A Textual Domain Specific Language for System-Theoretic Process Analysis

Jette Petzold

Master Thesis
March 25, 2022

Prof. Dr. Reinhard von Hanxleden
Real-Time and Embedded Systems Group
Department of Computer Science
Kiel University

Advised by
M. Sc. Lena Grimm

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Abstract

Risk analysis is an important part of system development in order to ensure safety. These analyses can help to reduce the risk of all kind of losses. For example a train should not be moving while the doors are open in order to prevent injury or loss of human life. Hazard or risk analysis techniques help to find such hazards that lead to accidents. One of these techniques is System-Theoretic Process Analysis (STPA), developed by Leveson. Usually, it is applied manually on paper, which is very time consuming and tedious. This problem is addressed by developing tools that systematize and partially automate the STPA process.

In this thesis the risk analysis topic is explored in more detail, including tools supporting STPA. Furthermore, this thesis proposes a Domain Specific Language (DSL) for STPA with an automatic visualization of the STPA components. The DSL is implemented as a Visual Studio Code (VS Code) Extension using Langium and Sprotty. An evaluation of the DSL revealed that the whole STPA process is supported. The DSL has potential to be a good alternative to the other tools, because of the visualization, which is not offered by most of the tools. Still, the DSL should be further improved by providing context tables introduced by Thomas. An evaluation with analysts using the DSL in real use-cases should be done in order to validate that the DSL is a good alternative.

Acknowledgements

First of all, I want to thank Prof. Dr. Reinhard von Hanxleden for the opportunity to write this thesis and for the constructive and helpful feedback during its conception. Moreover, I want to thank my advisor Lena Grimm for supervising my thesis and helping to organize the different topics. I also want to thank Niklas Rentz for his help with the VS Code Extension and Sprotty.

Thanks goes to the whole working group for the welcoming and friendly atmosphere and providing feedback to the DSL. Last but not least, I want to thank my family for supporting me through my course of studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Risk is a part of human existence and is part of our life [AR10]. When risks are ignored or misjudged people can die or get ill. Risk research started as soon as humans were able to reflect on the possibility of their own death and tried to avoid dangerous situations. Nowadays, risk research is done in many areas: medical, engineering, social, cultural, etc. In order to manage risks, several procedures must be done: risk identification, risk analysis, risk assessment, risk prioritization, and risk mitigation [SWH13]. However, these definitions are not generally accepted and hence these terms are also used to describe processes that include the other activities. The focus in this master thesis lies on the risk analysis, which has a long history in engineering, in the context of embedded systems.

Risk analysis is an important procedure during system development in order to reduce all kinds of losses. A loss can be for example the loss of a life or the loss of sensitive information and depends on the stakeholders. In order to prevent such losses, different hazard/risk analysis techniques exist, which can be applied during the development or afterwards. Wienen et al. [WBV+17] state that accident analysis methods are growing as a field of research. There exist some traditional approaches that were already used decades ago, but risk analysis of IT system require different approaches or adaptions of these techniques [SWH13]. Software itself can not be unsafe, only the entire system controlled by the software can be unsafe [Lev20]. That is because only physical entities can inflict damage and thus only physical processes, which may be controlled by software, can be unsafe. One relatively new method is the hazard analysis technique System-Theoretic Process Analysis (STPA), which also can be used for embedded systems. Usually, analysts execute STPA manually on paper.

The accident model of STPA is based on the System-Theoretic Accident Model and Processes (STAMP) introduced by Leveson [Lev04]. According to Leveson, STPA identifies more risks than traditional hazard analysis techniques such as Fault Tree Analysis (FTA) or Hazard and Operability Study (HAZOP) [Lev16]. A reason for that is that loss scenarios involving unsafe interactions among the system components are considered too and not only individual component failures.

Using STPA during development is often very time consuming when doing it manually without software support. Visual diagrams showing the relationships between identified elements of the aspects of STPA could be helpful, but is much work when a developer models them manually. There are already different tools and extensions for existing software that help to use STPA for a system. One of them is STPA based Hazard and Risk Analysis (SAHRA), which is an extension for Sparx Systems Enterprise Architect (EA) and based on Unified

Modeling Language (UML) [KRR16]. It allows to add several graphical elements, which represent different aspects of STPA such as losses and hazards.

1.1 Problem Statement

In order to simplify the usage of STPA and to minimize the time needed to successfully apply it, tools supporting STPA are helpful. The need for such tools that provide systematization, automation, and analysis completeness has been widely acknowledged [SPP+19]. Although there already exist tools supporting the usage of STPA, each of them has advantages as well as disadvantages. The goal of this thesis is to provide a DSL that combines the advantages and minimizes the disadvantages taking into account the evolution of other tools.

This thesis consists of two main parts: the exploration of the research field and the implementation of a DSL for STPA. These are completed by an evaluation stage. In the exploration part the topic of hazard analysis is explored in detail. This includes investigating the history of such techniques and examining alternatives to STPA. In order to determine the advantages and usefulness of STPA, real use-cases are observed, too. Furthermore, already existing tools supporting the application of STPA are checked out. In the implementation part the DSL is developed based on the results of the exploration. The evaluation outlines the benefits and differences to other approaches.

1.2 Outline

The next chapter introduces the foundations for the thesis such as the process of STPA and used technologies. In Chapter 3 already existing tools supporting STPA are presented and their (dis)advantages are outlined. Afterwards, Chapter 4 explores the risk analysis topic in more detail and hence provides an overview of alternatives to STPA and STPA related topics that can be further looked at in future work. Chapter 5 presents the conceptual ideas for the developed DSL and Chapter 6 introduces the implementation of it. Chapter 7 evaluates the DSL and finally Chapter 8 concludes with a summary and possible future work.

Chapter 2

Foundations

Developing a DSL for STPA requires to understand the STPA process at first. This chapter introduces basic concepts needed for the DSL and technologies used for the implementation. After outlining the STPA process in Section 2.1, Section 2.2 presents an improvement for STPA that is already in use: Context tables. Conclusively, Section 2.3 presents technologies needed for the actual implementation.

2.1 STPA Process

STPA is a relatively new hazard analysis technique based on System Theory, more precisely on STAMP [LT18]. In contrast to other techniques, it also considers accidents caused by unsafe interactions between components and not just component failures. The motivation in developing STPA was to include new causal factors that are not considered in other techniques [Lev16]. Additionally, in contrast to other methods such as FTA, STPA provides guidance to the users. A similar technique based on STAMP is Causal Analysis based on System Theory (CAST). The difference between CAST and STPA is that CAST is applied after an accident occurred in order to understand the causes, while STPA is used to develop a safe system. STPA can be used in any design phase but the most effective way to develop a safe system is to use it in the earliest phase, which is called safety-guided design. The results of STPA can be used in several ways, including: creation of requirements, identifying system recommendations, defining test cases and creating test plans, etc. [LT18]. Furthermore, the concept of emergence and the broad definition of a loss in STPA (explained later) enables STPA to be used for any system property not just safety. For example security properties can be checked.

STPA is an iterative process that does not need to be applied linearly [LT18]. Earlier results can be updated while the analysis progresses and more information becomes available. In the remaining section, the four steps of STPA are presented as explained in the STPA handbook [LT18]:

1. Define the purpose of the analysis
2. Model the Control Structure
3. Identify Unsafe Control Actions (UCAs)
4. Identify Loss Scenarios

2.1.1 Define the Purpose of the Analysis

In this step the goal of the analysis is determined. This includes, among others, whether safety, security, or another property should be ensured and the system that should be analyzed. In order to structure this step, it is divided into four parts:

Identify losses Depending on the industry, different notions are used for what should be prevented: Accident, mishap, adverse event, etc. In order to unify this, STPA introduces the definition of losses. A *loss* involves something that is of value to the stakeholders. For example this could be the loss of life or loss of reputation. In this phase the user can define any number of losses and rank or prioritize them.

Identify system-level hazards At first the system and its boundaries must be defined. A system is a set of components that work together in order to accomplish a common goal. It may contain subsystems or be part of a bigger system. Afterwards, the hazards can be defined. A *hazard* is a system state that will lead to a loss if certain worst-case environmental conditions are true. Each hazard should be traceable to one or more losses it causes if the worst-case conditions occur. The traceability is typically stated after the description. An example is given in the handbook [LT18]: "H-1: Aircraft violate minimum separation standards in flight [L-1, L-2, L-4, L-5]", whereas L-1 to L-5 point to already defined losses.

Identify system-level constraints For each hazard, a *system-level constraint* should be defined. It specifies the system conditions or behavior that are needed to prevent the hazard and hence the losses. They can be generated by inverting the condition of the hazard. For the hazard stated above the constraint is: "SC-1: Aircraft must satisfy minimum separation standards from other aircraft and objects [H-1]". As before, the conditions should contain tracing, this time to the hazards they prevent. This can be one or more hazards. Instead of preventing a hazard, the constraint can also state what should be done in case of a hazard without specifying a particular solution: "SC-3: If aircraft violate minimum separation, then the violation must be detected and measures taken to prevent collision [H-1]".

(optional) Refine hazards The hazards identified can be further refined into sub-hazards. For most applications this is not necessary but it can be helpful for larger and more complex applications. Each sub-hazard should be covered by a sub-constraint.

2.1.2 Model the Control Structure

After the goal of the analysis is defined, the analyst models the *control structure*. It is a system model composed of feedback control loops that should enforce constraints on the behavior of the system. The control structure should be a functional model, not a physical one. Control and authority in the system is indicated by the vertical axes: downward arrows represent control actions and upward actions represent feedback. A generic control loop can be seen Figure 2.1. The controller uses *control actions* to control and enforce constraints on the controlled process. Furthermore, it contains a *control algorithm* that determines what

control action should be send. The *process model* can contain the controller's belief about the environment, relevant aspects of the system, or the controlled process. It may be updated by the *feedback* the controlled process sends. Sensors and actuators are not needed yet, they will be added in a later step of STPA.

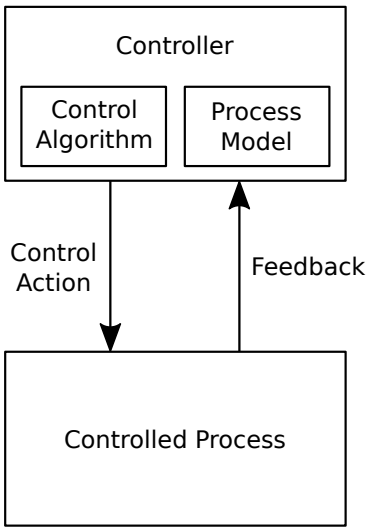


Figure 2.1. A generic control loop as shown in the STPA handbook [LT18].

This control loop can be used for software as well as for humans. For humans, the process model is normally called the *mental model* and the control algorithm may be called *operating procedures*. Besides the four elements already explained — controller, control action, feedback, and controlled process — a hierarchical control structure can also contain other inputs to and outputs from components that are neither control actions nor feedback. In order to handle more complex systems, abstraction is used. Instead of modeling every individual subsystem, they can be grouped together to a more abstract element of the control structure. Control actions and feedback paths can also profit from this principle. Several individual actions can be encapsulated by a broader action.

When modeling the control structure, responsibilities can be assigned to each entity of the structure. A *responsibility* states what an entity has to do in order to enforce the system-level constraints, possibly together with the other entities. Hence, when writing them down, they should reference the constraints they enforce. Additionally, the responsibilities can help to identify the control actions a controller needs.

The control structure does not need to be complete. When STPA is applied in an early development stage, some information might be missing. In such a case, the analysis can be started with an incomplete model and can help to identify missing parts. Furthermore, the control structure does not have to be linear. It may have a clear vertical linear hierarchy as well as multiple controllers at the same level that do not control each other or controllers that all control the same process. Controllers at the same level may communicate with each other, represented by a horizontal arrow. Additionally to the diagram, all clarifying information

or assumptions should be clearly documented. This concerns, among others, description and purpose of controllers and necessary information about control actions or feedback and controlled processes in general.

2.1.3 Identify Unsafe Control Actions

Based on the control structure, Unsafe Control Actions (UCAs) can be identified. A *UCA* is a control action that leads to a hazard in a specific context and worst-case environment. Hereby, *unsafe* refers to the identified hazards. A control action can be unsafe in four ways:

- ▻ Not providing the control action leads to a hazard
- ▻ Providing the control action leads to a hazard
- ▻ Providing a potentially safe control action but too early, too late, or in the wrong order
- ▻ The control action lasts too long or is stopped too soon

For each control action all types should be considered but some may be not applicable in every case. Additionally, there can exist more than one UCA in a single category. Each UCA consists of five elements: The source, the type, the control action that is unsafe, the context in which it is unsafe, and tracing to the hazards the UCA causes. Thereby, the source is the controller that sends the control action and the type is one of the four categories listed above.

Analogous to the hazards, constraints should be defined for the UCAs called *controller constraints*. They specify the behaviors of the controllers that must be satisfied in order to prevent the UCAs.

2.1.4 Identify Loss Scenarios

The last step of STPA identifies the loss scenarios, which are the causes for the UCAs. A *loss scenario* describes the causal factors that can lead to a UCA and hence to hazards. Two types must be considered when identifying them: Scenarios that lead to UCAs and scenarios in which control actions are improperly executed or not executed at all.

The first type of loss scenarios can be further divided into *unsafe controller behavior* and *causes of inadequate feedback and information*. Unsafe controller behavior can be identified by starting with a UCA and working backwards to explore the causes for (not) providing the control action. These scenarios may include: failures related to the controller, inadequate control algorithm, unsafe control input, and inadequate process model. In order to find the causes of inadequate feedback and information, the source of them must be examined.

For the causes of inadequate feedback and information, factors that affect the control path as well as factors that affect the controlled process, must be considered. Adding sensors and actuators to the control structure may help to identify those factors. It is important to not list the individual factors outside the context of a scenario. Otherwise, interaction between and combination of factors leading to hazards may be overlooked. In order to identify the causal factors more efficiently, Leveson provides guide words [Lev16].

2.1.5 STPA Outputs

Overall, STPA contains eight aspects, including the control structure. The traceability between these aspects, explained in the previous sections, is shown in Figure 2.2. The results of STPA can be used to create requirements, identify design recommendations, define test cases, and more [LT18].

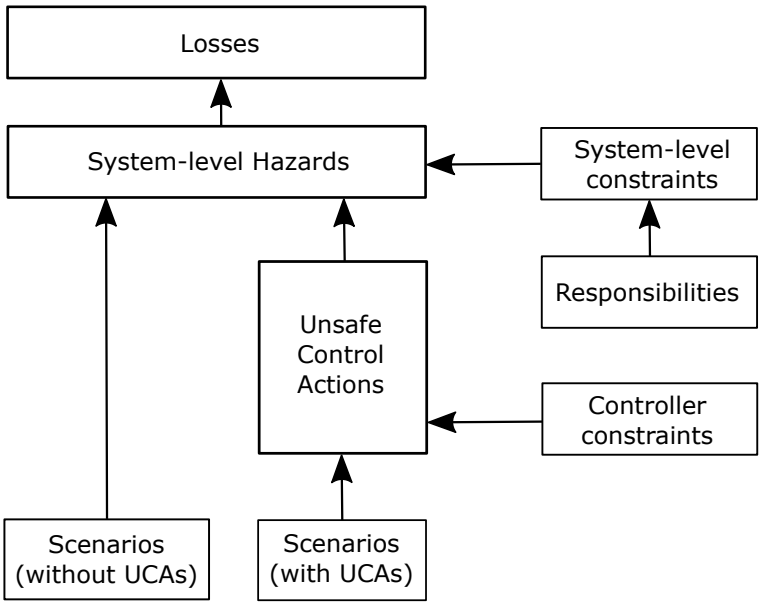


Figure 2.2. Traceability between the STPA aspects [LT18].

