rta		
MA12				
MA13		$s(T)$		
MA14				
MA21				
MA22				
MA23				
MA24				
MA25				
MA26				
MA27				
MA28				
MA29				
MA30				
MA31				
MA32				
MA33				

	MA9	MA10	MA12	MA13
MA1				
MA2				
MA3				
MA4				
MA5				
MA6				
MA7				
MA9				
MA10	$pc_0^1, pc_0^2, pc_1, pc_2, pc_4$			
MA11		mst_T, nst_T, cst_T		
MA12				
MA13			$pc_0^1, pc_0^2, pc_1, pc_2, pc_4$	
MA14			$pc_0^1, pc_0^2, pc_1, pc_2, pc_4$	msp_T
MA21				
MA22				
MA23				
MA24				
MA25				
MA26				
MA27				
MA28				
MA29				
MA30				
MA31				
MA32				
MA33				

	MA14	MA21	MA22	MA23
MA1				
MA2				
MA3				
MA4				
MA5				
MA6				
MA7				
MA9				
MA10	$s(T)$			
MA11	rta			
MA12				
MA13	$s(T)$			
MA14	$s(T)$			
MA21				
MA22		$Scea, t_d, t_d,$		
MA23			$(x_i, h_i, rta_i), \hat{s}(0), \hat{v}(0)$	
MA24				$(x_i, h_i, rta_i), \hat{s}(T_i),$ $\hat{v}_i, \hat{\gamma}_i, \widehat{w}(t)$
MA25				
MA26				
MA27				
MA28				
MA29				
MA30			(x_i, h_i, rta_i)	
MA31				
MA32				
MA33			(x_i, h_i, rta_i)	

	MA24	MA25	MA26	MA27	MA28	MA29	MA30
MA1							
MA2							
MA3							
MA4							
MA5							
MA6							
MA7							
MA9							
MA10							
MA11							
MA12							
MA13							
MA14							
MA21							
MA22					$(x_i, h_i, rta_i), \hat{v}^i, \hat{\gamma}^i,$ $\hat{s}(T_i), \hat{w}(t), EC$	$s(\bar{T}), t_d,$	
MA23							
MA24							
MA25							
MA26		EC					
MA27							
MA28	$(x_i, h_i, rta_i), \hat{s}(T_i),$ $\hat{v}_i, \hat{\gamma}_i, \hat{w}(t), \widehat{Vel}_i, \widehat{Des}_i$			EC			
MA29	$\hat{s}(T_i)$						
MA30							
MA31							$u(T^*)$
MA32							
MA33							

	MA31	MA32	MA33	EA1	EA5	EA7	EA8
MA1				$stage$	$sebStatus$		
MA2				$stage$	$sebStatus$		
MA3				$stage$	$sebStatus$		
MA4				$stage$	$sebStatus$		
MA5				$stage$	$sebStatus$		
MA6				$stage$	$sebStatus$		
MA7				$stage$	$sebStatus$		
MA9				$stage$	$sebStatus$		
MA10				$stage$	$sebStatus$		
MA11				$stage$	$sebStatus$		
MA12				$stage$	$sebStatus$		
MA13				$stage$	$sebStatus$		
MA14				$stage$	$sebStatus$		
MA21				$stage$			$Scea, t_d$
MA22				$stage$			
MA23		$\hat{s}^*(t), \hat{y}^*, \hat{v}^*$		$stage$			
MA24				$stage$			
MA25				$stage$			
MA26				$stage$			
MA27				$stage$			
MA28				$stage$			
MA29				$stage$		$s(\bar{T})$	
MA30		$\hat{s}^*(t), \hat{y}^*, \hat{v}^*, T^*_{upt}$					
MA31							
MA32	$u(T^*)$						
MA33		$\hat{s}^*(t), \hat{y}^*, \hat{v}^*$					

	EA9	EA12
MA1		
MA2		
MA3		
MA4		
MA5		
MA6		
MA7		
MA9		
MA10		
MA11		
MA12		
MA13		
MA14		
MA21	gen	
MA22	gen	
MA23		
MA24		
MA25		
MA26		
MA27		
MA28	mon	
MA29	mon	
MA30		$gen1$
MA31		
MA32		
MA33		$mon1$

