Software and Systems Modeling
https://doi.org/10.1007/s10270-024-01209-6

REGULAR PAPER

A system-theoretic assurance framework for safety-driven systems engineering

Alexander Ahlbrecht[1] · **Jasper Sprockhoff**[1] · **Umut Durak**[1]

Received: 20 November 2023 / Revised: 1 August 2024 / Accepted: 2 August 2024

Abstract
The complexity of safety-critical systems is continuously increasing. To create safe systems despite the complexity, the system development requires a strong integration of system design and safety activities. A promising choice for integrating system design and safety activities are model-based approaches. They can help to handle complexity through abstraction, automation, and reuse and are applied to design, analyze, and assure systems. In practice, however, there is often a disconnect between the model-based design and safety activities. At the same time, there is often a delay until recent approaches are available in model-based frameworks. As a result, the advantages of the models are often not fully utilized. Therefore, this article proposes a framework that integrates recent approaches for system design (model-based systems engineering), safety analysis (system-theoretic process analysis), and safety assurance (goal structuring notation). The framework is implemented in the systems modeling language (SysML), and the focus is placed on the connection between the safety analysis and safety assurance activities. It is shown how the model-based integration enables tool assistance for the systematic creation, analysis, and maintenance of safety artifacts. The framework is demonstrated with the system design, safety analysis, and safety assurance of a collision avoidance system for aircraft. The model-based nature of the design and safety activities is utilized to support the systematic generation, analysis, and maintenance of safety artifacts.

Keywords MBSE · Safety · STPA · SysML · GSN

1 Introduction

As safety-critical systems continue to evolve, they are increasingly characterized by interconnected components and software-defined functions [1]. The resulting complexity opens up risks for design errors and safety issues. To tackle the complexity, adapted approaches are necessary that enable a systematic integration of design and safety activities for modern safety-critical systems. A current trend in dealing with complexity is the extensive use of models for system design, safety analysis, and safety assurance [2–5]. This is also visible in the related approaches.

A recent approach for system design is model-based systems engineering (MBSE) [6]. MBSE uses the formalized application of modeling to guide the system development throughout all life cycle phases [7]. For the safety analysis of complex systems, the System-Theoretic Process Analysis (STPA) [8] has gained relevance by focusing on system interactions and specification errors [9]. Finally, for safety assurance, the creation of safety cases in Goal Structuring Notation (GSN) [10] has proven its suitability.

Recent literature highlights the benefits of integrating these recent approaches for system design (MBSE), safety analysis (STPA), and safety assurance (GSN) [11–14]. A promising solution would be a model-based integration of these approaches. However, current model-based frameworks only cover the integration of the system design with traditional safety analyses [4, 5]. Hence, this article complements with a model-based integration of recent approaches in

Communicated by Juergen Dingel.

Jasper Sprockhoff and Umut Durak have contributed equally to this work.

✉ Alexander Ahlbrecht
alexander.ahlbrecht@dlr.de

Jasper Sprockhoff
jasper.sprockhoff@dlr.de

Umut Durak
umut.durak@dlr.de

[1] Institute of Flight Systems, German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Brunswick, Germany

Published online: 13 September 2024

form of a system-theoretic assurance framework. Particular focus is placed on the connection of the safety analysis and safety assurance activities. The framework is implemented in an MBSE tool and used to highlight the advantages of a model-based integration. For instance, it is elaborated and demonstrated how safety activities such as the generation and assessment of a safety case can be supported.

To elaborate the framework, the article is structured in the following manner. First, background information and related work are introduced in Sect. 2 and 3. Then, the resulting framework is elaborated in Sect. 4. Subsequently, the application of the framework is demonstrated in Sect. 5. Finally, a discussion is provided in Sect. 6, while a conclusion follows in Sect. 7.

2 Background

2.1 Model-based system design

To design complex systems, systematic approaches are necessary. MBSE is a development paradigm that uses modeling to guide the system development throughout the whole system life cycle [7]. It builds on the systematic nature of Systems Engineering (SE) and formalizes its application. To effectively apply MBSE, a suitable method, language, and tool are required [15]. In terms of the method, multiple frameworks are available and help to support the systematic application of the MBSE paradigm. Examples include MagicGrid [16], Arcadia [17], and OOSEM [6]. In terms of languages, the systems modeling language (SysML) is one of the most common. SysML is a graphical modeling language that currently builds on the unified modeling language (UML) [18]. In the second SysML version[1] that is currently under development [19], the language will receive its own meta-model and include extended features such as a textual representation. Using a semi-formal language like SysML, automation can be implemented through scripting languages in various MBSE tools [20]. Moreover, it is possible to extend SysML modeling capabilities by creating stereotypes. Stereotypes, allow the integration of domain-specific modeling extensions or other model-based approaches (e.g., GSN and STPA) into MBSE environments.

2.2 Model-based safety analysis

In addition to the systematic design, adapted analyses are necessary to determine the safety of complex and software-intensive systems. Accordingly, STPA targets to identify software-related issues such as incomplete specifications or inadequate system interactions [9]. Because of its adjusted

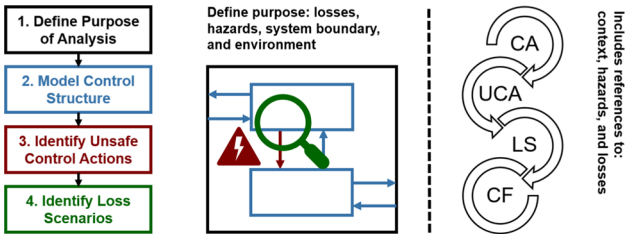


Fig. 1 Steps (left) and relations (right) of STPA

focus, it is also incorporated as a method in the appendix of the recent ISO 21448 standard, which focuses on assuring Safety Of The Intended Functionality (SOTIF) [21]. Moreover, a guidance describing how STPA can be applied during the typical aviation safety process is in development [22]. In the following, STPA's top-down analysis approach is elaborated in accordance with the left side of Fig. 1.

Initially, the purpose of the analysis is defined by specifying the system, its boundary, and the environment. Complementary, losses that should be prevented are selected and the corresponding hazards derived. In the next step, a control structure of the system is modeled that describes the components, their control hierarchy, and related interactions in the form of control actions (CA) and feedback. This control structure is then used to identify potential unsafe control actions (UCA) that can cause a hazardous system state. To uncover UCAs, CAs are analyzed in various scenarios that help to identify if the CA can pose a risk by being executed inadequately or not being executed. After identifying UCAs, the goal is to determine loss scenarios (LS) and their contributing Causal Factors (CF) that would lead to the execution of a UCA. Finally, mitigations (M) are derived for each LS to prevent their occurrence during system operation. Ms can require adjustments of the architecture or further analyses to increase the confidence in the system design. A detailed elaboration of the process is given in the STPA-Handbook [8], while the lessons learned are discussed in the SAE-J3187 Guidance [23]. Recently, the SAE-J3307 development was initiated which targets the development of a STPA standard [24].

In previous work by the authors, it was demonstrated how this process can be integrated into an MBSE environment by using a SysML profile [25]. The integration allows for advanced supporting functionality such as automated coverage indications [26] and STPA-related change impact considerations [27]. Another advantage of using a model-based STPA is that a traceability between the resulting artifacts can be established. Accordingly, in [26], it was shown how a STPA-related summary can be automatically generated including the elements and relations displayed on the right side of Fig. 1.

[1] https://github.com/Systems-Modeling/SysML-v2-Release.

2.3 Model-based safety assurance

Following the topics of design and safety, a model-based approach to assurance in form of assurance cases is introduced. Assurance cases are used to justify a claim of the system under consideration with a systematic argument based on created evidence [28]. In this article, the focus is on arguing the safety of the system with an assurance case. An assurance case with this specific focus is also referred to as a safety case. The argument is created by thoroughly breaking down an overarching goal into smaller sub-goals until justification with appropriate evidence is feasible.

A popular choice for constructing and visualizing safety cases is GSN. It is important to mention that there are alternative approaches such as the structured assurance case metamodel (SACM) [29]. The recent GSN standard [10] defines multiple elements such as goal (G.), strategy (S.), context (C.), assumption (A.), and justification (J.), which are connected by defined relationships. The definitions are as follows. Goal "presents a claim that forms a part of the argument". Strategy "describes the inference that exists between a goal and its supporting goal(s)". Context "presents a contextual artifact. This can be a reference to contextual information, or a statement". Assumption "presents an intentionally unsubstantiated statement". Justification "presents a statement of rationale" [10].

Because safety cases exist for multiple decades, tool support has been established as summarized in [30]. Although safety cases provide a systematic way of structuring an assurance argument, they should only be viewed as a framework and do not guarantee safety [31]. Ultimately, the dedication and effort of the safety activities determine the meaningfulness of the safety case.

3 Related work

In recent literature, possibilities and advantages when integrating the presented approaches are highlighted. For instance, [11] presents an approach to combine SysML with STPA to enable a systematic development and verification of systems. Moreover, [12] combines STPA with a safety case to target the assurance of an automotive perception system. Complementary, [13] discusses how the STPA can be linked to certification through a GSN-based safety case pattern. Finally, the risk analysis and assessment modeling language (RAAML) [14] proposes a standardization for the usage of both STPA and GSN within SysML.

Even though the related work already builds a foundation to integrate the approaches, the model-based integration is currently limited. For instance, RAAML proposes an integration, but does not provide any guidance on how to systematically apply the model-based approaches together.

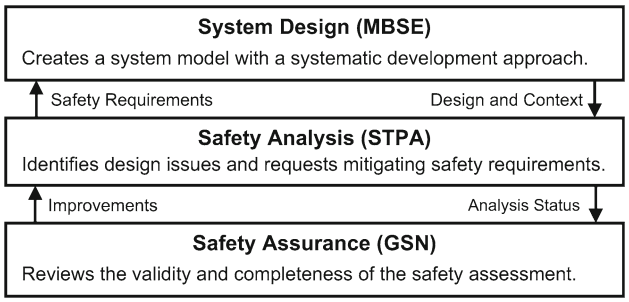


Fig. 2 Framework overview

At the same time, the related work shows that a model-based integration can be very valuable to assist the complex system development and verification tasks. Prominent examples are AdvoCATE and OpenCERT, which not only provide a framework, but also tool support to integrate safety and assurance activities [5, 32]. These model-based frameworks also use the model-based nature of the approaches to provide automated assistance in activities such as the construction and maintenance of safety cases [5]. However, similar to [4], these frameworks are currently focusing on traditional safety analyses and do not cover novel safety analysis approaches such as the STPA.