2.2 STPA Context Table

Applying STPA manually can be very tedious especially in complex systems. Partially automation could reduce the necessary work. Thomas developed in his dissertation an automation for identifying UCAs and requirement generation as well as a method to detect conflicts between the safety requirements and other requirements [Tho13]. In order to automatize the UCA generation, he formalized them first. An UCA consists of four parts: source controller, type, control action, and context. The context is defined by variables and their values. As an example he uses a train that has the variables *train motion* with the values *stopped* or *moving*, *train location* with the values *at platform* or *not at platform*, and *emergency* with the values *emegerncy exists* and *no emergency*. The controller's process model must contain these variables and their values. Hence, the variables in the process model are a superset of the context variables. Moreover, the process model variables can be organized into a hierarchy. With this formalization UCAs can be stated more precisely.

The identification of hazardous contexts can not be automated effectively but a framework can guide the process. Thomas proposes a new procedure to systematically identify UCAs. It

is divided into two parts: the first one inspects hazardous control actions whereas the second one inspects control actions that are not provided in a context in which inaction leads to a hazard. The process of the first part is as follows: At first a controller and its associated control actions are selected. Afterwards, the process model of the controller is defined in order to determine the states that affect the safety. Subsequently, potentially hazardous control actions can be identified. This is done by inspecting all possible combinations of relevant process values and determining for each of them whether sending the control action will lead to a hazard. This is called the context table, an example can be seen in Table 2.1. In this table the control action *open door* is inspected and there are three process variables: *train motion*, *emergency*, and *train position*. Each row represents a possible combination of the values of the variables whereby some rows are taken together using the *doesn't matter* value. The last column determines whether the control action is, in the given context, hazardous. This column can be further divided by adding timing information such as *too early* or *too late*. The second part of the process uses the same basic process. The process model variables, their potential values, and the possible combinations are determined. Then instead of determining whether sending the control action leads to a hazard, it is determined whether the absence of the action would lead to a hazard.

Table 2.1. Example of a context table [Tho13].

Control Action	Train Motion	Emergency	Train Position	Hazardous control action?		
				If provided any time in this context	If provided too early in this context	If provided too late in this context
Door open command provided	Train is moving	No emergency	(doesn't matter)	Yes	Yes	Yes
Door open command provided	Train is moving	Emergency exists	(doesn't matter)	Yes	Yes	Yes
Door open command provided	Train is stopped	Emergency exists	(doesn't matter)	No	No	Yes
Door open command provided	Train is stopped	No emergency	Not aligned with platform	Yes	Yes	Yes
Door open command provided	Train is stopped	No emergency	Aligned with platform	No	No	No

The described process provides a systematic approach and guidance for identifying UCAs. Nevertheless, it is still "brute-force" and can be very time consuming when applying it to contexts with many variables and values. This is why Thomas additionally proposes automated algorithms assisting in executing this procedure. In order to generate a list of all potential UCAs (all possible four-tuples), the following information is needed: the hazards, the controllers in the system, the control actions for each controller, the relevant variables for the hazards, and potential values for each variable. The decision whether a potential UCA is

indeed an UCA can not be automated. The analysts still have to go through the generated list and decide for each item whether it is unsafe or not. Usages of this automated method on real systems identified safety-critical requirements not considered in the original development of the systems.

Based on the identified UCAs, requirements must be created in order to define the wanted behavior. For this procedure Thomas also proposes an automated algorithm. He defines three functions: $HP(H, SC, CA, Co)$, $HNP(H, SC, CA, Co)$, and $R(SC, CA, Co)$. $HP(H, SC, CA, Co)$ returns true if providing the given control action CA from the source controller SC leads to the hazard H in the given context Co. $HNP(H, SC, CA, Co)$ covers the other case: it returns true if not providing the control action CA leads to the hazard H. After defining both of the functions, $R(SC, CA, Co)$ is defined for specifying the system behavior. It returns true if the controller SC must send the control action CA in the given context Co. In order to prevent hazards, $R(SC, CA, Co)$ must fulfill the following:

1. If a control action CA leads to any hazard in a specific context Co, the control action CA must not be provided in the context Co. Formally: if $HP(H, SC, CA, Co)$ returns *true* for any hazard H, $R(SC, CA, Co)$ must return $false$.

2. If not providing the control action CA in a specific context Co leads to hazard, this control action must be provided in this context. Formally: if $HNP(H, SC, CA, Co)$ returns *true* for any hazard H, $R(SC, CA, Co)$ must return *true*.

The resulting requirements can be documented by using a formal requirements language for example SpecTRM Requirements Language (SpecTRM-RL). It can be the case that there is no safe action for a specific context. If action as well as inaction leads to the same hazard, the system has a fundamental design flaw that must be eliminated. If action as well as inaction causes a hazard that is different, there exists a conflict between two safety-related goals, which must be resolved. In order to detect these flaws, Thomas introduces a consistency check: Given a controller, a control action, and a context, if a hazard exists for which HP is true, HNP must be false for all hazard. This can be checked automatically independent of R. The found conflicts can for some cases be automatically resolved. If the losses and hence the hazards are prioritized, the control action that causes the least important hazard can be selected to resolve the conflict. However, engineers using their creativity and expertise can find the best solutions. Hence, the automation should mostly be used to detect conflicts and to tag them for review.

For complex systems the context tables can get very large. That is why Thomas proposes three techniques to handle them better: abstraction and hierarchy, logical simplification, and continues process model variables. *Abstraction* can be applied to the control actions as well as to the variables in order to allow high-level analysis. This can be refined later for a more detailed analysis. With *logical simplification* several similar rows in the context table can be combined using *does not matter* terms for variables. Tools can help to perform this reduction automatically. For *continuous process variables* it is not necessary to consider an infinite number of values. Based on the hazards, more fitting values can be determined. For example instead

of an exact velocity, only the values *moving* and *not moving* could be used. Another helpful method to handle complex systems is the *rule-based approach*. This approach defines a set of rules that determine hazardous contexts. An automated tool can, after generating the context table, apply the rules and mark the rows that lead to a hazard. Afterwards, the tool could apply logical simplification, check whether rules are conflicting, and check whether there are still rows for which no rule applies. This technique was already successfully applied for tables with hundred of rows using only a few rules.

If the proposed process for identifying UCAs is used, the results can be further used to guide the last step of STPA: the identification of loss scenarios. Thomas presents two parts for which the list of hazardous control actions can be used: Identification of scenarios in which safe control actions may not be executed and identification of scenarios that help identify causes of the UCAs. For scenarios with not executed safe control actions, the identified UCAs must be inspected. They contain the needed information to identify scenarios in which a provided control action that is not followed leads to a hazard: the command and the context. Afterwards, the causal factors causing the scenario must be identified. These can but do not have to be component failures. Given the UCAs and context tables the basic scenarios can be generated automatically, but the causal factors must be identified manually. Nevertheless, this approach can lower the duplication of work between the identification of UCAs and of scenarios and guarantees that only scenarios are considered that definitely lead to a loss. Furthermore, direct traceability between the context tables and the scenarios is provided. For the identification of scenarios that help identifying causes of UCAs, an important causal factor is a flawed process model. If the controller's information about the system and/or environment is incorrect, it can lead to UCAs. The context tables, generated in the previous step, provide the contexts in which flaws can and cannot cause hazards. This can be used to identify the type of process model flaw that can cause a hazard. Relevant process model flaws for the example context table in Table 2.1 can be seen in Figure 2.3. For each hazardous control action, including inaction, the process model flaws are listed that lead to the UCA. Engineers need to know which types of flaws cause hazards in order to design a safer system. Moreover, focusing on these types instead of the individual variables can reduce the workload. Originally, each UCAs is inspected separately, which can lead to much repetition in the identified causes. In the new approach this repetition is eliminated by first identifying the general process model flaws for all UCAs. This leads to a set of unique flaws in which each flaw can be hazardous in several ways. If context tables were used in the previous STPA step, the process model flaws can be generated automatically and traceability to the corresponding hazards is provided. Besides the control path, also the feedback path can be analyzed, which is done in a similar way. For this procedure Thomas translated the four types of UCAs to feedback types:

- A feedback parameter required for safety is not provided
- An incorrect feedback parameter is provided that leads to a hazard
- Correct feedback is provided too late, too early, or out of sequence, causing a hazard

▷ Correct continuous feedback is stopped too soon or applied too long, causing a hazard

Hazardous control action	Relevant process model flaws
Door open command not provided when train is stopped at platform and person in doorway	• Controller incorrectly believes train is moving • Controller incorrectly believes no person in doorway • Controller incorrectly believes train is not aligned
Door open command not provided when train is topped and emergency exists	• Controller incorrectly believes train is moving • Controller incorrectly believes no emergency exists
Door open command provided when train is moving	• Controller incorrectly believes train is moving
Door open command provided when train is stopped unaligned with platform and there is no emergency	• Controller incorrectly believes train is aligned • Controller incorrectly believes there is an emergency
Door open command provided more than X seconds after train stops during an emergency	• Delayed realization that train is stopped • Delayed realization of emergency

Figure 2.3. Example of process model flaws [Tho13].

Gurgel et al. proposes the rule-based approach [GHD15]. The authors state that defining the hazardous contexts is the most critical task and it is time consuming doing it manually as well as exhaustive and demands a careful analysis since the number of contexts can get very large. The goal of the proposed rule-based approach is to aid the identification of those hazardous contexts. A *rule* is a logical expression of variable states and means that those states represent a hazardous context. For contexts in which the status of a variable does not matter the keyword *ANY* can be used, meaning that any state can be considered for that variable. After defining the rules, the analyst must check the rules for conflicts, redundancy, and correctness. For each hazardous context a safety constraint can be defined. In order to test their approach, the authors developed a tool that automatically generates the context table based on the defined rules. They conclude that the tool saves great effort and working time due to the automation and was useful for generating hazardous contexts based on the rules. Furthermore, it helped to point out rework of contexts due to addition or removal of rules.

2.3 Used Technologies

This section gives an overview of the technologies used in the implementation of the DSL for STPA. First, Section 2.3.1 presents the VS Code Extension Application Programming Interface (API), which is needed to implement the DSL as an VS Code Extension. Afterwards, Section 2.3.2 explains the Eclipse Layout Kernel (ELK), which is used for layouting diagrams. The introduction of Sprotty in Section 2.3.3 also covers the usage together with a lanugage

server as a VS Code Extension. At last Section 2.3.4 outlines Langium, including a possible combination with Sprotty.

2.3.1 VS Code Extension API

VS Code provides the ability to customize and enhance it through the Extension API[1]. With the help of extensions, new programming languages can be supported, webviews displaying custom webpages can be created, the look of VS Code can be changed with a color theme, and more. The extension entry point, activation events, and contributes are specified in the *package.json*. The entry file must contain two functions: `activate` and `deactivate`. The function `activate` is executed when the activation event, specified in the *package.json*, happens and activates the extension. For example, when implementing a language server extension, the `activate` function can start the language client. With the `deactivate` function a clean up can be done before deactivating the extension. The contributes in the *package.json* are for example commands, which are provided to the user. These commands also have to be implemented with `registerCommand`, which can be done in the `activate` function.

2.3.2 ELK

ELK[2] is a collection of layout algorithms as well as an infrastructure that connects diagram editors or viewers to automatic layout algorithms. However, ELK itself does not render the drawing. It only computes positions and possibly dimensions for diagram elements. Besides a collection of layout algorithms, several layout options are provided. For usages in JavaScript, the layout-relevant part of ELK is also available in elkjs[3].

The main layout algorithm provided by ELK is the layered algorithm[4]. It is based on the approach of Sugiyama et al. [STT81]. The algorithm places the nodes into subsequent layers in order to route as many edges as possible into one direction. Generally, the algorithm consists of five phases [SSH14]: Cycle breaking, layer assignment, crossing minimization, node placement, and edge routing. *Cycle breaking* is needed to create a topological ordering of the nodes. Therefore, cycles must be eliminated. Based on this ordering, the layers can be assigned. Thereby, it must be considered that the source and target of an edge are not allowed to be in the same layer. The *crossing minimization* phase orders the nodes in their layer with the goal to reduce edge crossings between consecutive layers. Afterwards, *node placement* assigns the actual vertical positions to the nodes and the last phase routes the edges.

Besides this standard approach, ELK provides layout options to change the strategy of each phase. If the strategy is set to `INTERACTIVE`, the position coordinates of the nodes must be set manually before layouting. Based on these positions, the layer and the position in the layer are determined by the layout algorithm. For example in the crossing minimization phase in

[1] `https://code.visualstudio.com/api`

[2] `https://www.eclipse.org/elk/`

[3] `https://github.com/kieler/elkjs`

[4] `https://www.eclipse.org/elk/reference/algorithms/org-eclipse-elk-layered.html`

a left-to-right layout, the nodes get ordered in their layer based on their y value instead of minimizing edge crossings. The same applies for the layer assignment. Based on the x value and the width of the nodes, the layers are assigned such that overlapping nodes are in the same layer and non-overlapping nodes in different layers. However, in-layer edges are still not allowed. Finally, the actual position coordinates of the nodes are updated based on the assigned layer and position in the layer.

2.3.3 Sprotty

Sprotty[5] is an open-source web-based framework for diagrams implemented in TypeScript. It uses Scalable Vector Graphics (SVG) and the rendering is stylable with Cascading Style Sheets (CSS). Additionally, animations are included and transitions for morphing diagrams are provided. Sprotty also provides a connection to ELK for automatic layout of the diagram. Ideally, an app using Sprotty consists of a client and a server. The client only has the model of the current diagram and renders it while the server has the underlying semantic model and knows how to map it to diagram elements. Sprotty integrates with the Language Server Protocol (LSP). For example a Xtext-based language server can be combined with it[6].

In order to combine Sprotty with a language server as a VS Code extension[7], three components are needed: a languager server, a webview, and the extension. An overview of the architecture can be seen in Figure 2.4. As *language sever* again Xtext could be used. The *webview* contains the Sprotty client. As usual, it is responsible for the rendering and the creation of SVG as well as processing the user events. Furthermore, Sprotty `Actions` can be exchanged with the extension. The *extension* is responsible for starting the language server as well as creating the webview and connecting it to the language server when the command to show the diagram is triggered by the user. It is also responsible for providing the command in the first place.