$\widehat{w}(t)$	Prediction of the headwind.
pl	Weight of the payload.
vt	Vehicle type of the traffic.
bl	Battery level of the vehicle
gh	The ground habitat
cat	The category of the adjacent airspace
cd	Cloud.
vb	Visiability.
$\hat{s}^i(t)$	The projected cruise trajectory
$\hat{v}^i(t)$	The projected cruise speed
vm	Vehicle operation mode
$\hat{s}_{tf}$	The projected traffic trajectory
$\hat{v}_{tf}$	The projected traffic speed
$\tilde{s}$	The current vehicle position
$\tilde{s}_{tf}$	The current traffic position
$\widehat{wa}$	The projected weather impacted area
$\widetilde{wa}$	The current weather impacted area
lt	The location of the terrain
et	The elevation of the terrain
$\widehat{ab}$	The projected airspace boundary
$\widetilde{ab}$	The current airspace boundary
dc	Descent clearance
st	The desired start time of the descent.
$s(T)$	The desired descent trajectory
tcl	The current traffic congestion level.
dp	The current descent procedure.
ttp	The current requirement on the traffic throughput
vl	The current vertiport layout
wc	The current weather condition
swp	The current operational status of the weather service provider
cw	The current crosswind
tem	The current temperature
pt	The payload type
ge	The ground elevation
$\widehat{wc}$	The predicted weather condition
$\widehat{cw}$	The predicted crosswind
$\widehat{tem}$	The predicted temperature
$\widehat{swp}$	The planned operational status of the weather service provider
$\widehat{bl}$	The predicted battery level
$\widehat{tcl}$	The predicted traffic congestion level.
$\widehat{dp}$	The planned descent procedure.
$\widehat{ttp}$	The planned requirement on the traffic throughput
$\widehat{vl}$	The planned vertiport layout

References

[1] C. Specifications, “Acceptable means of compliance for large aeroplanes cs-25,” *European Aviation Safety Agency, Amendment*, vol. 24, no. 10, 2020.

[2] P. Johnston and R. Harris, “The boeing 737 max saga: lessons for software organizations,” *Software Quality Professional*, vol. 21, no. 3, pp. 4–12, 2019.

[3] *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety related systems*. International Electrotechnical Commission, 2000.

[4] *ISO 26262: Road vehicles-Functional safety*. International Standard ISO/FDIS, 2011.

[5] *DO-178C: Software considerations in airborne systems and equipment certification*. RTCA, Incorporated, 2012.

[6] *DO-254: Design Assurance Guidance for Airborne Elecronic Hardware*. RTCA,Incorporated, 2000.

[7] *DO-331: Model-based Development and Verification Supplement to DO-178C and DO-278A*. RTCA, Incorporated, 2011.

[8] *DO-333: Formal Methods Supplement to DO-178C and DO-278A*. RTCA, Incorporated, 2011.

[9] *ARP 4754A:Guidelines for Development of Civil Aircraft and Systems*. SAE, Incorporated, 2010.

[10] S. Maitrehenry, S. Metge, P. Bieber, and Y. Ait-Ameur, “Toward model-based functional hazard assessment at aircraft level,” *Advances in Safety, Reliability and Risk Management: ESREL 2011*, p. 390, 2011.

[11] E. Ruijters and M. Stoelinga, “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools,” *Computer science review*, vol. 15, pp. 29–62, 2015.

[12] J. Dunjó, V. Fthenakis, J. A. Vílchez, and J. Arnaldos, “Hazard and operability (hazop) analysis. a literature review,” *Journal of hazardous materials*, vol. 173, no. 1-3, pp. 19–32, 2010.

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

[14] G. Le Lann, “An analysis of the ariane 5 flight 501 failure-a system engineering perspective,” in *Proceedings International Conference and Workshop on Engineering of Computer-Based Systems*, pp. 339–346, IEEE, 1997.

[15] *ARP 4761:GUIDELINES AND METHODS FOR CONDUCTING THE SAFETY ASSESSMENT PROCESS ON CIVIL AIRBORNE SYSTEMS AND EQUIPMENT*. SAE, Incorporated, 1996.

[16] N. G. Leveson and J. P. Thomas, “Stpa handbook,” *Cambridge, MA, USA*, 2018.

[17] T. Prosvirnova, *AltaRica 3.0: a model-based approach for safety analyses*. PhD thesis, Ecole Polytechnique, 2014.

[18] B. Larson, J. Hatcliff, K. Fowler, and J. Delange, “Illustrating the aadl error modeling annex (v. 2) using a simple safety-critical medical device,” *ACM SIGAda Ada Letters*, vol. 33, no. 3, pp. 65–84, 2013.