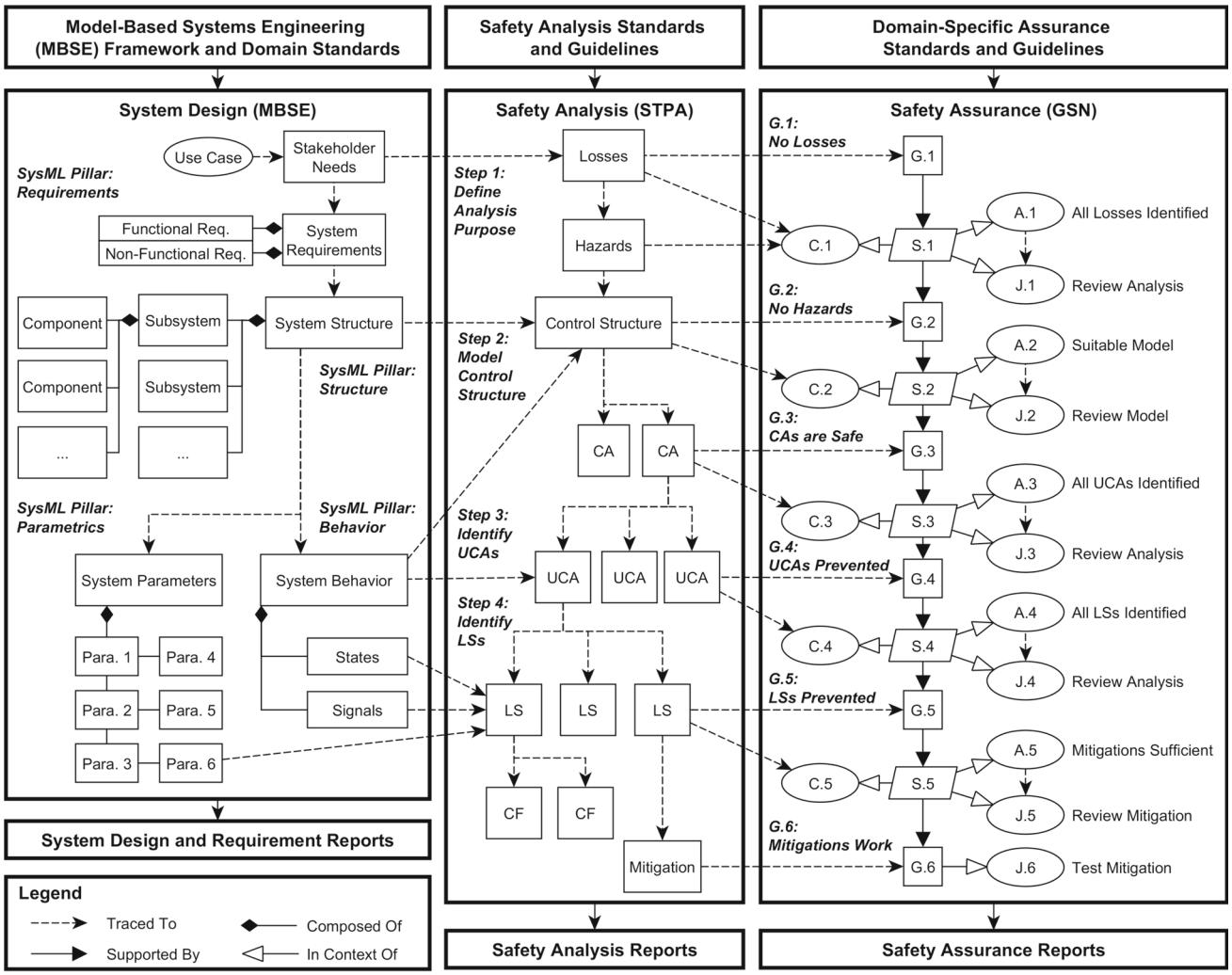
Considering the related work, a suitable baseline for the envisioned integration of system design, safety analysis, and safety assurance is available. However, there is no related work that covers a model-based and end-to-end integration of the recent approaches.

4 Assurance framework

In this article, valuable concepts of the related work are combined into a model-based end-to-end framework for system design, safety analysis, and safety assurance as shown in Fig. 2.

4.1 Framework overview

To establish an efficient integration for the development of complex safety-critical systems, the possibility for assisting automation is essential. Hence, a sufficient traceability and formality is required that enables the use of automation. For this reason, explicit traceability relationships between elements are defined throughout the framework as highlighted in Fig. 3. The model-based fashion and traceability in the framework enable the implementation of assisting automation that will be explained in more detail in Sect. 4.2. In the following, the structure of the framework will be elaborated.



Abbreviations: Control Action (CA), Unsafe Control Action (UCA), Loss Scenario (LS), Causal Factor (CF), Goal (G), Strategy (S), Context (C), Assumption (A), Justification (J)

Fig. 3 System-theoretic assurance framework for system design, safety analysis, and safety assurance

4.1.1 System design

For the system design, it is recommended to follow a systematic development approach according to an MBSE framework. For the proposed assurance framework displayed in Fig. 3, SysML was selected as the MBSE language. Reasons are the extensive utilization in research and industry [33], the possibility for different views on the system [18], and continuous improvements of the language [19].

In the assurance framework, all four pillars of SysML (requirements, structure, behavior, and parametrics [18]) are utilized to provide a holistic overview of the developed system. Requirements are derived by considering the use cases that are relevant for the respective stakeholders. These stakeholder requirements are then translated into system requirements and serve as the baseline for the design. Afterward, the system structure is defined and refined over time. Complementary, the system behavior is modeled as well as the parameters characterizing the system. By utilizing a model-based approach for the system design, a traceability throughout the system development can be established. From the system model, requirement reports or system design documents can be generated.

4.1.2 Safety analysis

In addition to the system design activities, a parallel consideration of the system's safety is proposed in the assurance framework. Consequently, it is shown in Fig. 3 how a simultaneous safety analysis (STPA) can be established and linked. To correctly execute the STPA, the corresponding guidance documents [8, 23] serve as an input.

In Fig. 3, it is shown how all four steps of the STPA are linked through the related artifacts. At the same time, a trace-

ability is established to the corresponding parts of the system model. For instance, losses and hazards can be derived with the help of the stakeholder needs. Moreover, the control structure represents the system structure and builds the baseline for the safety analysis. Similarly, the behavior of the system is integrated into the findings of the safety analysis. In the end, Ms are derived and refine the system requirements with a safety perspective. For readability purposes, this relationship is not displayed in Fig. 3. Overall, the traceability between system design and safety analysis activities allows for a safety-driven design process [27].

4.1.3 Safety assurance

Even with a systematic safety analysis, it is not guaranteed that every aspect is fully covered. To improve the confidence in the safety analysis and thus the system design, each part of the analysis is examined from an assurance point of view. Accordingly, it is shown in Fig. 3 how all safety artifacts of the STPA are traced to a corresponding layer in the safety case. To be able to consistently refer to a layer in the safety case, the term *safety case level* is used throughout this article. The levels are abbreviated with Lx where $x \in \{1, 2, 3, 4, 5, 6\}$. All safety case levels are based on GSN elements and represent the following argument.

The overarching goal is to build a safe system that does not trigger any losses (*G.1*). In the STPA, the idea is to mitigate all system hazards to prevent the loss occurrence (*G.2*). To mitigate the hazards, no UCA should be executed. Hence, the STPA analyzes each CA to identify if it could be unsafe (*G.3*). If a UCA is identified, it should be assured that the UCA is not executed during system operation (*G.4*). Consequently, LSs leading to the UCA execution have to be prevented (*G.5*). If the M for the LS is implemented and verified (*G.6*), the corresponding argument branch is completed.

In addition to the basic breakdown of the safety argument through goals and strategies, context, assumption, and justification elements are utilized. With context elements, references to the elements of the safety analysis are established. At the same time, each safety case level makes use of assumptions and justifications to establish a valid safety argument. For instance, if the analysis focuses on the prevention of losses to ensure the system's safety (*G.1* in Fig. 3), it is important that every loss is identified. Hence, an assumption (*A.1* in Fig. 3) is derived and traced to a justification (*J.1* in Fig. 3) that enforces a rigorous review of the loss identification process.

4.2 Framework implementation

Because the integration of the system design and safety analysis activities were elaborated, implemented, and demonstrated in previous work by the authors [25, 27, 34], the focus of the implementation and demonstration in this article is on the model-based integration of the safety analysis and safety assurance activities. The goal is to enable an automatic generation of a safety case baseline and to support the assessment through the model-based nature and traceability of all approaches.

4.2.1 Modeling concepts

To implement the automated safety case generation, a sufficient integration of MBSE, STPA, and GSN is required. Hence, appropriate modeling concepts are necessary. For demonstration purposes, the idea was to establish a proof-of-concept implementation using SysML to integrate all approaches. In previous work, it was demonstrated how STPA and MBSE can be integrated allowing for automated creation and validation of analysis tasks [25]. Moreover, the RAAML provides guidance on how to model safety cases in a SysML model using stereotypes for the GSN elements and relationships. However, the RAAML [14] implementation of the GSN standard can benefit from application-specific adjustments. For instance, for each safety case level addressed by the concept, specific element references are always required. At the safety case L1, a reference to the loss elements of the system under consideration establishes a clear baseline for the safety assessment. Hence, the stereotypes were adjusted to include the appropriate references as shown in Fig. 4. Specifically, the goal (green) and context (black) elements were refined for each safety case level and include a reference to a corresponding STPA element (grey). Each goal references the element which the corresponding safety case part addresses. The context elements, however, reference all related STPA elements, including the element that is addressed in the next safety case level. An example is *GoalHazardsL2* which references the related hazards, while the *ContextHazardsL2* links the CAs that are addressed at safety case L3. Overall, the integration of both SysML profiles (STPA and GSN) enables the safety case generation and assessment.