When using Sprotty for an application, a container module configuring the diagram elements and a `SprottyStarter` creating the container is needed. Diagram elements are configured by binding a type, for example `node`, to a model element and a view. For both, Sprotty provides standard implementations, for example `SNode` with `RectangularNodeView` and `SEdge` with `PolylineEdgeView`. However, custom model elements and corresponding views can be implemented by extending these standard ones. Besides implementing a view for a graph element, SVGs can be additionally styled with CSS[8]. In a CSS file, styles such as colors can be added. When using a class selector, the style is only applied to elements with a specific class attribute[9]. A class selector starts with a period followed by a class name. Afterwards, the style is stated in brackets. Only the elements that are provided with the specific name of the class selector in the actual class attribute are affected by the defined style.

[5] `https://www.typefox.io/blog/sprotty-a-web-based-diagramming-framework`, `https://github.com/eclipse/sprotty`

[6] `https://github.com/eclipse/sprotty-vscode/tree/master/examples/states-xtext`

[7] `https://www.typefox.io/blog/using-sprotty-in-vs-code-extensions`

[8] `https://www.w3.org/Style/CSS/Overview.en.html`

[9] `https://www.w3schools.com/css/css_selectors.asp`

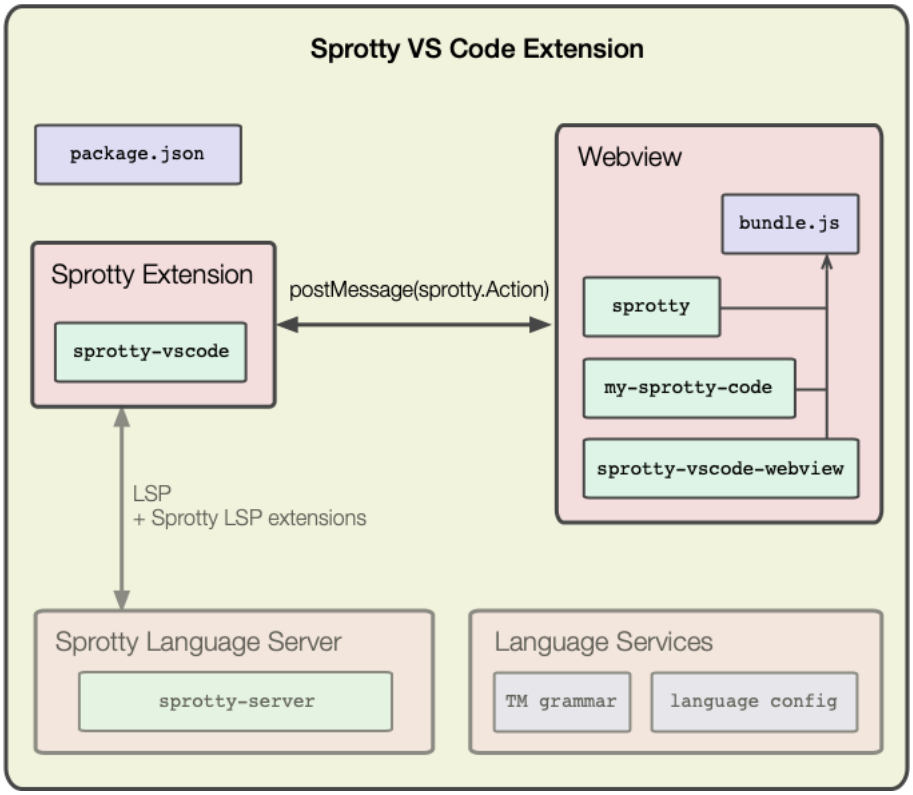


Figure 2.4. Architecture of a VS Code Extension using Sprotty[10].

2.3.4 Langium

Langium[11,12] is an open source language engineering tool, created by TypeFox[13] with the goal to lower the barrier of creating a DSL. It supports the LSP and is written in TypeScript. The motivation of developing Langium is that the VS Code Extension API has become increasingly relevant. In contrast to Xtext[14], Langium allows language engineering in TypeScript, the same technology used for VS Code Extensions. When creating a DSL with Langium, three files are mandatory: a file containing the grammar, a file defining the module, and a main file that creates the connection to the client, creates the wanted services defined in the module, and starts the language server.

The grammar declaration language is similar to Xtext. A Langium *grammar file* starts with the name of the grammar using the keyword `grammar`. There are three rule types: terminal rules, parser rules, and the entry rule. For all the Extended Backus-Naur Form (EBNF) is used. *Terminal rules* start with the keyword `terminal` and create an atomic symbol while parser

[10] https://github.com/eclipse/sprotty-vscode
[11] https://langium.org/
[12] https://langium.org/docs/grammar-language/
[13] https://www.typefox.io/
[14] https://www.eclipse.org/Xtext/documentation/

```
export const HelloWorldModule: Module<HelloWorldServices, PartialLangiumServices &
    HelloWorldAddedServices> = {
   validation: {
      ValidationRegistry: services => new HelloWorldValidationRegistry(services),
      HelloWorldValidator: () => new HelloWorldValidator()
   }
};
```

Listing 2.1. Example of a language server module.

rules define the structure of objects to be created by the parser and create an Abstract Syntax Tree (AST). A *parser rule* starts with the name of the rule followed by a colon. Afterwards, the rule is stated, for example[15]: `Person: 'person' name=ID;`. This rule creates an object of type `Person` with the property `name` that must match the terminal `ID`. The *entry rule* defines the starting point of the parsing. It is stated similar to a parser rule but starts with the keyword `entry`. In addition to the grammar file, a *langium-config.json* file is required that references the grammar file. Based on the grammar declaration, a parser and an AST are generated.

Langium also offers the most essential language services: completion, validation, go to definition, find references, document highlights, and document symbols. Default implementations are provided for these services. However, they can be overridden with custom implementations in the module of the language server. An example with a custom validator can be seen in Listing 2.1. Besides overriding default implementations, also new services can be registered here. Langium can also be used in combination with Sprotty. Therefore, the `DiagramGenerator` service must be overridden, which translates a model of the DSL to a SGraph. In order to layout the generated graph, also the `ModelLayoutEngine` service can be configured with the `ElkLayoutEngine`. If specific options should be set for the layout, the `LayoutConfigurator` service is used.

In the *main file*, the connection to the client is created and the defined services are created by creating the module. If only Langium is used, it is sufficient to only start the language server here. However, if Langium is used in combination with Sprotty, also the `addDiagramHandler` method must be called. Furthermore, reactions to notification send to the language server can be defined here.

[15] `https://langium.org/docs/grammar-language/`

Chapter 3

Related Work

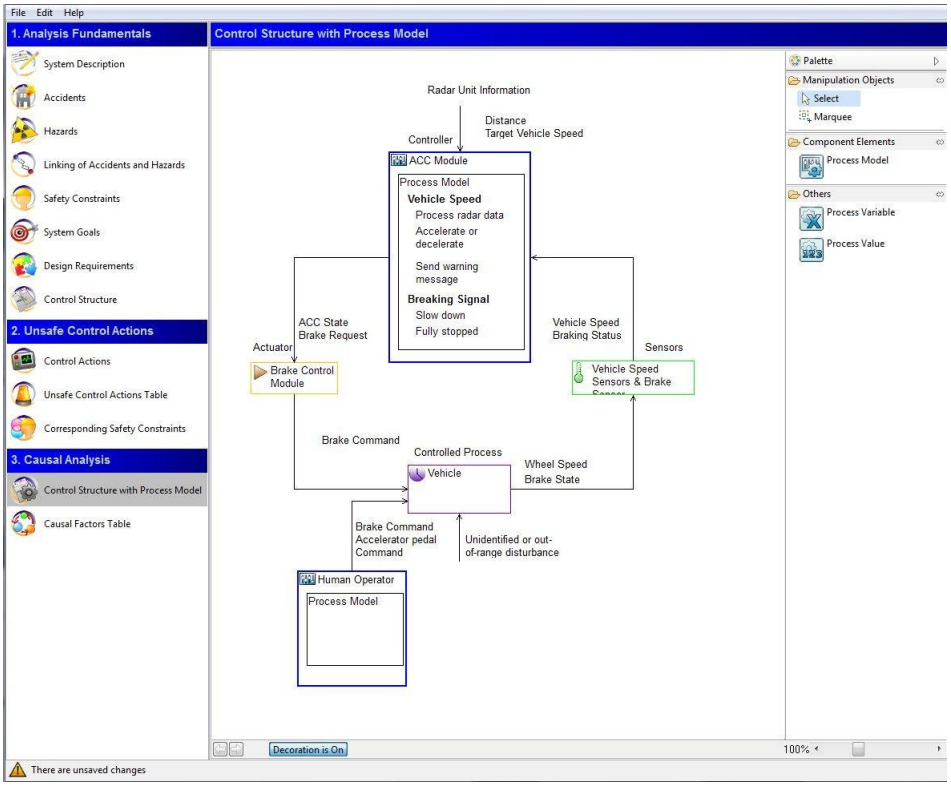
There are already several tools supporting the usage of STPA. The challenge of such tools is to help applying STPA by providing a systematic process, aiding in repetitive tasks, and resulting in a complete list of safety requirements [SPP+19]. In this chapter a variety of tools supporting STPA is presented. Additionally, Section 3.12 outlines comparisons of a subset of them.

3.1 A-STPA

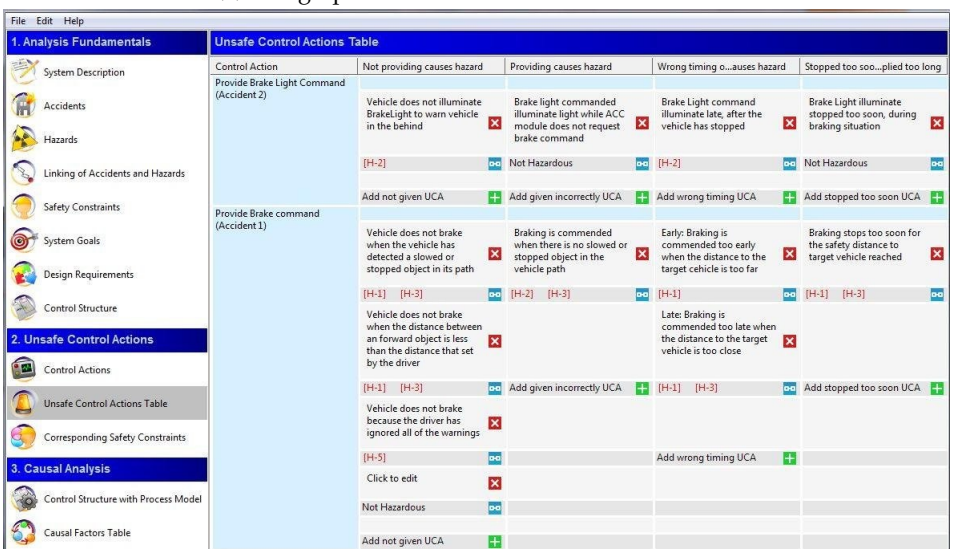
A student project in the software engineering programme of the University of Stuttgart developed A-STPA in order to assist analysts applying STPA [AW14]. The aim of the tool is to automate all activities during STPA. It is an open-source tool developed in Java and built on the Eclipse platform. The first version supports the management of the STPA aspects and tracing between them. For the creation of the control structure, a graphical editor is provided (Figure 3.1a) that also supports the creation of process models. UCAs and causal factors are maintained in tables. Figure 3.1b shows an example of a UCA table. The results of the analysis can be exported as a PDF.

The process in A-STPA consists of four phases. At first, data models for every STPA aspect are provided. Afterwards, mapping rules between the components are added to link the components, for example hazards to accidents. Additionally, the control actions are extracted from the control structure and added into a table. In the third phase, analysis information is put in the tables including UCAs, safety constraints, and causal factors. At last, the components of the control structure can be edited. Since STPA is an iterative process, the user can perform changes in a step, which are then passed on to subsequent steps.

The user interface of A-STPA consists of several views. Two of them are shown in Figure 3.1. On the left side in the tool the current view can be selected. The views are categorized into three groups. *Analysis Fundamentals* contains the views for the fundamentals such as losses, hazards, and the control structure. *Unsafe Control Actions* contains a view for control actions, which are automatically extracted from the control structure and added to a table. The table can also be edited manually. This group also contains the view for UCAs and corresponding safety constraints. These constraints are automatically derived from the UCAs. However, the user still can edit the table manually. The last group is *Causal Analysis* and it contains a view for the causal factors table.



(a) The graphical editor for the control structure.



(b) A UCA table.

Figure 3.1. A-STPA tool[a].

[a] https://sourceforge.net/projects/astpa/

The eXtensible Markup Language (XML)-specification is used to document the results, but they can also be exported as an PDF file. As a major issue the authors state that there are no rules for drawing a control loop and missing standard notations for causal factors. One topic for their future work is to improve the guidance regarding the causal factor analysis.

Usages of A-STPA in different areas has shown some shortcomings [AW15b]. These concern the documenting of UCAs, drawing control structure diagrams, documenting causal factors, and supporting STPA application in different areas. Furthermore, Thomas proposed an improvement for identifying UCAs as explained in Section 2.2, which is especially interesting for safety analysts. However, it is not integrated in A-STPA. Since the architecture of A-STPA is not extendable to such improvements, the tool was superseded by the Extensible STAMP Platform (XSTAMPP).

3.2 XSTAMPP

The Extensible STAMP Platform (XSTAMPP) is open source and based on the Eclipse Rich Client Platform (RCP) [AW15b]. The goal is to support the usage of STPA in different areas and to be easily extendable. Moreover, there is potential to extend XSTAMPP with support for CAST. The main functions are the same as in A-STPA as well as additional ones. Among other functionality, the user can add new user interfaces to customize the project and several user interface editors can be opened at the same time. The export function is extended such that the results can also be exported as a JPEG or an Excel file and not just a PDF. In order to achieve the goal to be easily extendable, plug-in libraries can be integrated. As future work the safety requirements derived by STPA should be automatically transformed to formal specifications such as Linear Temporal Logic (LTL). This can help to verify design models and software code against the safety requirements.

A new version of the tool, XSTAMPP 2.0, is available as open source platform [AW16]. It is written in Java and based on the Eclipse Plug-in-Development Environment (PDE) and RCP. For this version, new plug-in tools were developed: Automated CAST (A-CAST), which implements the CAST functionality, and the Extended Approach to STPA (XSTPA). XSTPA integrates the proposed improvement from Thomas and supports automatic generation of context tables. These are used to define the safety requirements, which are automatically transformed into a formal specification in LTL. The algorithm for the transformation is explained by Abdulkhaleq and Wagner [AW15a]. Besides these two new plug-ins, there is still the A-STPA plug-in in order to apply STPA without the improvements proposed by Thomas. Furthermore, XSTAMPP 2.0 provides improved usability regarding the drawing of the control structure diagram. In the future, the developers want to add another plug-in called STPASec, which should implement the steps of STPA for Security (STPA-Sec). Additionally, a plug-in should be developed that connects A-STPA with model checking.