[19] S. Procter and P. Feiler, “The aadl error library: An operationalized taxonomy of system errors,” *ACM SIGAda Ada Letters*, vol. 39, no. 1, pp. 63–70, 2020.

[20] P. Feiler and J. Delange, “Automated fault tree analysis from aadl models,” *ACM SIGAda Ada Letters*, vol. 36, no. 2, pp. 39–46, 2017.

[21] M. Machin, E. Saez, P. Virelizier, and X. de Bossoreille, “Modeling functional allocation in altarica to support mbse/mbsa consistency,” in *International Symposium on Model-Based Safety and Assessment*, pp. 3–17, Springer, 2019.

[22] P. Bieber, C. Bougnol, C. Castel, J.-P. H. C. Kehren, S. Metge, and C. Seguin, “Safety assessment with altarica,” in *Building the Information Society*, pp. 505–510, Springer, 2004.

[23] T. Prosvirnova, M. Batteux, P.-A. Brameret, A. Cherfi, T. Friedlhuber, J.-M. Roussel, and A. Rauzy, “The altarica 3.0 project for model-based safety assessment,” *IFAC Proceedings Volumes*, vol. 46, no. 22, pp. 127–132, 2013.

[24] H. Mortada, T. Prosvirnova, and A. Rauzy, “Safety assessment of an electrical system with altarica 3.0,” in *International Symposium on Model-Based Safety and Assessmemt*, pp. 181–194, Springer, 2014.

[25] M. Tlig, M. Machin, R. Kerneis, E. Arbaretier, L. Zhao, F. Meurville, and J. Van Frank, “Autonomous driving system: Model based safety analysis,” in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 2–5, IEEE, 2018.

[26] S. Kabir, K. Aslansefat, I. Sorokos, Y. Papadopoulos, and Y. Gheraibia, “A conceptual framework to incorporate complex basic events in hip-hops,” in *International Symposium on Model-Based Safety and Assessment*, pp. 109–124, Springer, 2019.

[27] D. Chen, N. Mahmud, M. Walker, L. Feng, H. Lönn, and Y. Papadopoulos, “Systems modeling with east-adl for fault tree analysis through hip-hops,” *IFAC Proceedings Volumes*, vol. 46, no. 22, pp. 91–96, 2013.

[28] M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, and S. Tonetta, "Safety assessment of altarica models via symbolic model checking," *Science of Computer Programming*, vol. 98, pp. 464–483, 2015.

[29] M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I. Lee, "Model-driven safety analysis of closed-loop medical systems," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 3–16, 2012.

[30] M. Althoff, *Reachability analysis and its application to the safety assessment of autonomous cars*. PhD thesis, Technische Universität München, 2010.

[31] P. Sinha, "Architectural design and reliability analysis of a fail-operational brake-by-wire system from iso 26262 perspectives," *Reliability Engineering & System Safety*, vol. 96, no. 10, pp. 1349–1359, 2011.

[32] S. D. Krach, "Model-based architecture robustness analysis for software-intensive autonomous systems," in *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 104–107, IEEE, 2017.

[33] O. Lisagor, J. McDermid, and D. Pumfrey, "Towards a practicable process for automated safety analysis," in *24th International system safety conference*, vol. 596, p. 607, Citeseer, 2006.

[34] F. Ortmeier and W. Reif, "Failure-sensitive specification: A formal method for finding failure modes/universität augsburg, institut für informatik. 2004 (2004-3)," *Forschungsbericht. www. uni-augsburg. de*.

[35] O. Lisagor, L. Sun, and T. Kelly, "The illusion of method: Challenges of model-based safety assessment," in *28th international system safety conference (ISSC)*, Citeseer, 2010.

[36] P. Braun, J. Philipps, B. Schätz, and S. Wagner, “Model-based safety-cases for software-intensive systems,” *Electronic Notes in Theoretical Computer Science*, vol. 238, no. 4, pp. 71–77, 2009.

[37] K. L. Hobbs, A. R. Collins, and E. M. Feron, “Risk-based formal requirement elicitation for automatic spacecraft maneuvering,” in *AIAA Scitech 2021 Forum*, p. 1122, 2021.

[38] M. D. Mesarovic and Y. Takahara, *General systems theory: mathematical foundations*. Academic press, 1975.

[39] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[40] M. Sun, K. Rand, and C. Fleming, “4 dimensional waypoint generation for conflict-free trajectory based operation,” *Aerospace Science and Technology*, vol. 88, pp. 350–361, 2019.