4.2.2 Safety case pattern

Since the goal is to automatically generate the safety case from a model-based STPA execution, a concrete safety case pattern is required. Therefore, the pattern defined in [13] was selected and adapted. A systematic structure of the safety case is achieved by covering each step of the STPA execution with a respective segment of the safety case. With this approach, both the STPA and the safety case benefit from each other. The STPA provides a safety-driven pattern for the safety case by explicitly determining the potentially unsafe parts of the system that must be analyzed. At the same time, the safety case adds a holistic view and assurance perspec-

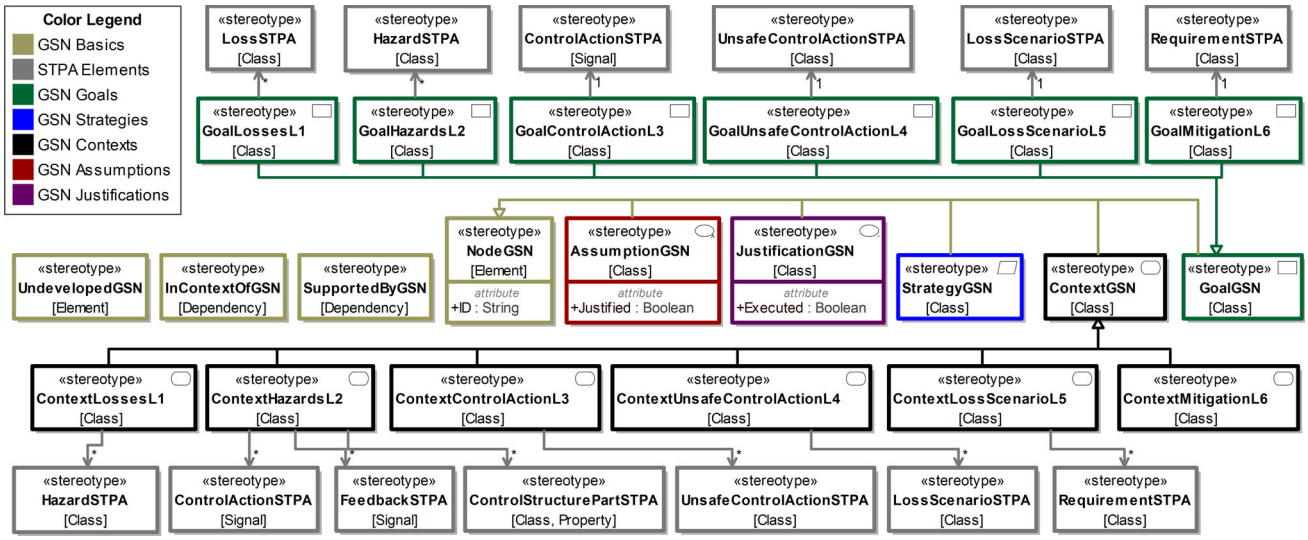


Fig. 4 Safety case stereotypes with references to stereotypes of the STPA profile

tive by demanding a systematic analysis of assumptions and appropriate justifications for each step of the STPA. The proposed STPA-based safety case pattern is illustrated in Fig. 5.

In comparison with the pattern defined in [13], a separate layer for the CAs was added. This extension was applied to represent the typical STPA relationship structure shown on the right side of Fig. 1. Since CAs are explicitly analyzed during the STPA process, a separate assurance layer can help to identify and prevent related mistakes. For instance, this layer enforces a review of UCAs for each CA and makes it obvious when the analysis of a CA is forgotten or not executed with the necessary rigor. Moreover, a justification was added to the initial pattern of [13] to ensure the validity of each assumption. At each safety case level, a similar structure of GSN elements is applied, as depicted in Fig. 5.

First, the goal sets the target for the safety case level. For instance, the goal *G.1* at safety case L1 is that the system is free of unacceptable risks leading to losses (*No Ls*). Attached to the goal, a subsequent strategy defines the approach of how the goal can be translated into one or multiple supporting goals. In addition, a context element is attached to the strategy at each safety case level and provides links to all related elements of the model-based STPA.

At the same time, each safety case level makes use of assumptions. Assumptions can be related to verification or validation of the STPA activities. For instance, a verification-related assumption is that all relevant losses related to the system under investigation are identified (*A.1.1* in Fig. 5). In contrast, a validation-related assumption is that a suitable approach is chosen to mitigate the hazards (*A.1.2* in Fig. 5).

The majority of assumptions must only be determined once and can be reused for future applications. However, each system is different and therefore may add new assumptions or invalidate ones that were previously derived. Accordingly, it is recommended to check each assumption for the system under development and adapt and extend them where necessary. To give an idea of possible activities to justify assumptions, justification measures are linked via trace relationships in Fig. 5. The connection between justifications and assumptions is adapted in comparison with the typical use in the GSN standard [10] and introduced to enforce a process where assumptions and justifications can be explicitly checked by the modeling tool.

Due to the relevance of assumptions and justifications for the proposed process, detailing notes are provided in Fig. 5. For instance, a systematic analysis of the model (*A.2.2*) helps to improve the model's completeness. In general, the model quality is essential and determines the correctness and completeness of the related safety analysis findings [35]. Moreover, a thorough review of the analysis findings (*A.3.2*, *A.4.2*) helps to obtain an impression of the analysis depth and potential weaknesses. Overall, the template of Fig. 5 and related process adds another analysis dimension to the safety case pattern of [13].

4.2.3 Safety case generation

After elaborating the overarching approach that combines the selected approaches, it will now be explained how properties such as efficiency can be established. A foundation for this is the model-based and therefore semi-formal nature of all approaches. This is because it allows the incorporation of assisting automation. In this article, the focus of automation is on the systematic generation of the safety case baseline and the assessment of the safety case properties.

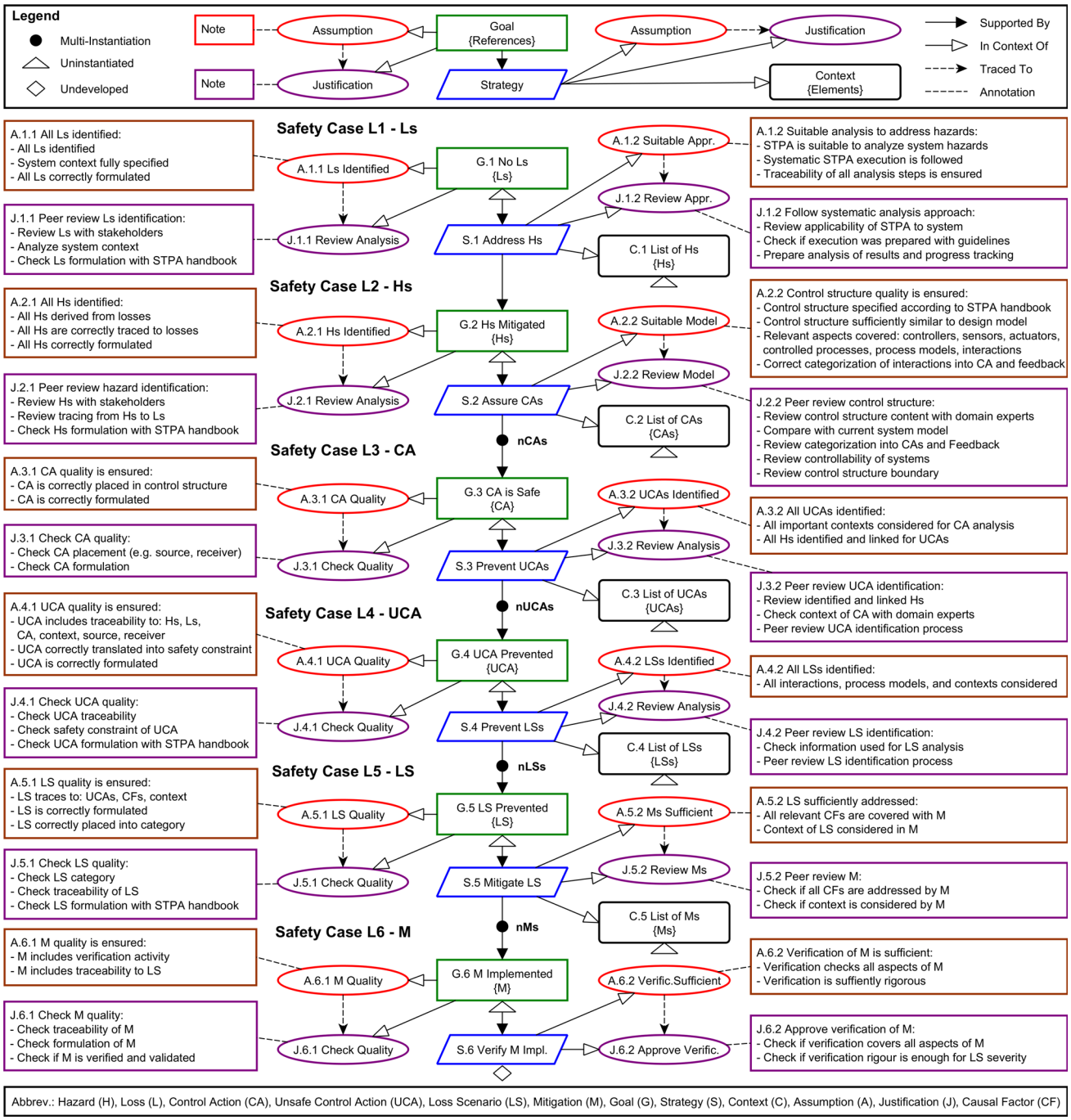


Fig. 5 Safety case pattern with elements and relations for each safety case level

First, the functionality to automatically create a GSN-based safety case is elaborated. For this functionality, the traceability of the model-based STPA implementation is used which is established according to the relations depicted in the safety perspective of Fig. 3. Essentially, a model-based *STPA Summary*[2] can be generated and used as an input for the safety case generation. In Algorithm 1, the implementation is shown in pseudo-code. To express a dependency between two elements in Algorithm 1, a subscript is used. For instance, UCA_{ca} means that only UCAs related to the respective CA are used. The same principle applies to LS_{uca} and M_{ls}.

With Algorithm 1, the safety case is generated in a depth-first manner. Initially, the GSN structure for the safety case

[2] Includes all model-based STPA elements (Losses, Hazards, CAs, feedback, UCAs, LSs, CFs, Ms) as well as their relations.