3.3 RM Studio

RM Studio developed a standalone module for STPA[1]. Generally, the module can be operated by one user or shared by multiple users. The STPA module provides an individual view for each aspect of STPA. An overview of the tool's interface can be seen in Figure 3.2. On the left side in the editor, the individual views can be selected. In the view for losses, hazards, and constraints the user can add components by entering an ID and a description. Additionally, the created losses are available in the hazard view and the hazards are available in the constraint view in order to enable linking between them. Another view shows the relationship between the losses and hazards graphically. An example can be seen in Figure 3.2. Losses as well as hazards are represented as rectangles with a symbol representing the aspect in the upper left corner. The references of the hazards to the losses are represented by arrows. The control structure can be created in its own view via drag and drop. Control actions that are stated in this structure are automatically extracted to the UCA view. In this view, the user can select a control action and can add UCAs for each category. Additionally, an UCA can be linked to hazards and constraints. The loss scenario view contains three windows: one lists the available control actions, another shows the control loop graphically, and in the third one the user can choose whether the loss scenario is based on a UCA or hazard and can add a scenario. A progress check is also provided in order to ensure completion of all STPA aspects. Furthermore, there is an option to create a full report, which generates a file containing all specified information.

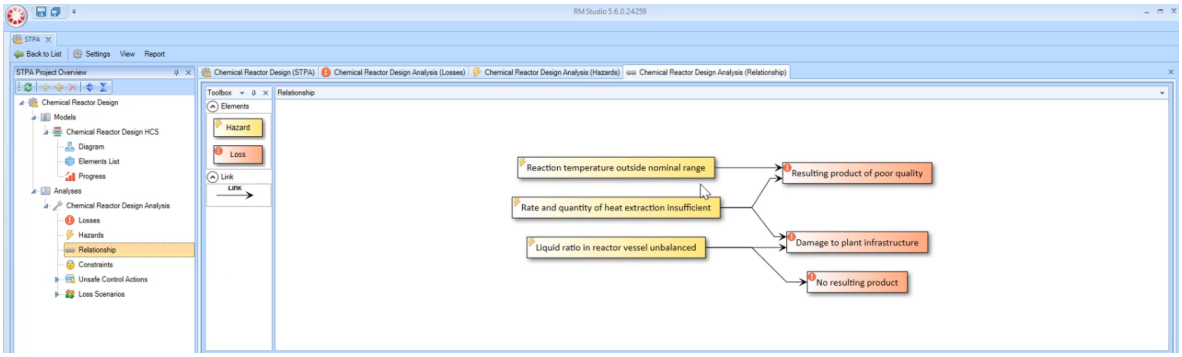


Figure 3.2. The user interface of RM Studio's STPA software solution[2].

3.4 SAHRA

Krauss et al. [KRR16] created an extension for Sparx Systems EA[3], called STPA based Hazard and Risk Analysis (SAHRA). It integrates STPA with visual modeling languages such as UML

[1] https://www.riskmanagementstudio.com/stpa-software-solution/

[2] https://www.riskmanagementstudio.com/stpa-software-solution/

[3] https://sparxsystems.com/

[Fil13] and System Modeling Language (SysML) [FMS06]. In order to provide additional diagram types, UML profiles, and more for STPA modeling, SAHRA includes a DSL profile for STPA. Although the development is specified to EA, the general concepts for extending UML are generic. Generally, the graphical elements for the different aspects of STPA are represented with rectangles containing an ID, a description, and an icon in the upper-right corner representing the aspect. Linking between defined components is shown with arrows as seen in Figure 3.3. Additionally, UCAs are connected to a keyword. Keywords are represented in the same way as the other graphical elements and contain as description the keyword. For modeling the control structure, SAHRA provides element types for controller, controlled processes, sensors, and actuators. The visualization is the same as for the other STPA aspects. Feedback and control actions are visualized by arrows with a description. Furthermore, a process model element can be used instead of a controller element. In contrast to the normal controller, the process model contains process variables. A process variable is visualized by a rectangle with a title — the variable name — and below a list of possible values.

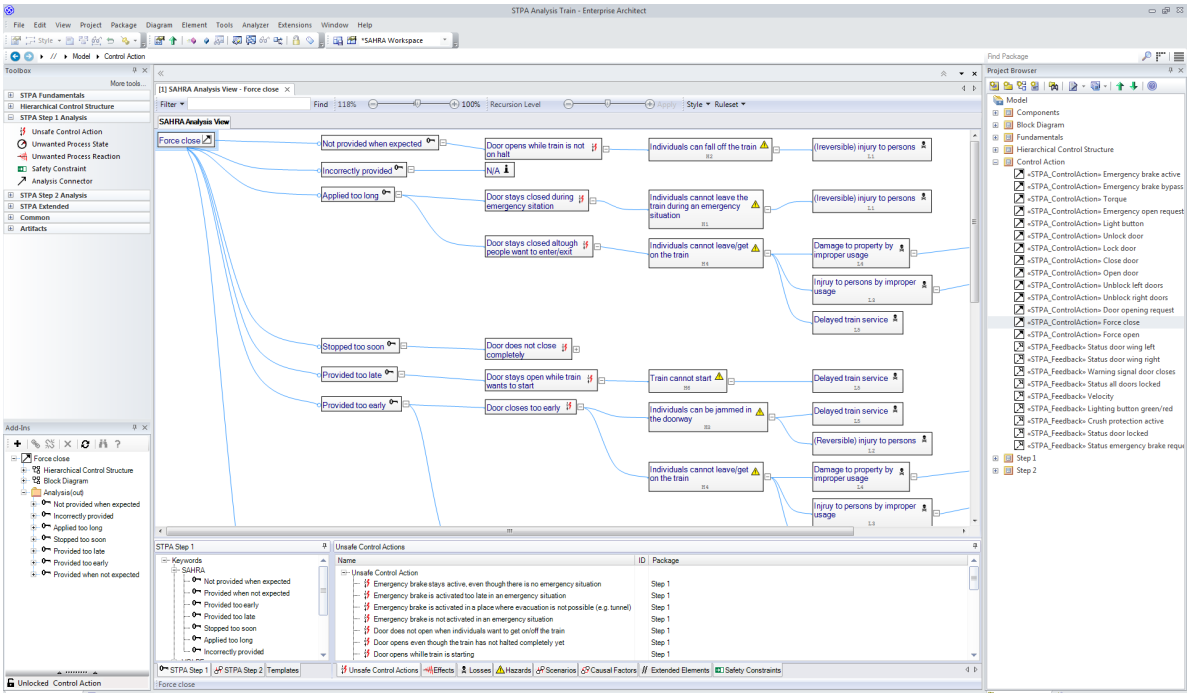


Figure 3.3. An example diagram in SAHRA [KRS+16].

According to Krauss et al. the mind map style provided by SAHRA helps to see relationships at a glance and provides more flexibility regarding documentation details [KRS+16]. Furthermore, because of the integration of SAHRA with EA, the user benefits from already existing features such as multi-user support, automation, and more. SAHRA has been already used in industry as well as in academic projects and it has been successful in both.

3.5 STAMP Workbench

The STAMP Workbench[4] provides several views for the different steps in STPA. At first, the preconditions can be added to a table by stating an ID and a description. For the losses, hazards, and constraints, there is a joint view that contains a table with a column for each aspect. An example of such a table can be seen in Figure 3.4a. Linking is done implicitly by adding the components that should be linked in the same row. As before each component has an ID and a description. Another view is the *Component Extracting Table*. It contains a table in which components for the control structure can be added. For a component, the following attributes can be entered: name, responsibility, control action, feedback, I/O, and remarks. Based on this table, the control structure diagram can be automatically generated. Nevertheless, the diagram can be drawn manually too, using drag and drop. The components of the control structure are visualized by rectangles containing the name of the component as seen in Figure 3.4b. For the *UCA view* a table is used, in which the control actions are automatically added by extracting them from the control structure. However, this table is not a context table as proposed by Thomas. In order to identify the causal factors, a control loop diagram can be opened for a UCA, only showing the relevant components and omitting the others. Additionally, hint words for identifying causal factors can be displayed. The results can be added in an additional view. In this view, the UCA that is inspected must be selected first and afterwards a table is displayed. The identified causal factors can be added to this table by stating an ID, the causal factor, the used hint word, and scenarios. At last, countermeasures can be documented in the countermeasures table.

3.6 Astah System Safety

The Japanese company ChangeVision created Astah[5,6] in 2006 as a UML modeling tool. Nowadays, it includes six different tools aiming to support modeling, safety assessment, and safety analysis for safety-critical systems. Among them is a tool for STPA that combines STPA with SysML by enabling model conversions. It provides diagrams for the control structure and the control loops as well as several tables for the other STPA aspects. Additionally, tables for preconditions and countermeasures are provided. The control structure diagram is created using SysML and drag and drop. When creating several diagrams, the *Control Structure Entire View* can be used to enable the simultaneous view of all views. When opening the *UCA table*, the control actions are automatically extracted from the control structure. For each control action the UCAs can be added according to their category just as in the STAMP Workbench. The control loop diagram, loss scenario table, and countermeasure table are also similar to the ones in the STAMP Workbench, including hint words display for the causal factors. When

[4] https://www.ipa.go.jp/sec/stamp_wb/manual/tutorial/basic/basic.html#ref-tutorial-basic

[5] https://astah.net/products/astah-system-safety/

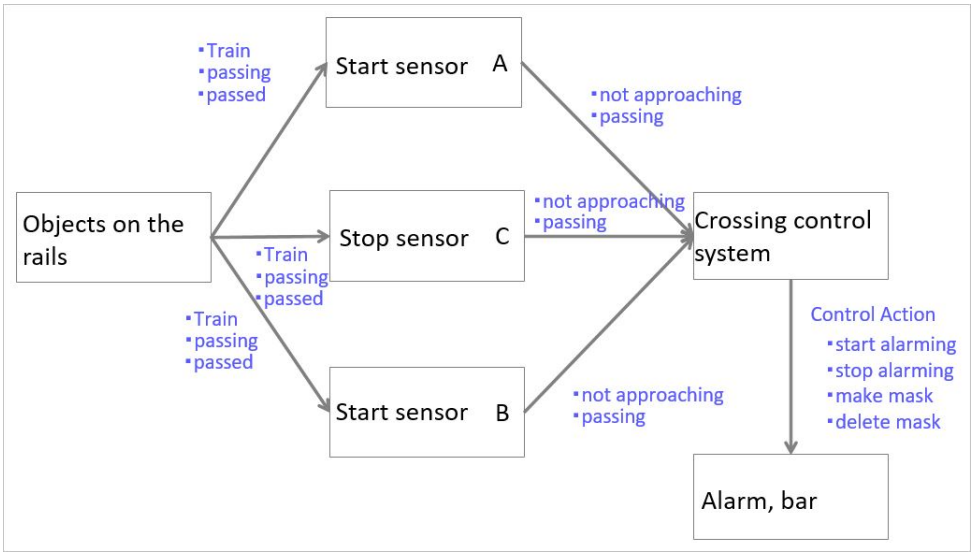
[6] https://astah.net/support/astah-system-safety/start-stamp-stpa-with-astah/

Accident Hazard Safety Constraint Table

Accident Hazard Safety Constraint Table / Accident Hazard Safety Constraint Table

Accident ID	Accident	Hazard ID	Hazard	Safety Constraint ID	Safety Constraint
A1	Collision with train and cars or people on the crossing	H1	Crossing does not close when the train is approaching or passing	SC1	Crossing must be closed when the train is approaching or passing
A1	Collision with train and cars or people on the crossing	H2	Crossing opens before the train has completely passed by	SC2	Crossing must not open when the train is passing
A2	Traffic jam occurs when the crossing continues to be closed	H3	Crossing is closed even when no train is approaching or passing	SC3	Must not close the crossing when no train Is passing or approaching
A2	Traffic jam occurs when the crossing continues to be closed	H4	Crossing does not open even after the train has passed	SC4	Must open the crossing after the train passed

(a) An example of a table in the STAMP Workbench.



(b) An example control structure in the STAMP Workbench.

Figure 3.4. The STAMP Workbench[a].

[a] https://www.ipa.go.jp/sec/stamp_wb/manual/tutorial/basic/basic.html#ref-tutorial-basic

the user is finished, the tables can be exported to Excel and the diagrams to image files. All in all, it is quite similar to the STAMP Workbench.

3.7 CAIRIS

CAIRIS[7,8] is a security design tool that was not built especially for STPA but can still be used. The supported concepts are analogous to the ones needed for STPA. For the security analysis, *obstacles* can be created. These can also be used to create losses, hazards, and UCAs by setting their category accordingly. Additionally, the hazards can be linked to the losses and the UCAs to hazards by using associations. In another view, *goals* can be created that can be used to create system-level constraints. A view is provided where the defined goals and obstacles are automatically visualized. The control structure must be modeled as a data flow diagram. It contains processes and data stores that are analogous to the control algorithms and process models in STPA. When defining the UCAs they can be associated to the data flow, at which point the appropriate keyword must be entered. Loss scenarios are created by *tasks*, which can be links to hazards and system constraints. CAIRIS provides model validation checks that can determine issues leading to the scenarios. If all aspects are defined, CAIRIS can automatically generate requirement specifications. The developers consider to create a more specific STPA specification document in the future.

3.8 SafetyHAT

Safety Hazard Analysis Tool (SafetyHAT) aims to help analysts become proficient with STPA [BVJ14]. It guides the analyst though the preparation and analysis of STPA by providing a streamlined data entry process. Additionally, a relational database is provided to organize and manage the data produced by the analysis. The STPA process is divided into eight steps as shown in Figure 3.5. The first three steps are entering system information, followed by four steps for the actual STPA analysis. At last, the results of the analysis can be exported to Excel. SafetyHAT was mainly developed for the analysis of transportation systems, which is why transportation-oriented guide phrases and causal factors for that area are included. Furthermore, the categories for UCAs are extended based on the experience made by applying STPA to transportation systems. However, the user can edit the categories and the guide phrases for causal factors.