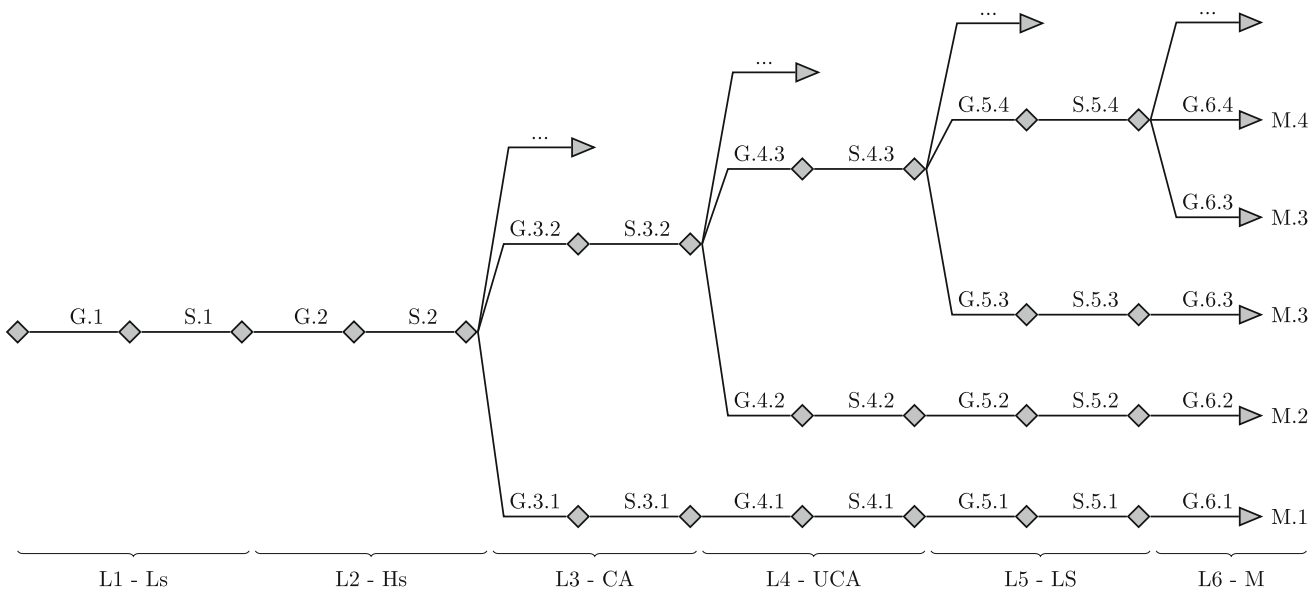


Fig. 6 Example of generated safety case structure with goals (G), strategies (S), and mitigations (M)

Algorithm 1: Safety Case Generation

Input: $STPA\ Summary$[2]
Output: $Safety\ Case$
$createGsnStructureL1(Losses, Hazards)$;
$createGsnStructureL2(Hazards, CAs, Feedback)$;
$connectL1andL2(StrategyL1, GoalL2)$;
forall the $ca \in CA$ **do**
  $createGsnStructureL3(ca, UCA_{ca})$;
  $connectL2andL3(StrategyL2, GoalL3)$;
  forall the $uca_{ca} \in UCA_{ca}$ **do**
    $createGsnStructureL4(uca_{ca}, LS_{uca})$;
    $connectL3andL4(StrategyL3, GoalL4)$;
    forall the $ls_{uca} \in LS_{uca}$ **do**
      $createGsnStructureL5(ls_{uca}, M_{ls})$;
      $connectL4andL5(StrategyL4, GoalL5)$;
      forall the $m_{ls} \in M_{ls}$ **do**
        $createGsnStructureL6(m_{ls})$;
        $connectL5andL6(StrategyL5, GoalL6)$;

```
// If an element already exists (e.g., LS
   or M), only the connect function is
   applied to link the existing element
```

L1 is created by the *createGsnStructureL1()* function. This means that the goal, strategy, context, assumptions, and justifications are created as well as their relationships (see Fig. 5). Since the *GoalLossesL1* references the losses while the *ContextLossesL1* references the hazards (see Fig. 4), both losses and hazards are defined as the inputs for the *createGsnStructureL1()* function. When using a model-based safety analysis approach, it is possible to automatically query the model for the required elements. In the next step of Algorithm 1, the safety case L2 is generated in a similar fashion. After creating the elements of safety case L1 and L2, the safety case levels are linked by connecting the strategy and goal elements as defined in Fig. 5. This procedure is repeated until the safety case is created in a structure similar to Fig. 6. Starting from safety case L3, the main difference is that each safety case level can cover multiple elements. For instance, on safety case L3, the safety case assesses each CA separately. Hence, the elements defined for safety case L3 in Fig. 5 are created for each CA. Essentially, this allows the safety case to branch out and expand in width as visible in Fig. 6.

When an M is used to address multiple LSs, Algorithm 1 would create multiple instances for the same M. In Fig. 6, this possibility is highlighted with *G.6.3*, which is linked to *S.5.3* and *S.5.4*. This duplication can also happen between safety case L4 and L5, when one LS involves multiple UCAs. To avoid duplicates, the algorithm should always check if an element already exists before creating it. If it already exists, the corresponding element can be linked and no new one has to be created. For the implementation of Algorithm 1 in an MBSE tool, scripting functionality can be used. Therefore, MBSE tools provide an Application Programming Interface (API) allowing to search for or create elements. The proof-of-concept implementation in this article was implemented with the Cameo Systems Modeler MBSE tool. In Sect. 5, the safety case generation is demonstrated.

4.2.4 Safety case assessment

Since a complex system will necessitate a large safety case, only generating the safety case baseline is not sufficient. In addition, assistance to efficiently work with the generated safety case is required to enable the handling of complex systems. Hence, safety case visualization, checking of safety

properties, and tracking of the safety progress are elaborated and demonstrated in this article.

Safety case visualization The first way to assist is a multi-layered depiction of the safety case. Visualization is important to deal with the size of a safety case for complex systems and to enable usability. Fortunately, the characteristics of a model support a representation in multiple ways. In the following, typical views to represent models are explained.

- ◇ *Diagram View*: Diagrams have their strength at providing a holistic view of the model by showing elements, properties, and relationships all at once. They are not only good at displaying but also for editing and adding of information.
- ◇ *Table View*: Tables excel at representing large amounts of information in a compact form. Therefore, they are a valuable tool to give a holistic overview of elements with multiple properties.
- ◇ *Matrix View*: Matrices are able to give a quick overview of the relationships between multiple elements. They can also be used to modify the relationships as required.
- ◇ *Relationship Map*: Relationship maps are excellent at giving an overview of the relationships between elements across multiple levels.

As elaborated, each of these views has its own advantages and can be used to analyze specific aspects of the safety case. To demonstrate their application, each view will be used to highlight parts of the safety case in the demonstration Sect. 5.

Safety case V&V The second way to support engineers in assessing the safety case is to implement checks that verify and validate its properties. To give an idea of how such checks can look like, some examples are provided in the following.

- ◇ Check for assumptions without justifications
- ◇ Check for justifications that were not executed
- ◇ Check for goals without element references
- ◇ Check for contexts without element references
- ◇ Check for undeveloped goals
- ◇ Check for elements without relationships

The first check is related to the statement that it is important to justify each assumption for the validity of the safety case. Moreover, if a justification activity is derived, it should be ensured that it is executed. Otherwise, the validity would still be in question. Further checks help to identify goals and context elements with missing references. Since the safety case is automatically generated by using the traceability of the model-based STPA, missing element references indicate that parts of the STPA were not executed to the full extent or that links were forgotten. For instance, no LSs references at the context of safety case L4 would indicate that no LSs were discovered for the corresponding UCA. Since it is unlikely that no LS exists for a UCA, the corresponding context element should be highlighted. Similarly, an "undeveloped" annotation on a goal element indicates that there was no further refinement. Checking and highlighting such elements helps to ensure that this refinement is not forgotten, which is especially helpful in the context of a large safety case for a complex system. Finally, checks should be implemented to identify the elements without relationships or traceability.

Implementation of checks is supported by verification languages such as the object constraint language (OCL) [36] and scripting languages that are integrated in MBSE tools. Some tools also allow to specify the severity (information, warning, error, etc.) of the checks.

Safety case progress tracking The third way to support the safety case assessment is to provide statistics that indicate the status of the safety case progression. An exemplary list of values to collect for the safety case statistics is provided below.

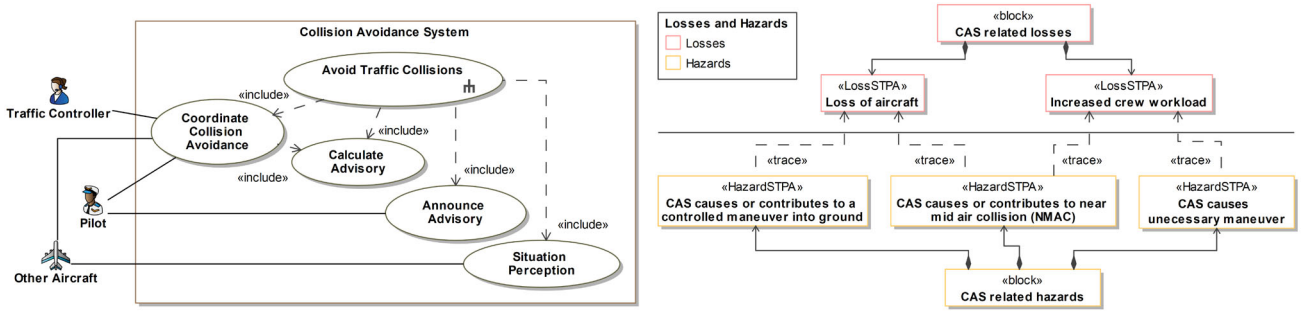
- ◇ Number of goals
- ◇ Number of contexts
- ◇ Number of strategies
- ◇ Number of assumptions
- ◇ Number of justified assumptions
- ◇ Number of justifications
- ◇ Number of executed justifications

By tracking the count of the safety case elements over time, it is possible to determine how the safety case evolves throughout the project. At the same time, the state (*Justified, Executed*) of the *AssumptionGSN* and *JustificationGSN* elements (see Fig. 4) provides information on the safety case progress. For instance, the number of justified assumptions and executed justifications is helpful to obtain an idea of the necessary effort.

To implement the elaborated functionality, the scripting ability of MBSE tools can be utilized. For the proof-of-concept implementation of this article, a function was implemented that queries the model for each element type of the safety case and returns the number per element type.

5 Framework demonstration

After elaborating the concept in the previous section, it is demonstrated in the following. Since the focus is on the functionality to generate and assess the safety case, an in-depth explanation of the modeling and safety analysis activities is omitted. Instead, a brief introduction of a system example



(a) Use case of *CAS* (b) Losses and hazards of *CAS*

Fig. 7 *CAS* use case (left) and potential losses and hazards of *CAS* (right)

#	Name	Text	Traced From	Traced To
1	115 Collision Avoidance System (CAS) Requirements	Below this requirement, all requirements for the CAS will be collected.	Avoid Traffic Collisions	Collision Avoidance System
2	115.1 CAS Functional Requirements	Below this requirement, all functional for the CAS will be collected.	Avoid Traffic Collisions	Collision Avoidance System
3	115.1.1 Perception of Own Aircraft State	The CAS shall be able to provide a perception of the own aircraft state (position, orientation, speed, etc.).	Situation Perception	OwnAircraftState
4	115.1.3 Perception of Other Aircraft State	The CAS shall be able to provide a perception of other aircraft states (position, orientation, speed, etc.) in an area of 10 km.	Situation Perception	OtherAircraftState
5	115.1.2 Traffic Advisory Calculation	The CAS shall be able to calculate a traffic advisory. The calculated advisory provides specific actions to the pilot that would prevent a near mid air collision.	Calculate Advisory	Advisory
6	115.1.4 Traffic Advisory Announcement	The CAS shall be able to announce the traffic advisory to the pilot.	Announce Advisory	Advisory
7	115.1.5 Collision Avoidance Coordination	The CAS shall be able to coordinate the collision avoidance maneuver with the other aircraft.	Coordinate Collision Avoidance	ADS-B

Fig. 8 *CAS* system requirements with traceability to model elements

and the safety analysis results are provided to establish the context. It is important to note that the example is simplified and merely serves the purpose of elaborating the approach.