3.9 An STPA Tool

The Massachusetts Institute of Technology (MIT) developed An STPA Tool [Suo16; ST14]. It allows the user to specify hazards, draw the control structure, and identify UCAs. For the

[7] https://cairis.org

[8] https://cairis.readthedocs.io/en/latest/stpa.html

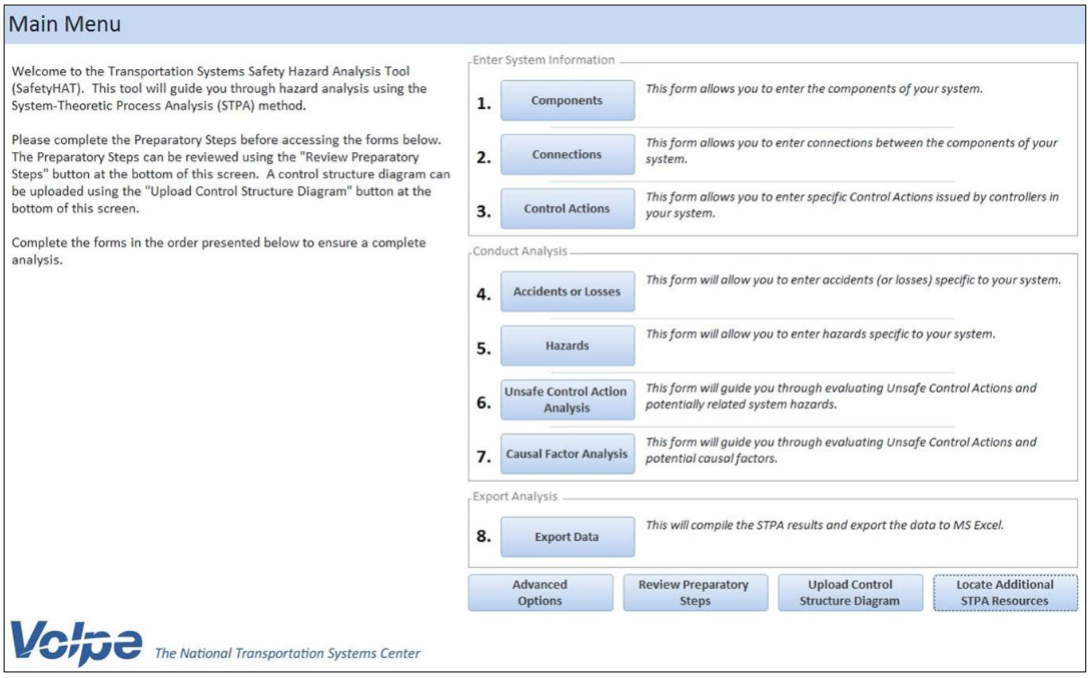


Figure 3.5. The main menu in SafetyHAT [BVJ14].

hazards and the control structure, a 2-D graphical editor is used. In order to identify the UCAs, context table templates are automatically generated based on the control structure. The user then selects which rows cause hazards. Moreover, the user can define rules that are used to automatically determine which rows cause hazards. Figure 3.6 shows the interface for that. Rules are defined by stating the values of the process model variables, whether the control action is provided or not, and the related hazards. Besides this formal definition, the rules are also shown in natural english. Conflicts between defined rules are automatically detected by the tool. Furthermore, logical simplification is used to simplify the table and the identified UCA can also be displayed in natural english. When the context table and thus the identification of UCAs is finished, safety requirements are automatically generated in executable forms using SpecTRM-RL. The results can be stored in a XML file, which can be exported to Specification Tools and Requirements Methodology (SpecTRM) for consistency and completeness checking.

The different notation forms can help the user to learn STPA and make the analysis more efficient [Suo16]. For beginners, the natural english may be easier to use while engineers who work with Excel files can export the context tables. For complex systems, the tool helps to identify UCAs with less time and effort according to Suo. However, some improvements are proposed. The tool can be extended to also support identification of loss scenarios. Moreover, more guidance could be provided and the tool-assisted analysis extended to include other emergent properties such as security as well.

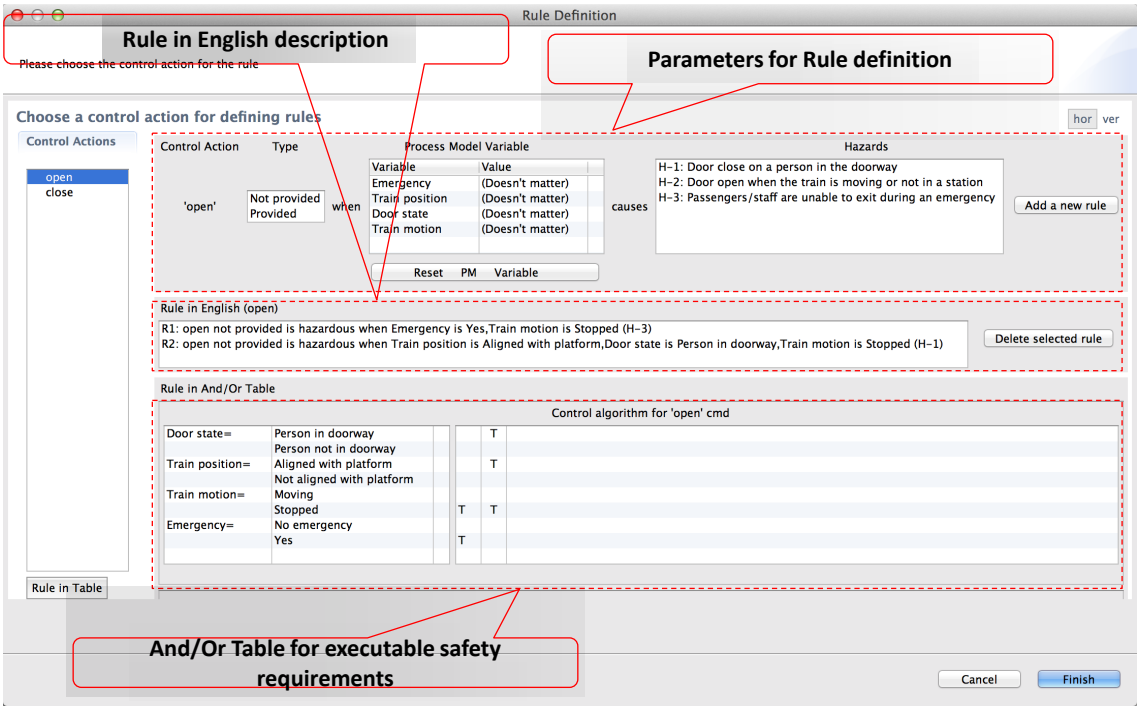


Figure 3.6. The interface for defining rules in An STPA Tool [ST14].

3.10 WebSTAMP

Souza et al. developed a STAMP compliant web application, called WebSTAMP [SPP+19]. It aims to help analysts performing STPA and STPA-Sec and it is a collaborative tool. In contrast to other tools, WebSTAMP focuses on the identification of UCAs and loss scenarios. Losses, hazards, system-level constraints, and control structure diagrams are not supported. For identifying UCAs, the rule-based approach is used using the context tables proposed by Thomas (see Section 2.2). Based on the process model variables defined by the user, the context table is automatically generated. An example can be seen in Figure 3.7. For each row in the table, the user can choose which UCA category, introduced in Section 2.1, lead to a hazard. Since this work is tedious, the rule-based approach proposed by Gurgel et al. (see Section 2.2) is also supported. If a rule is applied for a row in the table, it is stated in an additional column. For each identified UCA, a safety constraint is generated automatically. The loss scenarios are also documented in a table for each UCA alongside with the *associated causal factors*, *recommendations*, and *rationale*. They can be created by the user, but the tool also provides a set of generic scenarios, which can be adjusted. Additionally, WebSTAMP shows a guide question in order to help identifying important scenarios.

In order to evaluate WebSTAMP, the authors used it for analyzing a small set of systems. The results show that the tool helps to apply STPA/STPA-Sec more systematic, automated, and comprehensively. Furthermore, the chances of overlooking UCAs is reduced. For the future,

#	Glucose Level	Reservoir level	Battery level	Pump Operational Status	Index Rule	Control Action provided	Control Action not provided	Wrong order of Control Action	Control Action provided too early	Control Action provided too late	Control Action stopped too soon	Control Action applied too long
1.	Below	Below	Low	Transmitting	R1	Hazardous	?	?	?	?	?	?
2.	Below	Below	Low	Not transmitting	R1	Hazardous	?	?	?	?	?	?
3.	Below	Below	Normal	Transmitting	R1	Hazardous	?	?	?	?	?	?
4.	Below	Below	Normal	Not transmitting	R1	Hazardous	?	?	?	?	?	?

Figure 3.7. An example context table in WebSTAMP [SPP+19].

the developers want to add a Graphical User Interface (GUI) to draw the control structure and to extend the tool to also support CAST and leading indicators.

3.11 Prototype

In his master thesis, Ludvigson explored the functionality a tool could provide to support the application of STPA [Lud18]. Based on a set of suggestions and requirements he worked out, he implemented a prototype aiming to fulfill those requirements. For defining the losses and hazards, Ludvigson suggests strict list management with support of linking the individual components. Additionally, a visualization of the traceability would be helpful. Regarding the UCAs, providing a table in which they can be entered would be sufficient. However, the extension of Thomas (Section 2.2) could be used, in which case the tool should support the generation of a context table and also provide possibilities to simplify the table using for example logical simplification. Moreover, allowing the definition of rules would reduce the work for the analyst. In order to prevent that the analyst needs to provide the same information twice, information should be transferred between the tables and the control structure. Since STPA is not linear, re-evaluation suggestions could be shown when information is changed or added in a previous step. Furthermore, some form of version control would be helpful. Besides these requirements for the different steps of STPA, Ludvigson also proposes functional requirements for handling user profiles such as a profile system, authentication, and traceability of components to the user that created them. Moreover, some quality requirements should be fulfilled by a tool: usability, modifiability, interoperability, and security. *Usability* means that all STPA steps are supported. *Modifiability* allows a user to make changes regarding the analysis and a developer to make changes to the system. The possibility to exchange information between systems is guaranteed by *interoperability*. The last requirement, *security*, protects data and information from unauthorized access.

The prototype that was developed only supports a subset of these proposed requirements. Ludvigson states that it should not be used until more implementation is done.

3.12 Comparison

Ludvigsen [Lud18] compares his Prototype with other tools supporting STPA based on the set of suggestions and requirements he generated using literature and experience. In the

comparison the tools XSTAMPP, STAMP Workbench, SafetyHAT, and An STPA Tool are inspected. Table 3.1 shows the suggestions he states and whether they are provided by the tools. All tools provide a way to establish the fundamentals, such as losses and hazards. Thereby, only XSTAMPP and SafetyHAT provide support for managing lists of STPA components such as search, filter, and sort. These two and STAMP Workbench are also the only ones which provide help information to guide the user through the STPA process. Regarding the traceability, all tools offer basic support. A major difference between the tools is the identification of UCAs. While XSTAMPP, STAMP Workbench, and SafetyHAT use the original method, the Prototype and An STPA Tool use the context tables as proposed by Thomas. Both tools using the context table offer logical simplification and An STPA Tool supports the definition of rules for the table. However, support for redefining an UCA as a hazard is not provided by anyone. Creating a control structure is only supported by XSTAMPP, STAMP Workbench, and An STPA Tool. The Prototype and SafetyHAT rely on providing a control structure from another application. Nevertheless, all tools except the Prototype support hierarchical control structures and communication between the different levels in the control structure. Re-evaluation suggestions proposed by Ludvigson are not provided by any tool, not even the Prototype. Version control is also not supported by the already existing tools except for the Prototype. Regarding team management, XSTAMPP has support by allowing to assign roles such as admin or user. The Prototype should have support for multiple users accessing a project but it is not yet added whereas the remaining tools have no support at all. Overall, Ludvigson states that, in the preliminary study, XSTAMPP was found to be the currently best solution and the prototype needs more development before it is used.

Souza et al. [SPP+19] also identify requirements for STAMP based tools. These requirements, and which tools provides them, are shown in Table 3.2. The first six requirements are functional ones, whereas the last four are non functional requirements. According to the authors, STAMP Workbench and SafetyHAT do not really fulfill the requirements or it is not described. WebSTAMP and XSTAMPP however, fulfill most of the requirements. While WebSTAMP fulfills five of the functional and one of the non-functional requirements, XSTAMPP fulfills four functional and three non-functional ones. Regarding the automation of UCAs, it is not clear whether XSTAMPP provides support.

Table 3.1. Comparison of tools by Ludvigsen [Lud18].

Suggestion	**Prototype**	**XSTAMPP**	**STAMP Workbench**	**SafetyHAT**	**An STPA Tool**
List management	No	Partial	No	Partial	Unknown
Guide	No	Partial	Partial	Partial	Unknown
Traceability	Partial	Partial	Forced	Partial	Unknown
UCA table	No	Full	Full	Partial	No
Context table	Full	Partial	No	No	Full
Logical simplification	Partial	Partial	No	No	Full
Hazard rules	No	No	No	No	Full
Linking UCA to hazards	Full	Full	Partial	Full	Full
Redefine UCA to hazard	No	No	No	No	No
Continuous process model variables	No	No	No	No	No
Extracting fundamentals from control structure	-	Partial	Partial	-	Unknown
Importing or exporting control structure	No	-	-	No	-
Hierarchical control structure	No	Partial	Partial	Partial	Partial
Re-evaluation suggestion	No	No	No	No	Unknown
Version control	Partial	No	No	No	Unknown
Team management	Partial	Partial	No	No	Unknown

Table 3.2. Comparison of tools by Souza et al. [SPP+19].

Requirements	**WebSTAMP**	**STAMP Workbench**	**SafetyHAT**	**XSTAMPP**
Create safety and/or security analyses	Yes	Only safety	Only safety	Yes
Systematize and automate UCA identification	Yes	No	No	Yes
Systematize loss scenario identification	Yes	No	No	Unknown
Provide collaborative analysis	Yes	No	No	No
Provide change management	Yes	Unknown	Yes	Yes
Provide support for verification of the analysis	No	Unknown	Unknown	Yes
Provide rich user experience	No	Yes	No	Yes
Provde analysis reusability	No	Unknown	Unknown	Yes
Provide security environment	No	No	No	No
Provide portability	Yes	Only windows	Only windows	Yes

Chapter 4

Exploration of Risk Analysis

This chapter presents results of the exploration of the risk analysis topic. Over the years, many risk analysis techniques were developed in order to ensure the safety of systems. They can be grouped by so called accident models [Hol16a]: *sequential*, *epidemiological*, and *systemic*. These accident models, which are the foundation for the analysis techniques, are explored in more detail (Section 4.1). Section 4.2 outlines alternatives to STPA, their (dis-)advantages, and combinations of techniques. Afterwards, STPA related topics are presented that offer insight in the usefulness of STPA and possible improvements. In particular these topics are: use-cases (Section 4.3), extensions (Section 4.4), and improvements (Section 4.5). Conclusively, Section 4.6 presents leading indicators. STPA can help in the creation of them and the DSL could support this. However, this is not done in this thesis and is open for future work.

4.1 Causality/Accident Models

In the 1970s Hollnagel introduced the What-You-See-Is-What-You-Get (WYSIWYG) principle, which states that a printed version of a document should look the same as the screen image. Analogous, the What-You-Look-For-Is-What-You-Get (WYLFIWYG) principle was proposed [Hol16b]. It states that the assumptions about possible causes for accidents determine which kind of failures will be found. This means that the foundation of a risk analysis technique — the causality model — determines the kind of risks that can be found. In the following, the three accident model types are explained further in the same order they were developed [WBV+17], starting with the sequential type (Section 4.1.1), which is the oldest one, followed by the epidemiological type (Section 4.1.2), and at last the systemic type (Section 4.1.3). Concluding, Section 4.1.4 compares these accident model types.

4.1.1 Sequential Model

The first accident model types are sequential ones, also called “Linear Chain-of-Failure Events Model” [Lev21]. One of their principles is *decomposition* to handle complex systems. Thereby, the system gets divided into smaller components that are examined individually and the results are combined in order to understand the whole system. For example, physical components are broken into components that interact with each other [LT18]. The behavior is modeled as a chain of events where each event is triggered by the preceding one. The basic idea is that accidents are caused by a chain of failures resulting in incorrect behavior or accidents [Lev21]. Thereby, direct causality is assumed, meaning that one event or failure is

the reason for the following one. In order to prevent accidents, one event in this chain must be eliminated, which leads to preventing the next event from occurring. If the model is used to analyze the causes of failures, the analysis starts with the accident and then works backwards to an initial event, also called the *root cause* [Lev21]. Which event is labeled as initial is not clear and depends on the analyst. Primarily, this accident model considers human faults and component failures as causes of accidents.

One model that belongs to the sequential category is the *Domino Model* invented in 1931 by Heinrich [Lev21]. It contains five dominos representing events and the fall of one domino leads to the fall of the next one. The events the dominos represents are shown in Figure 4.1: Social Environment and Ancestry, Fault of the Person (Carelessness), Unsafe Act or Condition, Accident, and Injury. If one of the dominos is removed, the accident is prevented. Heinrich states that the best way to prevent an accident is to remove the third one. There exists some extensions of the Domino model that contain more factors. However, common to all is the linearity of events and that mostly an error caused by humans is the cause for accidents.

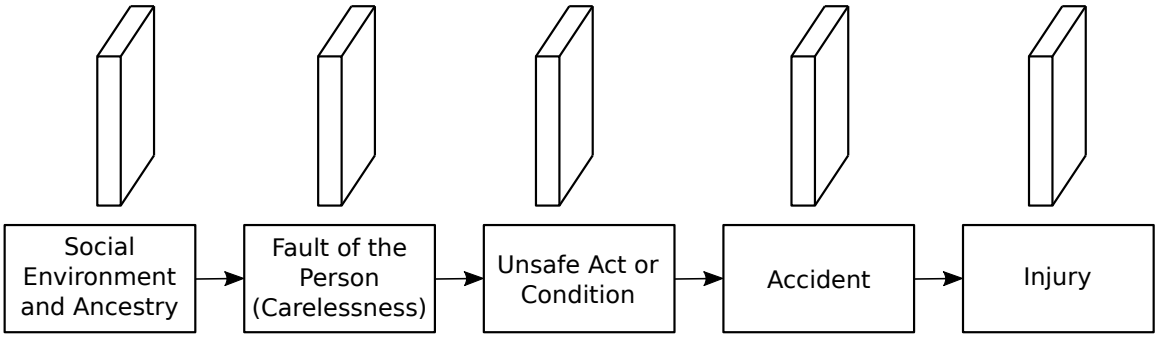


Figure 4.1. The Domino Model of Heinrich as illustrated by Leveson [Lev21].

One advantage of sequential models is that they are easier to understand than other models, but since they focus on component failure and human error, vital accident causes may be missed [WBV+17]. The basic assumptions of the model about the causes of accidents were true in the past and still are for certain properties in the systems today. However, nowadays, systems are more complex, which leads to new causes for accidents, such as unsafe interactions between components where each component itself does not fail [LT18]. Additionally, the role of the human in complex systems has changed leading to fewer accidents caused by human error.

Examples for risk analysis techniques based on sequential models are FTA (Section 4.2.1) and Failure Modes and Effects Analysis (FMEA) (Section 4.2.2) [LT18].

4.1.2 Epidemiological Model

In the 1980s epidemiological models arose due to the need of more powerful and complex accident models [Hol16a]. The analysis of major industrial accidents in the 1980s showed

that sequential models are not sufficient anymore. Epidemiological models differ from the sequential ones in four main points [Hol16a]:

performance deviation Human error as the main cause for accidents is replaced by performance deviation. This includes technological components as well as persons.

environmental conditions Epidemiological models consider environmental conditions that can lead to the performance deviation. These conditions exist for technology as well as for persons.

barriers In order to prevent accidents, barriers are used. They can be added at all stages of the accident development. A barrier can be material, functional, symbolic, or immaterial and has a function, for example preventing movement [Hol99].

latent conditions Failures can be divided into latent and active failures, which lead to the introduction of latent conditions. Latent conditions are already present before an accident sequence starts while active failures trigger such a sequence. The causes for latent conditions can be for example design failures or degradation of system functions (e.g. corrosion).

Generally, an accident is viewed as the result of the interaction of a host, agent(s), and environmental factors. The interactions are complex and random and can not be described by linear interactions between events [Lev21]. Leveson distinguish two types of epidemiology: descriptive epidemiology and investigative epidemiology. In the *descriptive epidemiology* incidence, prevalence, and mortality rates for accidents in large population groups are determined based one age, sex, etc. These are used to describe the general distribution of injuries in the population. In *investigative epidemiology* data on the causes of injuries is collected and used to create countermeasures. Additionally, Leveson states that the epidemiological model assumes that accidents have common factors that can be determined by statistical evaluation similar to epidemiology handling diseases.

AcciMap [BHN09] is an accident model categorized as epidemiological by Wienen et al. [WBV+17]. However, other researchers categorize it as systemic [YRL19]. Another model that can be categorized as epidemiological [WBV+17] is the Swiss cheese model developed by James Reason in 1990 [Rea90]. Again, this categorization is not accepted by everyone. Leveson argues that it is a sequential model [Lev21]. The *Swiss cheese model* states that accidents are the result of failures in four stages as seen in Figure 4.2: organizational influences, unsafe supervision, preconditions for unsafe acts, and unsafe acts. These stages are represented by slices of Swiss cheese, hence the name of the model. The holes in a slice, which can vary in size and position, represent weaknesses or more specifically failed or absent defenses in this stage [Hol10]. An accident happens when the holes of all slices align. When using the Swiss cheese model to analyze the cause of an accident, the analysis starts from the accident and traces it backwards. The analysis searches for active failures, which are unsafe acts committed by people, and for latent conditions, which arise from decisions made by designers and other

people. A similarity between the Swiss cheese model and the domino model is the linear chain of events structure. However, the Swiss cheese model also considers random behavior of components and independence between the failures in the different layers [Tho13].

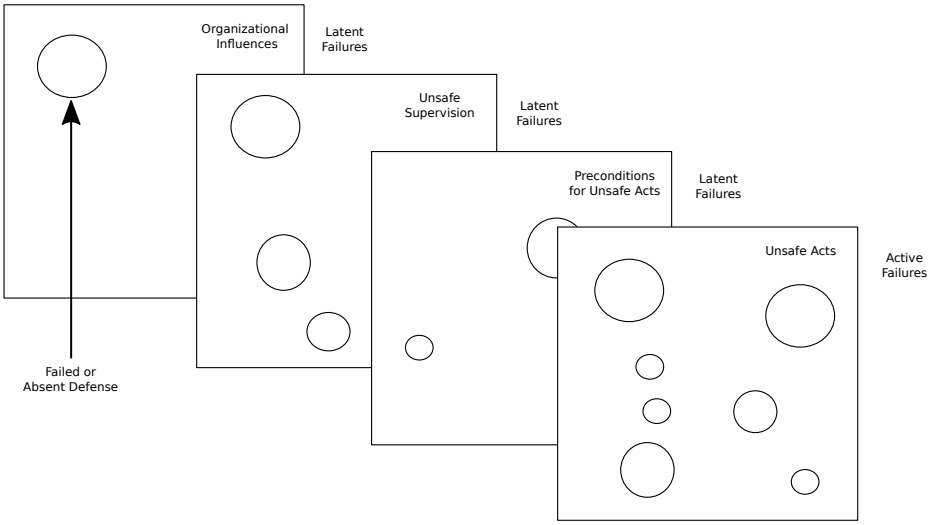


Figure 4.2. The Swiss cheese model of Reason as illustrated by Leveson [Lev21].

One risk analysis technique that is based on an epidemiological model is the Cognitive Reliability and Error Analysis Method (CREAM) [Hol10] (Section 4.2.4).

4.1.3 Systemic Model

Systemic models, also called *systems theory*, consider not only linear chains of events but also multiple independent causes resulting in an accident [Lev21]. It is based on two principles [Lev16]:

emergence and hierachy A complex system can be divided hierarchically into several organization levels. Each level has emergent properties, which do not exist on lower levels. While reliability is a component property, safety is viewed here as such an emergent property. This means that the safety of a system can only be analyzed by inspecting the whole system and not every component individually. A component can be safe in a certain environment and in another it is not, which is why the safety cannot be determined by only looking at the component.

communication and control Instead of trying to prevent failures in order to prevent accidents, the focus lies on enforcing constraints. Each hierarchical level enforces constraints on the level underneath it by sending control actions. Communication between the system components is necessary to share information of the environment. Additionally, a controller must acquire the status of the controlled system in order to know which control action must be send. This information can be provided through communication by sensors. Moreover, a

controller must have a goal, must be able to affect the state of the system component, and must contain a model of this component in order to be able to control the component as desired. This leads to a typical control loop as shown in Figure 4.3.

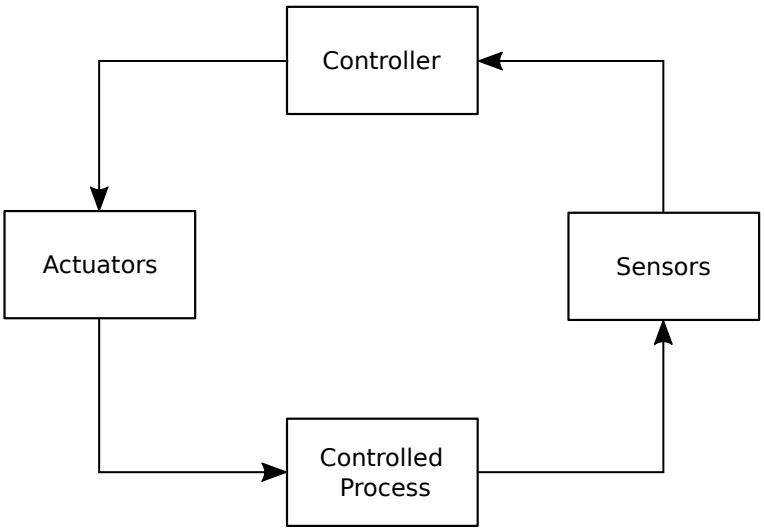


Figure 4.3. A standard control loop for systemic models [Lev16].

System theory views safety as an emergent property that arises when the components interact with the environment [Lev16]. In order to ensure safety, a set of constraints is enforced on the system components. Accidents arise due to interactions that violate those constraints. In this way, component failures as well as interactions that lead to accidents can be considered. All in all, safety is now a control problem and not a reliability problem anymore.

The two most known accident models that are based on systems theory [WBV+17; YRL19] are STAMP (Section 4.1.3) and the FRAM (Section 4.1.3).

STAMP

As already mentioned, System-Theoretic Accident Model and Processes (STAMP) is based on systems theory [Lev16]. Hence, the focus lies on enforcing behavioral safety constraints instead of preventing failures. Safety is handled as a control problem instead of a reliability problem. The goal of the control is to enforce the constraints.

Besides the basic concepts of system theory, STAMP is based on three concepts [Lev16]:

safety constraints In order to ensure safety, system-level constraints must be identified and responsibilities for enforcing them must be assigned.

hierarchical control structure Between the hierarchical levels of the system, control processes take place enforcing the safety constraints on the lower level. Communication between the levels is needed in order to execute the control processes. From top to bottom control

commands must be sendable to enforce the safety constraints and from bottom to top feedback about whether constraints are fulfilled must be sendable. Accidents occur if the control is inadequate. This can happen because of missing constraints, inadequate control commands, incorrect execution of a command, or flawed feedback.

process models As mentioned in the previous section a controller needs certain elements in order to work properly. The goal in this case is enforcing the safety constraints. The ability to affect the state of the controlled process and the acquisition of the system state are provided by the control commands and feedback. The last element that is left is the process model.

The model should contain the relationship between the system variables, their current value (state of the system), and possible actions to change the state. Based on this, an appropriate control action is chosen. The feedback received by the controlled system updates the process model. Mostly, accidents occur because the process model does not match the actual system state leading to four possible scenarios:

1. Incorrect or unsafe control commands are given
2. Required control commands are not provided
3. Potentially correct control commands are provided at the wrong time
4. Control is stopped too soon or applied too long

Just as in systems theory, accidents occur due to flawed processes violating the constraints [Lev16]. The causes of accidents can be understood by identifying the violated constraints and determining the reason for the failing control.

According to Leveson, STAMP can find more failures than component failures and the analysis is more sophisticated [Lev16]. Additionally, it can be used for implementing a safety-guided-design, meaning that the safety analysis guides the design of the system instead of first designing the system and finding its flaws afterwards.

The two most used risk analysis techniques based on STAMP are STPA (Section 2.1) and CAST (Section 4.2.3), both developed by Leveson [LT18]. STPA is a proactive method, meaning that during the development potential causes of accidents are analyzed, whereas CAST is retroactive, meaning that it is used to analyze accidents that already happened and identifies their causes.

FRAM

Hollnagel introduced the Functional Resonance Accident Model (FRAM) in 2004. It is a systemic model that describes non-linear relationships and interactions between functions of the inspected system and their variability [MN21]. FRAM is based on four principles [HHC14]:

equivalence The explanations for different kinds of accidents/consequences can be the same.

approximate adjustments People adjust what they do in order to match the actions with the conditions.

emergence Not all results can be explained by a specific cause.

resonance The cause-effect principle is not suited for all cases. Non-linear interactions and outcomes are explained with functional resonance.

When using FRAM [HHC14] at first, a model of a process must be created describing a typical situation, not a specific one. The model is based on the functions that must take place in order to execute the process. These functions are defined by six aspects: input, output, requirements, resources, control, and time. The functions are visualized with hexagons as shown in Figure 4.4. If two functions have the same value for an aspect, for example the first function as output and the second function as input, there is a potential connection, called *coupling*. Moreover, functions can be divided into foreground and background functions. This categorization depends on the importance of a function in the model. A function is categorized as *foreground* if its variability influences the outcome of the process being inspected, whereas *background* functions are assumed to not vary during the process.

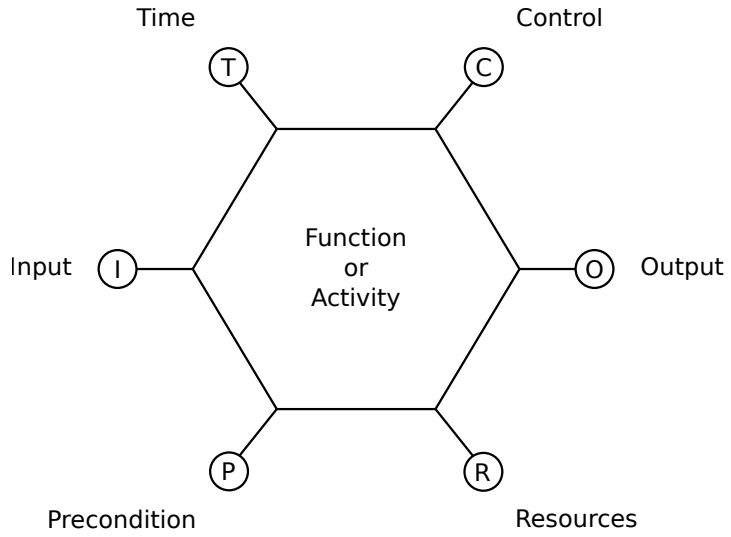


Figure 4.4. The graphical representation of a function in FRAM [HHC14].