5.1 System design

The considered system is a *collision avoidance system (CAS)* envisioned to support the *Pilot*. In Fig. 7a, an overview of its main use case is displayed. The main purpose of the *CAS* is to avoid traffic collisions. Therefore, it has to be able to calculate and announce advisories to the *Pilot* of the respective aircraft. As depicted in the design perspective of Fig. 3, the use cases and corresponding stakeholder requirements are used to derive the system requirements. Some of the system requirements are displayed in Fig. 8. The system requirements are traced from the use cases and traced to the elements of the system model.

In addition to investigating use cases, systematic design frameworks analyze the system context to identify relationships with other systems. Accordingly, not only the *CAS* itself is modeled in Fig. 9, but also the related systems and context participants. The *CAS* can issue an *Advisory* to inform the *Pilot* about dangerous traffic situations. Moreover, it depicts how the *CAS* receives feedback in form of the *OwnAircraftState* and *OtherAircraftState* from the *Sensor Unit*. This situation perception is used to identify if an *Advisory* is necessary. Considering the *Advisory*, the *Pilot* is able to adjust the control of the *Aircraft* through the *ManeuverCMD* to the *Aircraft Control System*.

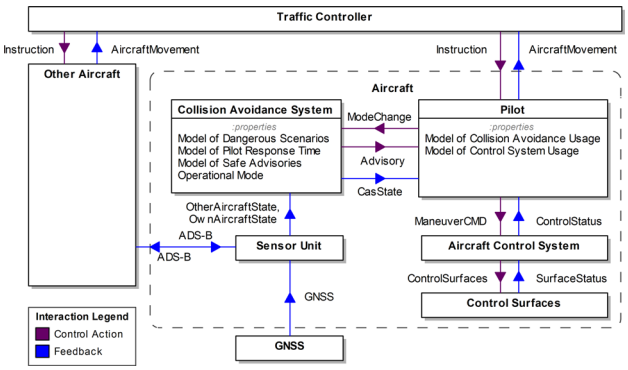


Fig. 9 Control structure of *CAS*

5.2 Safety analysis

As described in Sect. 4.1.2, the safety analysis should be executed in parallel to the design activities. Therefore, the model-based system design artifacts can be traced to the safety analysis as highlighted in Fig. 3. For instance, the stakeholder needs can be helpful in identifying losses and deriving hazards. Since the goal of the *CAS* is to prevent traffic collisions, a related loss would be a *Loss of Aircraft* due to a traffic accident. In relation to this loss, a hazard is that the *CAS causes or contributes to a controlled maneuver into ground*. Other examples are shown in Fig. 7b.

After defining losses and hazards for the first step of the STPA, the control structure has to be modeled in the second step. When following a systematic design framework, the

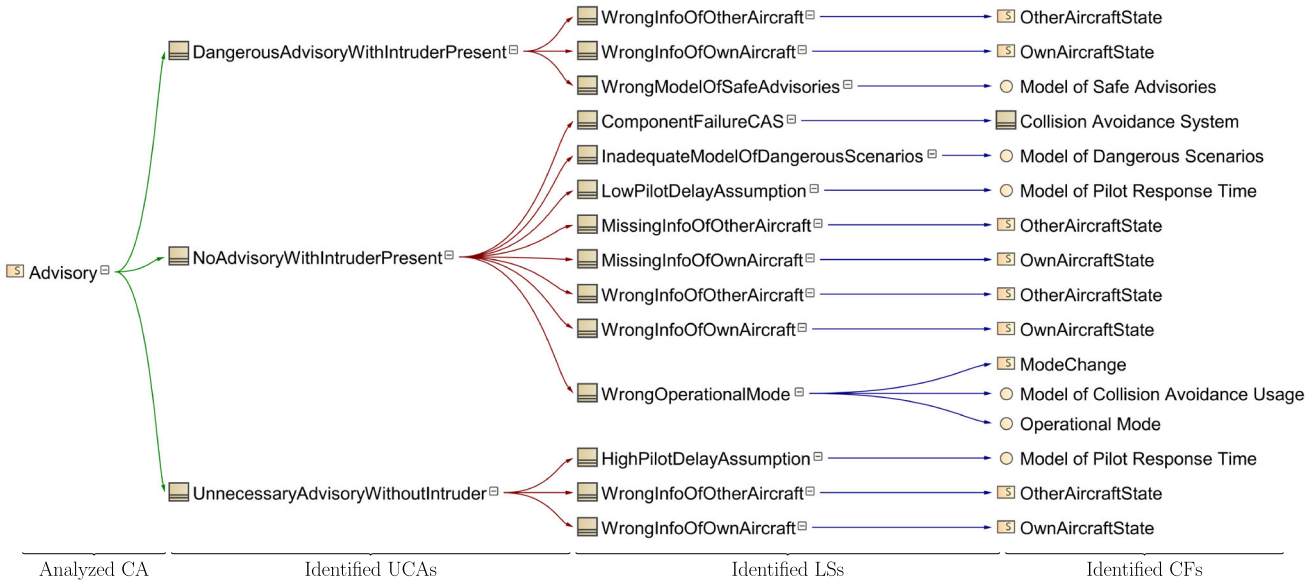


Fig. 10 Safety analysis elements and their relationships for the advisory CA

resulting system model already includes important information for the control structure. However, an important addition from the control structure perspective are the process models. Adding process models helps to identify more nuanced LSs later in the safety analysis. Consequently, some process models are added in form of properties to the *Pilot* and *CAS* in Fig. 9. However, in Fig. 9, only the title for the process models is shown. For instance, the *Model of Dangerous Situations* is a concise representation of all the beliefs that the *CAS* uses to classify a traffic situation as dangerous. Another control structure feature is the categorization of interactions into *feedback* and *control action*.

Using the control structure, the identification of UCAs and LSs can be executed in typical STPA fashion. An excerpt of the results for the *Advisory* CA is displayed in Fig. 10. It is important to note that only the names of the CA, UCA, LS, and CF elements are displayed in Fig. 10 for readability purposes. In the model, each UCA and LS element has a more detailed description, links to related elements, information about the analysis status, and information on the analyst [26].

In case of the *Advisory*, three ways of getting unsafe (UCAs) are identified and depicted at the first stage of the graph in Fig. 10. For instance, not providing an *Advisory* in the context of a collision course could lead to a minimum separation violation. Identified reasons in form of LSs are depicted in the next stage of the graph. The LS *LowPilotDelayAssumption* refers to the situation where the *Advisory* is not provided in time due to an inadequate belief of the *Pilot* response delay. The *Pilot* response delay is an important parameter during the algorithm design and influences when advisories are forwarded to the *Pilot*. A high *Pilot* delay assumption can result in too early advisories while a short pilot delay assumption can lead to the inability of the *Pilot* to react. This is one of the LSs highlighting the importance of documenting the beliefs in form of process models to identify potential issues later in the analysis. In this case, the related process model titled *Model of Pilot Response Time* is linked as the CF.

Another LS leading to no *Advisory* is the selection of a *WrongOperationalMode*. In this LS, the *Pilot* selected a wrong mode of the *CAS* because he had an incorrect belief of how the system works and should be used. This is represented by the presence of multiple CFs: *ModeChange*, *Model of Collision Avoidance Usage*, and *Operational Mode*. This is only one of many examples that highlight how both the system specification and human machine teaming affect the system's safety. Identifying issues related to these aspects is one of the strengths of the STPA.

5.3 Safety assurance

As elaborated in Sect. 4.1.3, the confidence in the safety analysis results can be improved through the assurance perspective. Since establishing and analyzing the assurance perspective manually for a large safety case is time consuming and error-prone, Sect. 4.2 detailed supporting approaches. First, a safety case baseline can be automatically generated using the traceability of the model-based safety analysis. Supplementary, assistance can be provided in various facets to improve the assessment and extension of the safety case. Both, the generation and assessment are demonstrated in the following.

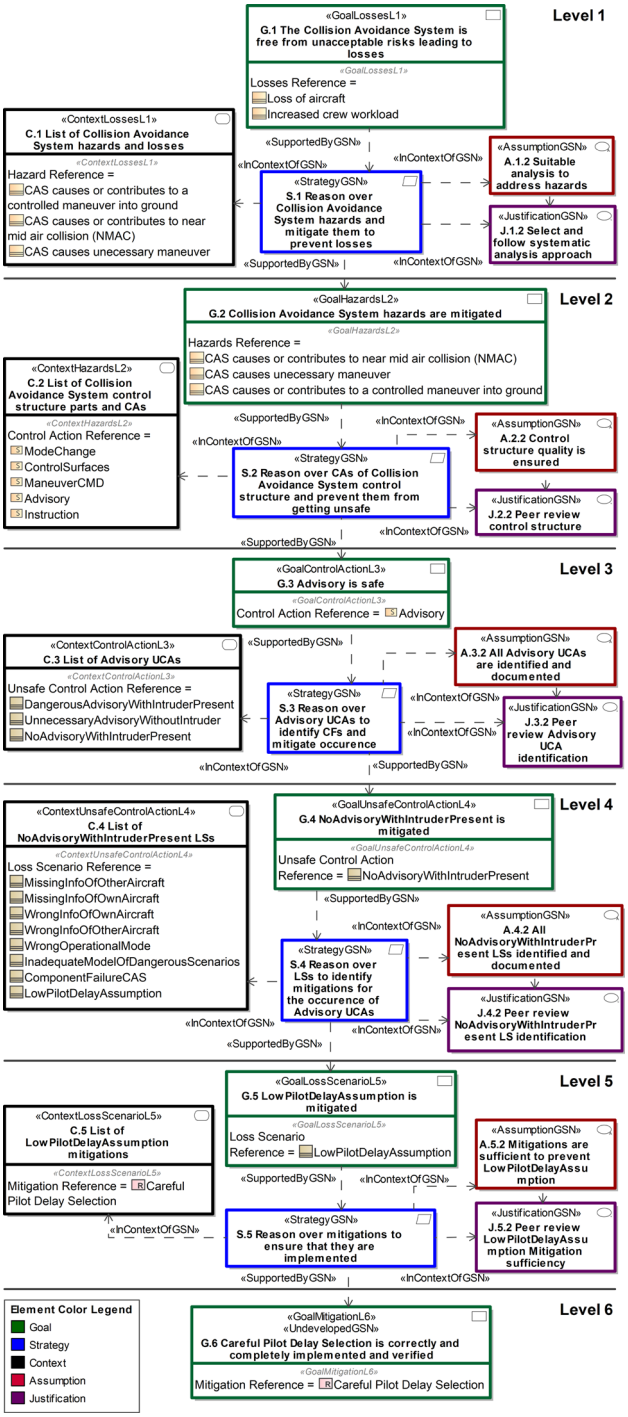


Fig. 11 Branch of generated safety case

5.3.1 Safety case generation

Using the results of the model-based STPA, a safety case can be generated according to the safety case pattern and functionality presented in Sect. 4.2.2 and Sect. 4.2.3. To give an idea of how a result looks like, a branch of the generated safety case for the *CAS* is depicted in Fig. 11. It is important to note that Fig. 11 only represents one branch of the safety case. To limit the scope, the focus is set on a branch of the *Advisory*. For this branch, all safety case levels are displayed to visualize how the pattern of Fig. 5 was implemented. In contrast to the pattern presented in Fig. 5, the context elements and references to the relevant STPA elements are included. However, the assumptions and justifications related to the goals are omitted for readability.