Variability is divided into potential and actual variability. *Potential variability* describes what can happen under certain conditions and *actual variability* describes what really happens in a certain situation Actual variability is also called *instantiated model*, which contains specific couplings for a specific situation. For an instantiated model, the coupling and resonance can be analyzed. The analysis takes place by going through the potential couplings between the functions. During this step a function can be categorized as *upstream*, meaning it already was analyzed or is currently inspected, or *downstream*, meaning it follows the one currently inspected. When inspecting a function, the focus lies on the variability of the output. There are three main reasons why an output may vary:

internal/endogenous variability The function itself varies.

external/exogenous variablity The environment in which the function is executed varies.

functional upstream-downstream coupling The output from an upstream function varies and is the input, requirement, resource, control, or time of the inspected function. These couplings are the basis of functional resonance.

Based on the source of variability, the way they manifest is inspected, called the *phenomenology of variability*. Possible phenotypes are listed by Woltjer and Hollnagel [WH08]. Subsequently, the functional resonance is defined. In the last step, required performance monitoring is specified and barriers for variability are identified [WH08]. These should prevent unwanted events from occurring or protect them from the consequences of such events.

FRAM is used as accident analysis and risk assessment in different sectors such as rail transport and aviation [YRL19].

4.1.4 Comparison

The sequential models worked well for the simple systems of the past. However, significant changes occurred over time concerning the types of systems that are built and the environment in which they are built [Lev16]. The assumptions underlying these models are not accurate anymore. Accidents that involve no component failure at all are missed and complex couplings and interactions among components cannot be grasped. According to Leveson, system theory provides a much better foundation for safety engineering, is much more powerful, and more effective. STAMP methods provide a better understanding of accidents and their causes. However, they do not assign blame, which is why they will not be useful in law suits. Another disadvantage of systemic models is that they are more resource-consuming [ADP19]. Yousefi et al. state as one major difference between systemic models and the other ones that the latter use cause-effect chains of events, whereas systemic models view an accident as a complex network of events [YRL19].

Wienen et al. [WBV+17] compared analysis methods that are based on different accident models. They found that the presented models have a common approach:

1. Identify the events that lead to the accident
2. Link the events to describe the history of the accident
3. Identify conditions that triggered the events (only in epidemiological and systemic models)
4. Identify components, feedback, and control (only in systemic methods)
5. Analyze how the accident could have been prevented
6. Conclude and state improvements

Furthermore, the authors compare the models along two axes: type of coupling and contextual awareness (Figure 4.5). Sequential models are suited for loose coupled systems and do not consider the socio-technical context. They can be used to quickly find the cause of an accident and can be easier to understand. However, they do not find all possible causes for accidents because they will miss the structural causes present in socio-technical systems. The main benefit of epidemiological models is that they are aware of the socio-technical context. That is why they can find causes of accidents in the company culture, safety procedures, etc. that the sequential ones can not. Additionally, the concept of barriers helps to identify measures to prevent the identified causes. The disadvantages are that they take more time since the scope of the analysis is larger and convincing the management to accept the causes can take time because managerial shortcomings may be one of them. Systemic models are more suited for complex, tightly coupled systems but they take more effort and time to apply since they contain a deeper analysis of the processes and the organization. According to Wienen et al., this extra effort is not justifiable in many cases, especially if the consequences of accidents are minor, which is why they are too expensive to be employed in regular businesses. In order to support this statement they note some applications of STAMP done by Leveson, which considered accidents with high impact for government organizations. Wienen et al. conclude that sequential methods can be suited for the resolution phase of an accident, while epidemiological methods should be used to achieve a deeper understanding and finding latent factors. The effort and costs for systemic methods are not justifiable by its results.

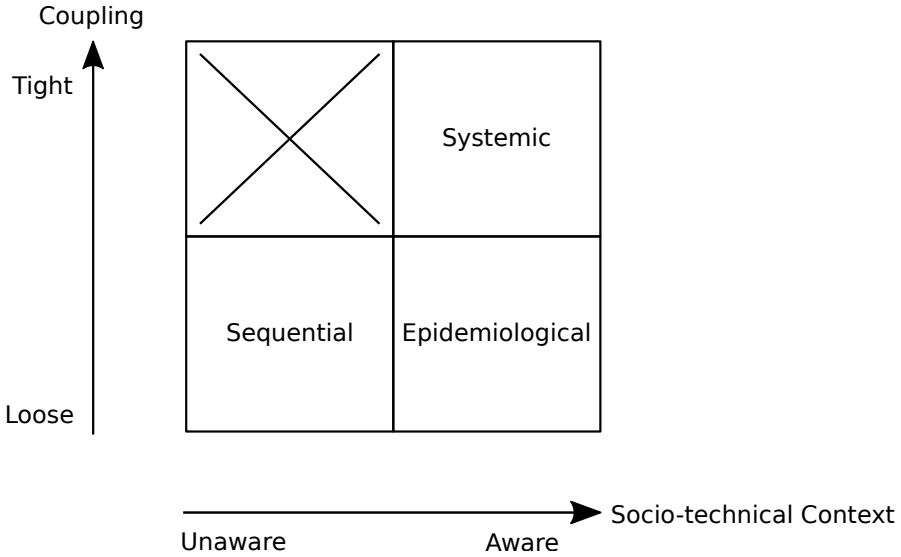


Figure 4.5. Comparison of the accident models based on two axes as shown by Wienen et al. [WBV+17].

However, systemic approaches are the predominant method by researchers for analyzing accidents [UW12]. Underwood and Waterson identified 302 references in which STAMP is the most cited with 52.0% followed by FRAM with 19.9% [UW12]. According to them the reason the systemic models are not used in industry at the time the paper was published may be a

combination of limited validation, usability, analyst bias, and the implications of not finding an individual to blame for an accident. Additionally, analysts who are familiar with sequential models may have difficulties changing to systemic ones and the costs to train them may be unjustifiable. The authors propose improving the communication between the research and practitioner communities in order to reach a common understanding. A more detailed inspection of the research–practice gap is done by Underwood and Waterson [UW13].

Adriaensen et al. review and assess safety analysis methods and conclude that there is no one-size-fits-all solution [ADP19]. They propose that risk management should be an attempt to improve systems instead of an isolated method/solution. However, they state the process industry will benefit from the systemic models, which are still predominantly applied within an academic context.

A comparison of the systemic models FRAM, STAMP, and AcciMap is done by Yousefi et al. [YRL19]. They state that STAMP is more comprehensive in identifying recommendations because failures/inadequate controls, unsafe decisions/control actions, and process model flaws are already captured. Comparing STAMP and FRAM to AcciMap showed some shortages on information in the first ones. However, the recommendations identified by STAMP were comparable with the ones of AcciMap. Furthermore, STAMP provides a deeper understanding of the causes of accidents because of the analysis of the hierarchical control structure. Wienen et al. state as the main difference between FRAM and STAMP the paradigm for analyzing the environment [WBV+17]. While STAMP uses a control model containing sensors, actuators, controllers, and controlled processes, FRAM uses functions with several parameters that can interact leading to events. One similarity of them is that they consider tight couplings between functions/components in order to link different parts of the system. FRAM is not as popular as other models and may not be applicable to certain accidents [Tho13]. It is based on the assumption that the reaction of subsystems to nonrandom noise mimics the reaction of nonlinear components to random noise. However, this is not proven. Furthermore, it is not very suited for accidents caused by component failures or for cases without variability since the model is based on the component variability.

In conclusion, systemic models may be too expensive to be widely used in industry, but they may provide the best understanding and identification of causes for accidents. Just looking at systemic models, STAMP seems to be suited best for most cases and it is the most used one.

4.2 Analysis Techniques

The goal of risk analysis is to identify potential causes of accidents before they occur in order to eliminate or control them [Lev16]. As already mentioned, the analysis techniques are based on accident models presented in the previous chapter. The most widely used ones are based on the sequential model and were developed around fifty years ago [Lev16]: FTA (Section 4.2.1), FMEA (Section 4.2.2), and HAZOP. The presentation of FTA and FMEA in the first two sections of this chapter is followed by a roundup of several other techniques

such as HAZOP in Section 4.2.4. However, these traditional techniques have limitations since the systems nowadays are more complex and software-intensive [Lev16]. Two relatively new techniques based on STAMP, developed to overcome those limitations, are explained in Section 2.1 (STPA) and Section 4.2.3 (CAST). Afterwards, Section 4.2.5 presents combinations of techniques and Section 4.2.6 compares the different techniques.

4.2.1 FTA

Ruijters and Stoelinga provide an overview of the state-of-the-art in Fault Tree Analysis (FTA) [RS15]. The most basic fault trees are the standard/static fault trees, which were developed in the 1960s. A fault tree is a directed acyclic graph with two type of nodes: events and gates. Events are failures in subsystems, especially failures in individual components. If an event is triggered by one or more other events, it is called *intermediate event*, otherwise it is a *basic event*. The event that is analyzed — the accident — is the root and hence the top of the tree and is called the *top event*. Figure 4.6 shows the graphical representations of events. Basic events are represented by circles while intermediate events are represented by rectangles. In cases where the tree is too large for one page, triangles are used to broadcast events between different trees. Besides the intermediate and basic categorization, an event can also be undeveloped, represented by a diamond. This is the case if there is not enough information or it is assumed that it is not necessary to further divide it into subtrees. In order to connect the events, gates are used. There are some extensions that add further gates, for example the NOT gate, but the set for standard fault trees is defined as follows:

AND The output event occurs if all input events occur.

OR The output event occurs if any of the input events occur.

k/N a.k.a. VOTING This gate has N inputs and the output event occurs if at least k of the input events occur.

INHIBIT Additionally to the input event, the condition to the right of the gate must also occur in order for the output event to occur. This gate could be replaced by an AND gate with two inputs.

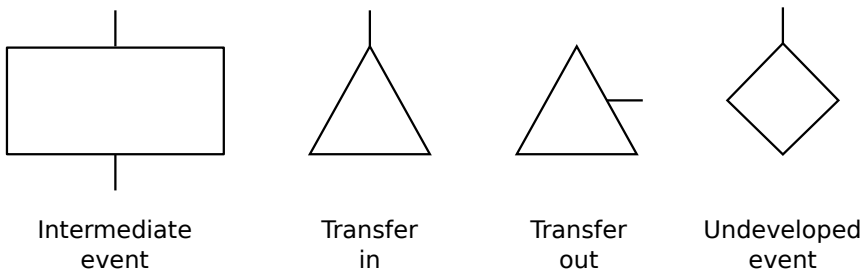


Figure 4.6. The graphical representations of events [RS15].

The graphical representations can be seen Figure 4.7. Creating the fault tree can be aided by software and several methodologies have been proposed [LGT+85].

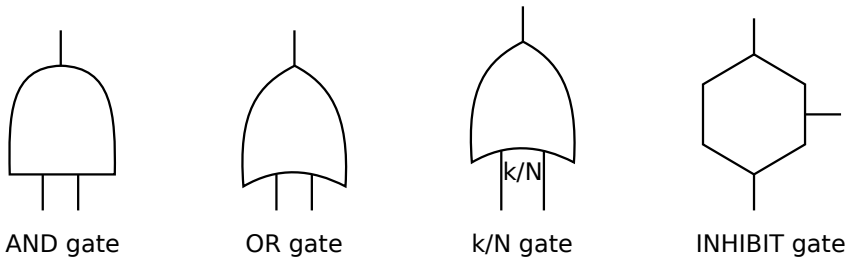


Figure 4.7. The graphical representations of gates [RS15].

The analysis of a fault tree can be done with different techniques, whereby the common ones are (minimal) cut set, (minimal) path sets, and common cause failures. If the reader is interested in these techniques, they are explained further by Ruijters and Stoelinga [RS15]. The used techniques can be divided into qualitative and quantitative ones. *Qualitative* ones are used to gain insight of the structure of the fault tree and to identify vulnerabilities, whereas *quantitative* ones derive numerical values. These values can be the error probability or the importance of a component for the reliability. Usually, a qualitative evaluation is done first, followed by a quantitative evaluation [LGT+85].

Some extensions are: dynamic fault trees, fault trees with fuzzy numbers, repairable fault trees, and more [RS15]. Another extension is Software FTA (SFTA), which is similar to the standard (hardware) FTA and uses a subset of the symbols [LH83]. In this way the trees of both methods can be connected, which is important since software must be viewed as a part of a bigger system instead of a separated part.

Mostly FTA is used as the first step of FMEA, which is explained next.

4.2.2 FMEA

Failure Modes and Effects Analysis (FMEA) is one of the earliest systematic techniques for risk analysis, developed in the 1950s [HVV+20]. Its accident model is the sequential one and hence it is based on reliability theory and considers safety as a component failure problem [SBF+19]. The general approach is as follows [Gil93]:

1. Identify the failure modes and their consequences.
2. Assess the chances that these faults occur.
3. Assess the chances that the faults are detected.
4. Assess the severity of the consequences.
5. Calculate a measure of risk.
6. Rank the faults based on their risks.

7. Perform actions to prevent the high-risk problems.

8. Check the effectiveness of the actions.

Mostly, it is used in the initial design phase of a project but it can be used in every stage [Ono97]. In the first step, the identified failure modes and their consequences are written down in a worksheet [HVV+20]. There are several different worksheets that can be used. Figure 4.8 shows the standard one. It has nine columns each specifying an important aspect connected to the failure mode. More examples are given by Onodera [Ono97]. Risk assessment is normally covered through risk priority numbers (RPNs), however, Gilchrist criticizes the usage of RPN and proposes an alternative approach [Gil93].

1	2	3	4	5	6	7	8	9
Equipment Name	Function	Failure Mode	Failure Causes	Failure Effects	Failure Detection	Failure Probability	Criticality Level	Remarks

Figure 4.8. The standard FMEA worksheet [Ono97]

An extension is Failure Modes, Effects and Criticality Analysis (FMECA) in which the failure modes are ranked based on their severity [SBF+19]. So instead of RPN, Criticality is used. However, some authors do not distinguish between FMEA and FMECA, for example Gilchrist [Gil93]. Another extension is the probabilistic FMEA in which the possibility that a component failure occurs is used. Analogous to FTA, there exists a software version of this method: Software FMEA (SFMEA) [SBF+19]. It extends FMEA by additionally considering software intensive components, for example embedded systems. Furthermore, it considers that software itself can not fail but incorrect behavior should be prevented. Details about this method are explained by Goddard [God93].

All in all, the method provides a foundation for qualitative reliability, maintainability, safety, and logistics analyses [HVV+20; Ono97]. FMEA has been highly successful [Gil93] for example Arabian-Hoseynabadi et al. apply it successfully to wind turbines [AOT10]. Furthermore, in software reliability analysis, SFMEA is one of the most used methods [HL12]. For example Lutz and Woodhouse successfully applied it to spacecraft software and state that a combination with FTA has found even more requirement issues [LW96].

4.2.3 CAST

Another analysis method that is based on STAMP is the Causal Analysis based on System Theory (CAST). It can be used during an accident investigation in order to determine why the accident occurred [Lev19]. One goal of the technique was to shift the focus to understanding why an accident occurred and preventing similar losses instead of stopping the investigation as soon as someone to blame was found [Lev16]. With the help of CAST the entire sociotechnical system can be examined and its weaknesses in the safety control structure can be found. The goal is to learn as much from an accident as possible instead of identifying one root-cause.

Similar to STPA the loss examined can be anything and does not have to be a typical safety or security accident [Lev19]. Additionally, safety is viewed as a control problem too. In contrast to STPA, CAST only assists in examining the particular scenario that occurred. However, applying CAST to past accidents can help in the STPA analysis by identifying loss scenarios that should be prevented or controlled.