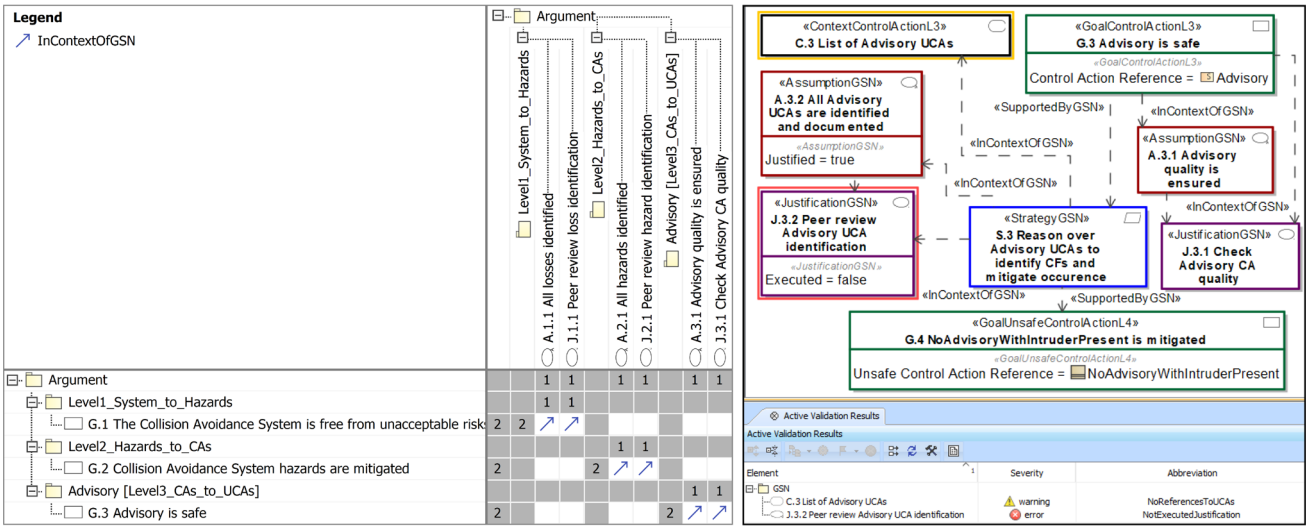
In Fig. 11, it is shown how the safety case gets more specific with each safety case level. The overarching goal *G.1* is to prevent unacceptable risks leading to losses. Therefore, the strategy *S.1* is to analyze and mitigate the hazards to prevent corresponding losses. A potential reason for hazards is the inadequate *Advisory*. Hence, unsafe *Advisories* are analyzed and mitigated. When comparing Fig. 10 and Fig. 11, it is visible how the safety case represents the analysis results and augments them with an additional perspective. Complementary to the STPA results in form of CAs, UCAs, and Ms, the safety case scrutinizes their validity and completeness. This is done through the addition of assumptions and justifications that have to be processed to establish a valid safety case. For instance, if some unsafe executions (UCAs) of the *Advisory* are not identified (A.3.2), the strategy (S.3) will fail to prevent them. Hence, the justification (J.3.2) demands a peer review of the UCA identification to reduce the likelihood of missing UCAs. To fulfill this justification, activities such as the coverage analysis presented in [26] can be executed. The process of adding, analyzing, and justifying assumptions is repeated on each safety case level.

5.3.2 Safety case assessment

If the assurance perspective is established in a model-based fashion, further assistance functionality becomes available, as elaborated in Sect. 4.2.4.

Safety case visualization and V&V One way to assist in handling large-scale safety cases is a multi-faceted visualization. In Sect. 4.2.4, it is explained how each view can have its own advantages and can therefore be leveraged to analyze a different aspect. Hence, the demonstration utilizes different visualizations to discuss aspects of the safety case in the following.

Since the safety case branch in Fig. 11 did not display the assumptions and justifications related to the goals (see Fig. 5), the matrix view in Fig. 12a highlights these relations. In the matrix, the goals of the first three safety case levels are displayed with their relations to assumptions and justifications. For instance, to address the hazards, as stated in *S.1*, it is assumed that an analysis with an appropriate quality is executed (*A.1.2*). Hence, the justification (*J.1.2*) is to follow and document a proven analysis approach. In this example, the proven analysis approach is the systematic STPA execution



(a) Goals to assumptions/justifications (L1-L3) (b) Validation examples (L3-L4)

Fig. 12 Matrix (left) and diagram (right) view highlighting parts of the safety case

#	Goal	Strategies	Contexts	Assumptions	Justifications	Supporting Goals
1	G.4 DangerousAdvisoryWithIntruderPresent is mitigated	S.4 Reason over LSs to identify mitigations for the occurence of Advisory UCAs	C.4 List of DangerousAdvisoryWithIntruderPresent LSs	A.4.1 DangerousAdvisoryWithIntruderPresent UCA quality is ensured A.4.2 All DangerousAdvisoryWithIntruderPresent LSs identified and documented	J.4.1 Check DangerousAdvisoryWithIntruderPresent UCA quality J.4.2 Peer review DangerousAdvisoryWithIntruderPresent LS identification	G.5 WrongInfoOfOtherAircraft is mitigated G.5 WrongInfoOfOwnAircraft is mitigated G.5 WrongModelOfSafeAdvisories is mitigated
2	G.4 NoAdvisoryWithIntruderPresent is mitigated	S.4 Reason over LSs to identify mitigations for the occurence of Advisory UCAs	C.4 List of NoAdvisoryWithIntruderPresent LSs	A.4.1 NoAdvisoryWithIntruderPresent UCA quality is ensured A.4.2 All NoAdvisoryWithIntruderPresent LSs identified and documented	J.4.1 Check NoAdvisoryWithIntruderPresent UCA quality J.4.2 Peer review NoAdvisoryWithIntruderPresent LS identification	G.5 MissingInfoOfOtherAircraft is mitigated G.5 MissingInfoOfOwnAircraft is mitigated G.5 WrongInfoOfOwnAircraft is mitigated G.5 WrongInfoOfOtherAircraft is mitigated G.5 WrongOperationalMode is mitigated G.5 InadequateModelOfDangerousScenarios is mitigated G.5 ComponentFailureCAS is mitigated G.5 LowPilotDelayAssumption is mitigated
3	G.4 UnnecessaryAdvisoryWithoutIntruder is mitigated	S.4 Reason over LSs to identify mitigations for the occurence of Advisory UCAs	C.4 List of UnnecessaryAdvisoryWithoutIntruder LSs	A.4.1 UnnecessaryAdvisoryWithoutIntruder UCA quality is ensured A.4.2 All UnnecessaryAdvisoryWithoutIntruder LSs identified and documented	J.4.1 Check UnnecessaryAdvisoryWithoutIntruder UCA quality J.4.2 Peer review UnnecessaryAdvisoryWithoutIntruder LS identification	G.5 WrongInfoOfOtherAircraft is mitigated G.5 WrongInfoOfOwnAircraft is mitigated G.5 HighPilotDelayAssumption is mitigated

Fig. 13 Table view of the safety case (L4 to L5)

according to the handbook [8]. However, it is also possible to extend the justification and demand other safety analysis approaches in addition to the STPA. The matrix view can also be used to display and quickly analyze other aspects. Examples are the relations between assumptions and justifications or the relations between strategies and supporting goals.

The subsequent diagram view in Fig. 12b is not new but extends safety case L3 of the branch shown in Fig. 11. Similar to the matrix view, it displays the assumption and justification related to *G.3*. In addition, the properties of the assumption *A.3.2* and justification *J.3.2* are shown. It is visible how the justification *J.3.2* was evaluated to be good enough (*Justified = true*) to justify *A.3.2*. At the same time, *J.3.2* was not yet executed (*Executed = false*) and therefore invalidates this part of the safety case. Accordingly, the implemented validation rule highlights *J.3.2* with an error. To demonstrate another validation rule defined in Sect. 4.2.4, the UCA entries in the context element *C.3* were removed. Hence, a second validation warning highlights the empty context element. Empty context elements are generated when no related elements are identified in the safety analysis. In case of empty UCA or LS context elements, this may indicate that something was missed or not yet executed. In general, the diagram view allows to display multiple relevant aspects at once (elements, properties, and relations) but demands space to achieve this. Hence, the diagram view's strength is in the analysis of local parts of the safety case. However, its usability reduces when trying to analyze many elements at once.

After analyzing the safety case L3 with the diagram view, a table view can be used to get an overview over the safety case L4. Since safety case L4 covers the identified UCAs displayed in Fig. 10, three main goals related to the *Advisory* are presented in Fig. 13. Each goal covers one of the UCAs identified during the analysis as visible in the summary in Fig. 10. In addition, the related strategies, contexts, assumptions, justifications, and supporting goals are displayed in the

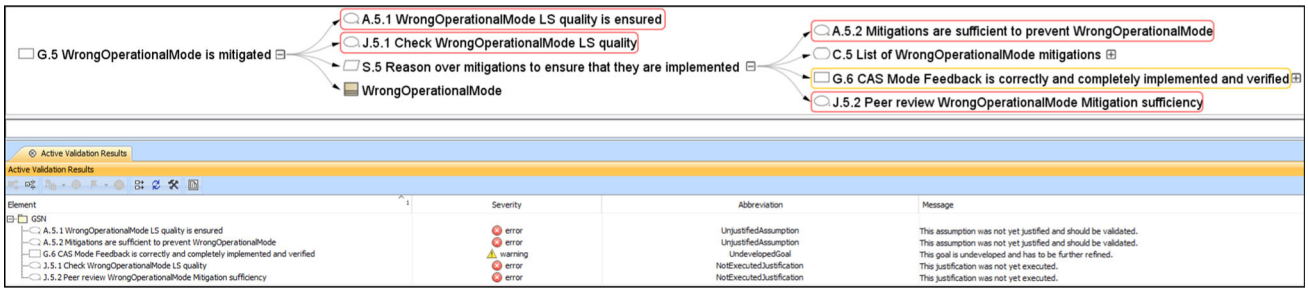


Fig. 14 Relationship map view of the safety case (L5 to L6)

table. With a high ratio of information over space, table views give a broad overview and can be used to quickly navigate the model.

For displaying the relationship between the safety case L5 and L6, a relationship view is used in Fig. 14. In this branch of the safety case, the goal (*G.5*) is to mitigate the *WrongOperationalMode* LS. Therefore, the Ms shall be analyzed to ensure that they are addressed (*S.5*). One M is the *CAS mode feedback*, which has to be implemented and verified (*G.6*). Similar to Fig. 12b, the validation is shown. The validation highlights that the goal (*G.6*) is currently classified as undeveloped. At the same time, the assumption (*A.5.2*) was not evaluated to be justified and the justification (*J.5.2*) was not executed. Accordingly, it is shown how the validation helps to highlight parts of the safety case that need further refinement.

After browsing through the safety case levels with different views, a final table view demonstrates that it is also possible to analyze and set specific safety case properties. In Fig. 15, the properties of the assumptions (Fig. 15a) and justifications (Fig. 15b) are displayed from safety case L1 till L4 for the branches of the *Advisory*. For each assumption in Fig. 15a, a corresponding justification (same #) in Fig. 15b is provided. When setting $Justified = true$ as displayed for the assumptions in Fig. 15a, it is acknowledged that the justification is sufficient. Similarly, the $Executed$ property of the justifications shows the justification status. Accordingly, in the justification table Fig. 15b, it is indicated that it was not yet reviewed if all LSs of the *NoAdvisoryWithIntruderPresent* UCA are identified (Fig. 15b #11) limiting the validity of this part of the safety case.

Safety case progress tracking The last demonstrated functionality for assessing the safety case is the creation and analysis of safety case statistics. In Fig. 16a, the number of elements for the whole safety case is tracked over time to give an idea of a possible progression of a safety case. Initially, there are no safety case elements (Fig. 16a #1). After executing the generation functionality described in Sect. 4.2.3, a safety case baseline is created for the current status of the safety analysis (Fig. 16a #2). However, the validity of the safety case and completeness of the safety analysis is not yet assessed. To improve the validity of the safety case, it is checked if the assumptions are justified, resulting in an increase in the number of justified assumptions (Fig. 16a #3). It is important to note that the number of 12 justified assumptions corresponds to the 12 justified assumptions shown in Fig. 15a, highlighting that both views are representations of the same model. After making sure that the justifications are adequate for the assumptions, the execution of justifications can be started. This results in an increase in executed justifications (Fig. 16a #4). Again, the number of 8 executed justifications corresponds to the number of executed justifications shown in Fig. 15b.

In addition to the overarching analysis of the safety case statistics, it is helpful to take a look at the statistics per safety case level as depicted in Fig. 16b. Thereby, the progress throughout all safety case levels can be observed and evaluated. In the example, it is visible the number of goals at safety case L4 (Fig. 16b #4) is substantially lower than the goals at safety case L3 (Fig. 16b #3). This indicates that not all CAs were analyzed yet. Simultaneously, zero justified assumptions and executed justifications at safety case L5 (Fig. 16b #5) as well as the lack of elements in safety case L6 (Fig. 16b #6) point to an incomplete analysis. Since the example is only created for demonstration and validation purposes, this accurately represents the incomplete state of the safety case.

In practice, the safety case progress will not be as linear, but instead include parallel advances. At the same time, the number of safety case elements will increase or decrease depending on the resulting adjustments on the system model. In the end, all assumptions should be justified and all justifications should be executed to form a convincing safety case. By validating assumptions and executing justifications, additional issues in the system design can be identified and addressed. Ultimately, this leads to an increasing confidence in the final system design.

#	Name	Justified
1	A.1.1 All losses identified	true
2	A.1.2 Suitable analysis to address hazards	true
3	A.2.1 All hazards identified	true
4	A.2.2 Control structure quality is ensured	true
5	A.3.1 Advisory quality is ensured	true
6	A.3.2 All Advisory UCAs are identified and documented	true
7	A.4.1 DangerousAdvisoryWithIntruderPresent UCA quality is ensured	true
8	A.4.1 NoAdvisoryWithIntruderPresent UCA quality is ensured	true
9	A.4.1 UnnecessaryAdvisoryWithoutIntruder UCA quality is ensured	true
10	A.4.2 All DangerousAdvisoryWithIntruderPresent LSs identified and documented	true
11	A.4.2 All NoAdvisoryWithIntruderPresent LSs identified and documented	true
12	A.4.2 All UnnecessaryAdvisoryWithoutIntruder LSs identified and documented	true

(a) State of assumptions

#	Name	Executed
1	J.1.1 Peer review loss identification	true
2	J.1.2 Select and follow systematic analysis approach	true
3	J.2.1 Peer review hazard identification	true
4	J.2.2 Peer review control structure	true
5	J.3.1 Check Advisory CA quality	true
6	J.3.2 Peer review Advisory UCA identification	false
7	J.4.1 Check DangerousAdvisoryWithIntruderPresent UCA quality	true
8	J.4.1 Check NoAdvisoryWithIntruderPresent UCA quality	true
9	J.4.1 Check UnnecessaryAdvisoryWithoutIntruder UCA quality	true
10	J.4.2 Peer review DangerousAdvisoryWithIntruderPresent LS identification	false
11	J.4.2 Peer review NoAdvisoryWithIntruderPresent LS identification	false
12	J.4.2 Peer review UnnecessaryAdvisoryWithoutIntruder LS identification	false

(b) State of justifications

Fig. 15 Sate of safety case assumptions (left) and justifications (right)

#	Date	Scope	Number of Goals	Number of Contexts	Number of Assumptions	Number of Justifications	Number of Justified Assumptions	Number of Executed Justifications
1	2023.07.26 12.03	Argument	0	0	0	0	0	0
2	2023.08.15 09.05	Argument	35	27	54	54	0	0
3	2023.08.15 09.06	Argument	35	27	54	54	12	0
4	2023.08.15 09.07	Argument	35	27	54	54	12	8

(a) Number of safety case elements tracked over time

#	Date	Scope	Number of Goals	Number of Contexts	Number of Assumptions	Number of Justifications	Number of Justified Assumptions	Number of Executed Justifications
1	2023.08.15 09.08	Level1_System_to_Hazards	1	1	2	2	2	2
2	2023.08.15 09.08	Level2_Hazards_to_CAs	1	1	2	2	1	2
3	2023.08.15 09.08	Level3_CAs_to_UCAs	12	12	24	24	3	1
4	2023.08.15 09.08	Level4_UCAs_to_LSs	3	3	6	6	6	3
5	2023.08.15 09.08	Level5_LSs_to_Mitigations	10	10	20	20	0	0
6	2023.08.15 09.08	Level6_Mitigations_to_Assurance	8	0	0	0	0	0

(b) Number of safety case elements per safety case level at one point in time

Fig. 16 Statistics highlighting the progress of the safety case

6 Discussion

This article proposes an integration of promising approaches for system design (MBSE), safety analysis (STPA), and safety assurance (GSN). The model-based nature of the integration enables assisting automation such as the generation and assessment of a safety case. However, even with automation, an advantage cannot be claimed without consideration of the usability of the generated artifacts. Specifically for complex systems, working with a generated safety case can be difficult. This is why this article also highlights how the model-based nature of the safety case can be utilized in various ways to improve the usability. Related aspects include the demonstration of a multi-faceted visualization, the automated highlighting of safety case parts that require further attention, and the tracking of the safety case progress. The model-based integration in the framework allows even more possibilities than demonstrated in this article. For instance, the end-to-end traceability of the model-based approaches enables a change impact analysis from the design till the assurance. If the design is changed, a functionality can automatically detect that the safety analysis needs to be adapted [27]. After adapting the safety analysis, a second functionality can identify and indicate that a corresponding update is also needed in the assurance perspective. Implementing such a multi-layered change impact functionality is only one of many examples to improve the usability even further.

Because the long-term goal would be to use the approach in the developments of safety-critical systems, caution with the following aspects is required. Although generating a safety case automatically around the safety analysis is systematic, it does not guarantee that the underlying argument is correct or sufficient. Accordingly, the validity of the safety case depends on the extent to which each part of the safety case is developed and refined. As discussed earlier, this is why it is very important to systematically identify and evaluate all assumptions and limitations at each safety case level. Therefore, validation frameworks such as [37] are valuable for establishing a consistent safety analysis quality and should extend the current list of assumptions and justifications of Fig. 5 in future work.

At the same time, the assurance framework in this article does not yet cover the relation to other safety analysis approaches. The reason is that the relations between STPA and other analysis methods are currently being defined in guidelines such as [22]. For reference, two examples for integration will be mentioned. First, it is possible to expand *J.1.2 Follow systematic analysis approach* of Fig. 5 and also demand the analysis with methods in addition to the STPA. Second, it is possible to use other safety analysis techniques

to refine and prevent issues that were identified with the STPA. For instance, if a component failure is identified as a reason for a LS, an M might request the execution of a Failure Modes and Effects Analysis (FMEA) or Fault Tree Analysis (FTA) for the corresponding components. Similarly, Ms can also include other valuable analysis approaches that help to tackle the challenges of software-intensive systems. Examples include scenario-based analysis approaches [38] or formal methods [39].

Another topic of discussion is the demonstrated implementation. It should be mentioned that the framework can also be implemented using other tools and languages as long as the required capabilities are present. A tool-related disadvantage of the demonstrated implementation is that it rests on a tool-specific API and cannot be easily transferred to other tools. However, there are attempts to standardize the API for SysMLv2, which might facilitate a transferable implementation in the future [19].

If an application for safety-critical systems is envisioned, a tool qualification might be required to verify and validate the automation. When using automation, caution is also needed. For instance, automation may introduce a false sense of confidence into the safety case generation and evaluation. If specific functionality is provided to check for properties of the safety case, it must be ensured that these checks are sufficient and not tempered with. This is particularly important when the safety case increases in complexity due to the subsequent reliance on automation.

7 Conclusion

Adapted system design, safety analysis, and safety assurance approaches are necessary to overcome the challenges of developing complex systems. Accordingly, this article proposes an end-to-end integration of recent approaches for system design (MBSE), safety analysis (STPA), and safety assurance (GSN). By connecting the approaches in a model-based fashion, assisting automation is enabled. In this article, the automation is used to increase the confidence in the system safety. Therefore, it is demonstrated how a safety case baseline can be automatically generated and assessed. The ability to generate a safety case baseline and assess its properties has shown promising potential to support the development of complex systems. Automated checks help to highlight safety case parts that need more attention while statistics provide a quick overview of the safety case status and progress.

In future work, it should be further evaluated how the system design, safety analysis, and safety assurance perspectives can be kept in a consistent state through the already established traceability in the model. A related functionality is a change impact implementation that automatically detects inconsistencies and updates the respective views. Finally, integrating the framework with domain-specific standards and methods should be further refined and evaluated.

Acknowledgements This work has received funding by the V&V4NGC project of the German Aerospace Center (DLR). In addition, this work has received funding by the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 957210-XANDAR.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Leveson, N.G., Thomas, J.P.: Certification of safety-critical systems. Commun. ACM **66**(10), 22–26 (2023). https://doi.org/10.1145/3615860
2. Frazza, C., Darfeuil, P., Gauthier, J.: MBSA in aeronautics: a way to support safety activities. In: Seguin, C., Zeller, M., Prosvirnova, T. (eds.) Model-Based Safety and Assessment, pp. 31–42. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15842-1_3
3. Fogarty, D., De Salvo, P., Edward, D.: Model-based systems engineering and model-based safety analysis: final report. Federal Aviation Administration, Tech. Rep. DOT/FAA/TC-20/42 (2021)
4. Krishnan, R., Bhada, S.V.: An integrated system design and safety framework for model-based safety analysis. IEEE Access **8**, 146483–146497 (2020). https://doi.org/10.1109/ACCESS.2020.3015151
5. Denney, E., Pai, G., Pohl, J.: AdvoCATE: an assurance case automation toolset. In: Ortmeier, F., Daniel, P. (eds.) Computer Safety, Reliability, and Security, pp. 8–21. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-33675-1_2
6. Walden, D.D., Roedler, G.J., Forsberg, K.J., Hamelin, R.D., Shortwell, T.M.: Systems Engineering Handbook, A Guide for System Life Cycle Processes and Activities. WILEY, San Diego (2015)
7. International Council on Systems Engineering (INCOSE). Systems engineering vision 2020. INCOSE, Tech. Rep. INCOSE-TP-2004-004-02 (2007)
8. Leveson, N. G., Thomas, J. P.: *STPA Handbook* (2018)
9. Leveson, N.G.: Engineering a Safer World, Systems Thinking Applied to Safety. The MIT Press, Cambridge (2016)
10. The Assurance Case Working Group (ACWG). *Goal Structuring Notation Community Standard Version 3* (2021)
11. de Souza, F.G.R., de Melo Bezerra, J., Hirata, C.M., de Saqui-Sannes, P., Apvrille, L.: Combining STPA with SysML modeling. In: 2020 IEEE International Systems Conference (SysCon) (2020). pp. 1–8. https://doi.org/10.1109/SysCon47679.2020.9275867

12. Acar Celik, E., Cârlan, C., Abdulkhaleq, A., Bauer, F., Schels, M., Putzer, H.J.: Application of STPA for the elicitation of safety requirements for a machine learning-based perception component in automotive. In: Trapp, M., Saglietti, F., Spisländer, M., Bitsch, F. (eds.) Computer Safety, Reliability, and Security, pp. 319–332. Springer, Cham (2022)
13. Hirata, C., Nadjm-Tehrani, S.: Combining GSN and STPA for safety arguments. In: Romanovsky, A., Troubitsyna, E., Gashi, I., Schoitsch, E., Bitsch, F. (eds.) Computer Safety, Reliability, and Security, pp. 5–15. Springer, Cham (2019)
14. Object Management Group (OMG). Risk Analysis and Assessment Modeling Language (2021)
15. Estefan, J.A.: Survey of model-based systems engineering (MBSE) methodologies. Incose MBSE Focus Group **25**(8), 1–12 (2007)
16. Aleksandraviciene, A., Morkevicius, A.: MagicGrid® Book of Knowledge. A Practical Guide to System Modeling using MagicGrid from No Magic, 2nd. Vitae Litera, Kaunas (2021)
17. Roques, P.: MBSE with the ARCADIA method and the Capella tool. In: 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016). Toulouse, France (2016)
18. Friedenthal, S., Moore, A., Steiner, R.: A Practical Guide to SysML, The Systems Modeling Language. Morgan Kaufmann, Amsterdam (2015)
19. Object Management Group (OMG). In: Systems Modeling Language (SysML®) v2 API and services Request For Proposal (RFP) (2018)
20. Rosenow, H.: Trade off bewertungsmethodik für tool- und methodenentscheidungen zur virtualisierung und modellbasierung in der entwicklung. Master's Thesis, Technische Universität München, München (2018)
21. International Organization for Standardization (ISO). ISO 21448:2022 road vehicles—safety of the intended functionality (2022)
22. SAE. Using STPA During Development and Safety Assessment of Civil Aircraft AIR6913, 2018-02-13. Work in Progress
23. SAE. J3187 _202305: System theoretic process analysis (STPA) recommended practices for evaluations of safety-critical systems in any industry (2023)
24. SAE. J3307: System theoretic process analysis (STPA) standard for all industries, WIP
25. Ahlbrecht, A., Durak, U.: Integrating safety into MBSE processes with formal methods. In: 2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC) (2021), pp. 1–9. https://doi.org/10.1109/DASC52595.2021.9594315
26. Ahlbrecht, A., Durak, U.: Model-based STPA: enabling safety analysis coverage assessment with formalization. In: 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), pp. 1–10 (2022). https://doi.org/10.1109/DASC55683.2022.9925883
27. Ahlbrecht, A., Zaeske, W., Durak, U.: Model-based STPA: towards agile safety-guided design with formalization. In: 2022 IEEE International Symposium on Systems Engineering (ISSE), pp. 1–8 (2022). https://doi.org/10.1109/ISSE54508.2022.10005396
28. Rushby, J.M., Xu, X., Rangarajan, M., Weaver, T.L.: Understanding and evaluating assurance cases. NASA, Tech. Rep. CR–2015-218802 (2015)
29. Wei, R., Kelly, T.P., Dai, X., Zhao, S., Hawkins, R.: Model based system assurance using the structured assurance case metamodel. J. Syst. Softw. **154**, 211–233 (2019). https://doi.org/10.1016/j.jss.2019.05.013
30. Maksimov, M., Fung, N.L.S., Kokaly, S., Chechik, M.: Two decades of assurance case tools: a survey. In: Gallina, B., Skavhaug, A., Schoitsch, E., Bitsch, F. (eds.) Computer Safety, Reliability, and Security, pp. 49–59. Springer, Cham (2018)
31. Leveson, N.G.: White paper on limitations of safety assurance and goal structuring notation (GSN) (2020)
32. de la Vara, J.L., Garcıa, A.S., Valero, J., Ayora, C.: Model-based assurance evidence management for safety-critical systems. Softw. Syst. Model. **21**(6), 2329–2365 (2022). https://doi.org/10.1007/s10270-021-00957-z
33. Wolny, S., Mazak, A., Carpella, C., Geist, V., Wimmer, M.: Thirteen years of SysML: a systematic mapping study. Softw. Syst. Model. (2020). https://doi.org/10.1007/s10270-019-00735-y
34. Ahlbrecht, A., Bertram, O.: Evaluating system architecture safety in early phases of development with MBSE and STPA. In: 2021 IEEE International Symposium on Systems Engineering (ISSE), pp. 1–8 (2021). https://doi.org/10.1109/ISSE51541.2021.9582542
35. Sun, L.: Establishing confidence in safety assessment evidence. PhD thesis, University of York (2012)
36. Cabot, J., Gogolla, M.: Object constraint language (OCL): a definitive guide. In: Bernardo, M., Cortellessa, V., Pierantonio, A. (eds.) Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18–23, 2012. Advanced Lectures, pp. 58–90. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-30982-3_3
37. Sadeghi, R., Goerlandt, F.: A proposed validation framework for the system theoretic process analysis (STPA) technique. Saf. Sci. **162**, 106080 (2023). https://doi.org/10.1016/j.ssci.2023.106080
38. Khastgir, S., Brewerton, S., Thomas, J., Jennings, P.: Systems approach to creating test scenarios for automated driving systems. Reliab. Eng. Syst. Saf. **215**, 107610 (2021). https://doi.org/10.1016/j.ress.2021.107610
39. Julian, K.D., Kochenderfer, M.J.: Guaranteeing safety for neural network-based aircraft collision avoidance systems. In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC), pp. 1–10 (2019) https://doi.org/10.1109/DASC43569.2019.9081748

Alexander Ahlbrecht works as a scientific associate in the Institute of Flight Systems at the German Aerospace Center (DLR). He received his master's degree in Electronic Automotive and Aerospace Systems at the TU-Braunschweig and is currently a PhD candidate at the TU-Clausthal. His research focuses on examining the potential synergy of model-based systems engineering (MBSE) and model-based safety assessment (MBSA) for the development of avionics systems. At DLR, he manages and contributes to multiple research projects.

Jasper Sprockhoff works as a scientific associate in the Institute of Flight Systems at the German Aerospace Center (DLR). He received his master's degree in Informatics at the TU-Clausthal. His current research focus is on the safe integration of machine learning (ML) algorithms in aviation systems. He is responsible for the project management and technical content of the MBSE4AI project at DLR while contributing to multiple other research projects.

Umut Durak is the Group Leader for Avionics Systems in the Institute of Flight Systems at the German Aerospace Center (DLR). He is also a Professor for Aeronautical Informatics in the Informatics Institute at the TU-Clausthal. His research interests concentrate on engineering of software intensive airborne systems. He has published 5 books and more than 80 papers in various conference proceedings and journals. He is an Associate Fellow and the Co-Chair of Software Technical Committee at the American Institute of Aeronautics and Astronautics (AIAA) and an Executive Board Member of the German simulation association Arbeitsgemeinschaft Simulation (ASIM).