The CAST analysis consists of five steps [Lev19]:

Assemble Basic Information In this step basic information about what happened is gathered. This entails system-level hazards that occurred and violated constraints.

Model Safety Control Structure Since safety is viewed as a control problem, the cause of a hazard is that the control structure and controls were not effective. In order to determine why and how the structure might be improved, it must be modeled first.

Analyze Each Component in Loss Starting from the bottom of the control structure, each component is analyzed. Thereby, the role it played in the accident and the explanation for its behavior is determined.

Identify Control Structure Flaws In contrast to the previous step, now the whole control structure is considered in order to identify the general systemic factors that contributed to the loss.

Create Improvement Program The last step is to create recommendations for improving the control structure in order to prevent further losses.

4.2.4 Other Techniques

There exist several more risk analysis techniques. Some of them are presented in this section.

One of the more widely used techniques is Hazard and Operability Study (HAZOP). In contrast to FTA and FMEA, it is a preliminary analysis technique [KRC15]. McKelvey points out some key problem areas of HAZOP: lack of experience of the analysts, failure to communicate, management short-comings, complacency and poor loss-prevention practices, shortage of technical information, and analysts are only human [McK88]. Moreover, he presents solutions that could counter these difficulties. Khamidi et al. successfully applied HAZOP to a mobile mooring system [KRC15]. The process consists of eight steps [McK88]:

1. Define the scope of the study
2. Select the analysis team
3. Gather information necessary for the study
4. Review the normal functioning of the process
5. Subdivide the process into logical and manageable subunits

6. Conduct a systematic review according to the established rules for the procedure being used

7. Document the review proceedings

8. Ensure that all recommendations from the study are adequately addressed

Sneak Circuit Analysis (SCA) was introduced in the 1960s to improve system reliability and is an effective extension for the traditional techniques such as FMEA and FTA [ZZ14]. It is a technique for evaluating electrical circuits. The aim is to uncover latent (sneak) circuits and conditions causing unwanted behavior[1]. The process consists of six steps [ZZ14]: data collection, system classification, data input, topology simplification, network tree development, and analysis. Disadvantageous are the large amount of effort, resources, and time needed to perform the technique [SBF+16].

Besides safety risk analysis, there also exist security risk analysis techniques such as CORAS, which consists of eight steps [LSS11]. The first four ones aim to understand the goal of the analysis and for example list the assumptions about the environment. The remaining steps are the actual analysis, containing the identification of concrete risks, and potential solutions.

In Deductive Cause-Consequence Analysis (DCCA) informal explanations are replaced by mathematical proofs [ORS05]. It is a formal generalization of FTA and FMEA. Ortmeier et al. apply this technique successfully to the Elbe Tunnel.

Event Tree Analysis (ETA) is a bottom-up, deductive, system safety analytical technique [Cle90]. The basic principle is to start with an initial event and generate a tree based on all possible permutations of the other events. Hence, each hierarchy level consists of one event represented by two nodes: one node presenting the event failed and one node representing the event was successful. For each path the probability that it happens is calculated. This technique can be used to complement for example FTA or FMEA. Furthermore, there exists an extension with fuzzy numbers in order to improve the risk assessment: Fuzzy FTA (FETA) [HCW01].

CTTA Risk Analysis and Management Methodology (CRAMM) is a qualitative risk analysis and management tool, developed in 1985 [Yaz02]. It identifies threats and vulnerabilities, which are visualized using Bayesian Networks [MZP+06]. They consist of a set of variables and a set of directed edges between them and are useful in representing possible cause-effect relationships. In order to identify and prioritize critical events, *What-if* studies are used.

One epidemiological technique is CREAM according to Wienen et al. [WBV+17]. It is an accident investigation technique based on a clear distinction between what can be observed, called phenotypes, and what must be inferred, called genotypes [Hol10]. For the genotypes exist three categories: individual, technological, and organizational.

[1] https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/risk_management/ss_handbook/media/Chap9_1200.pdf

4.2.5 Combinations

As already mentioned, some analysis techniques complement each other. This was used to create new techniques that are a combination of already existing ones. For example FRAM & System Hazard Analysis (SHA) [FM14] or SCA & FMEA [SBB93]. A small subset of them is presented in this section.

He and Li combine SFTA and SFMEA in order to create a new reliability analysis technique [HL12]. Practical applications have shown that this model can improve the reliability and ensure quality of software. Applying each of the techniques alone has drawbacks, which are eliminated by combining them. It starts with adopting SFTA to build a software analysis tree. Afterwards, SFMEA is used to analyze the basic events and determine their severity, fault reason, fault model, and fault effects. Subsequently, faulty blocks are located and improvement measures proposed.

Toda et al. combine STPA with FRAM [TMT18]. They transferred the four categories of UCAs to each aspect of each function in the FRAM model. The analysis results for each combination of an aspect of the function and a keyword include four descriptions: deviation, local influence, global influence, and severity. In a case study the authors found out that the new technique detected additional hazards compared to STPA. However, hazards concerning individual components of the system were overlooked.

Antoine showed that STPA is complementary to traditional techniques such as FTA [Ant13]. He states that FMEA and traditional techniques such as HAZOP and FTA could be used after an STPA analysis is finished in order to examine the identified hardware issues further. According to him, FTA would be the best candidate since it starts with a hazard event while the two other ones start with a system component. Additionally, it is a top-down approach just like STPA.

4.2.6 Comparison

As seen in the previous sections, there are many different techniques that could be used when developing a system. In order to select the best fitting one for a system, many comparisons were made between the techniques. Some of them are presented in this section.

According to Leveson [LT18], STPA has several advantages over the traditional techniques. While the traditional techniques aim to analyze simpler systems and mostly after the development is finished, STPA can analyze very complex systems and can already be applied in early development phases. Additionally, it includes software as well as human operators in the analysis. STPA finds all the causal scenarios found by the traditional techniques but also many more that were not found by the traditional ones. Leveson [Lev16] states that this statement is confirmed by the few comparisons that have been made. Furthermore, STPA needs less time and fewer resources.

A comparison of STPA to FMEA is done by Sulaman et al. [SBF+19]. They state as the major difference between STPA and any other bottom-up technique that in STPA the analyst can stop refining the causes at the point were an effective prevention or weakening can be identified. In their comparison with FMEA, they found out that STPA identified more software hazards

than FMEA and outperforms it in finding causal factors of identified hazards. Furthermore, causes identified by STPA are more detailed thanks to the keywords it provides as guidelines. However, FMEA identified more component failure hazards and is better in risk assessment of software failures. This is caused by the different focuses of the techniques. FMEA focuses on the architecture and complexity of the components while STPA focuses in finding causal factors for identified hazards. The authors conclude that the techniques complement each other well in their study.

Song [Son12] also compares STPA with FMEA by applying STPA to the Darlignton shutdown system, which is original done with FMEA. Moreover, he explains why FRAM is not suitable for this system. FRAM aims to find functional resonance and hence is suitable for systems that can be easily divided into several functions. As a second reason, he claims that characterizing the potential for variability is subjective. Just as other researchers, as an advantage of STPA is stated that it provides more guidance. FMEA has no guidelines for identifying hazards, failure modes, and failure causes. Although STPA identified more hazards and causal factors, Song states that these additional factors were not significant.

A comparison to FTA is done by Ishimatsu et al. [ILT+10] by applying STPA to the Japan Aerospace Exploration Agency (JAXA) H-IIB Transfer vehicle (HTV). Overall, they state the same advantages of STPA over FTA as it has compared to FMEA.

Another comparison is done with STPA and Functional Hazard Analysis (FHA) by Zikrullah et al. [ZKM+21] in a case-study. They state that researchers already found several weaknesses of FHA and STPA. For example STPA has the disadvantage that due to its attempt to increase the hazard coverage, the number of UCAs can explode. FHA captures a higher number of Hazardous events (HEs) than STPA captures UCAs because of the use of keywords for hazard identification, the function type classification in the selected model, and the modeling approach. In contrast to that, the number of identified loss scenarios is higher than the causal scenarios found by FHA. Each method identified unique hazards by using keywords the other method does not use. Nevertheless, the identified consequences and losses are the same by both methods. They conclude that both methods are suitable for analyzing novel technology. However, STPA is the better choice since the modeling technique captures all systemic properties, and the safety requirements are already assigned to an agent. In FHA an additional process is required in order to identify the agent for a requirement. The authors also mention that Leveson [LWF+14] found out that STPA could identify causal scenarios that FHA could not but according to Zikrullah both methods still suffer limitation for identifying scenarios caused by component failure or software error.

Google applied STPA as part of Site Reliability Engineering[2]. One of the results is that more solutions were found by STPA, which mostly are low cost and highly effective[3]. Moreover, the creation of the control structure lead to gaining insights regarding the control loop. Another advantage of STPA is that the results are generalizable and hence independent of the implementation. The presenters state that thanks to STPA potential accidents can be detected

[2] http://psas.scripts.mit.edu/home/2021-stamp-workshop-program/

[3] http://psas.scripts.mit.edu/home/wp-content/uploads/2021/06/2021-06-23-1210__Falzone_Thomas.pdf

beforehand and at much lower cost. Additionally, more recommendations per incident were generated. As conclusion the presenters share their insights about STPA in which they state that one of the major values of STPA for the analysts was the identification of system safety requirements.

4.3 STPA Use Cases

Despite the claim of some researchers that STPA is not useful in industry, at the STAMP workshop 2021 many organizations reported that they have been using STAMP/STPA[4]. Among these are Google, Amazon, Airbus DS, Intel Corp, NASA, Samsung, etc. There are several more case-studies using STPA in which STPA is also compared to other analysis techniques. A subset of them is presented here, showing the usability of STPA in several different domains.

JAXA develops various types of space systems in which safety is essential [ILT+10]. It has not experienced accidents that were caused by other factors than component failure but still considers using STPA in order to prevent future accidents. The HTV, which was originally developed using FTA, was chosen as an analysis target in a pilot case study. HTV is an unmanned vehicle with the task to carry necessary components and commodities to the International Space Station (ISS). The authors conclude that the causal factors identified by STPA cover all hazard causes of the original fault tree and that there were causal factors that were only identified by STPA. A second case study targeting HTV shows an application of STPA to identify possible unsafe interactions among multiple controllers [ILT+14]. These can arise due to conflicting or uncoordinated control actions. This application possibility is an advantage in comparison with FTA since multiple controller problem can not be captured by it. Another vehicle that is the target of a case study is the Crew return Vehicle (CV), which is a manned vehicle carrying crews and necessary components to the ISS [NKM+11]. The case study consisted of a safety analysis using STPA in parallel with mission design as a trial of safety guided design. It is important to perform system and safety design in parallel to design a safe system. The authors present their idea of a safety guided design process, which is applied to CV in this case study.

Baumgart et al. [BFP18] apply STPA to a case in the construction equipment domain: the quarry site case. Identifying hazards and deriving constraints were straight forward. However, developing the control structure toke time due to finding an appropriate abstraction level. The authors state that processes and guidelines for handling complex system of systems (SoS) using the control structure must be developed. They conclude that in the context of SoS, STPA has inadequacies and an improved technique must be developed for safety analysis of SoS. However, Leveson argues that there should be no difference between a complex system and a SoS[5]. The properties of an SoS are already contained in the system definition as defined in System Theory. Leveson shows that STPA can be used for SoS without changing anything. As

[4] http://psas.scripts.mit.edu/home/2021-stamp-workshop-information/#

[5] http://sunnyday.mit.edu/SOS-hazard-analysis.pdf

an example she uses a real defense system. She concludes that the new term SoS is unnecessary and misleading, since a system hazard analysis and a SoS hazard analysis are the same.

Even the European Parliament promoted 2008 a system-based approach for the safety management of road infrastructure, and the individual members are responsible for the realization [KK21]. Although a framework is provided, the details are not determined. Kraut and Koglbauer propose in their study a systemic approach for the road safety management process in Austria based on STPA. The application of the approach lead to improvement in road safety and uncovered a number of weaknesses in the safety management of the Austrian road network. Hence, the method and the results can be applied worldwide to road networks in order to increase the traffic safety.

STPA can also be applied to serviceability issues as shown by Slominski in her master thesis [Slo20]. As already mentioned in Section 2.1, the broad definition of a loss allows the usage of STPA for several emergent properties, not just safety. The technique can easily be adapted for serviceability by identifying serviceability losses. Hence, STPA can be used to understand system interactions and strengthen the service control structures. Case studies are used to demonstrate the application of STPA to serviceability. Each study identified design recommendations for serviceability as well as for the service control structure. The author concludes that STPA can be used for addressing existing issues as well as for early design phases of future systems in the serviceability domain.

Dong used STPA for re-analyzing a High Speed Train accident in China in his master thesis [Don12]. Additionally, he applied the method to the Communication Based Train Control (CBTC) system, which is similar to the one analyzed first. The methods helped to identify more inadequate controls and required improvements. The author points out that the control structure is very helpful for understanding and analyzing the system by providing a better understanding of the interaction between the system elements. In conclusion, STPA can be very comprehensive in identifying all kinds of loss scenarios.

Another application of STPA is done by Abdulkhaleq and Wagner [AW13] who used it for analyzing a safety-critical system in the automotive domain. The aim is to analyze the application of STPA and the difficulties for a real system in a commercial vehicle. They conclude that STPA is suited for an in-depth analysis. It is a more powerful and useful technique that is usable in the automotive domain.

In order to show the advantages of STPA compared to traditional techniques, Abrecht and Leveson [AL16] applied STPA to Naval Offshore Supply Vessels (OSVs), which utilize dynamic positioning in support of target vessel escort operations. They conclude that STPA is effective and adheres the guidelines for hazard analysis in Department of Defense (DoD) systems. Horney [Hor17] also concludes in his master thesis that STPA can improve the safety management for DoD systems. He describes a technique in order to guide DoD engineers when applying STPA.

The suitability of STPA for identifying managerial risks is evaluated by Pope [Pop19]. He conducts a case study in which a manufacturing process is restarted after 30 years using raw material that has been in storage. The result is that STPA produces useful results and is

relatively easy to review. This is also supported by other hazard analysis experts to which the results were shown.

Furthermore, Leveson et al. show the usefulness of STPA for certifying aircraft [LWF+14]. They compared the new technique with the common approach, which included traditional analysis techniques, in order to prove their thesis that STPA is more powerful. The results are that with STPA potentially unsafe interactions can be easier identified but the method does not identify function reliability requirements. However, the authors conclude that the common approach omits important causes of aircraft accidents.

4.4 STPA Extensions

The main purpose of STPA is ensuring safety. Nevertheless, it can be used for several properties as already mentioned before. For some properties, just the losses must be defined accordingly. However, for other properties extensions were developed. One of them is STPA-Sec. It is used to secure software against intentional disruptions and hence aims to provide more security [YL13]. Instead of focusing on threats as the cause for losses, the broader system structure is analyzed in order to identify vulnerable states that are exploited by a threat leading to a loss. Vulnerabilities are controlled by identifying and enforcing constraints on unsecure control actions that can lead to these vulnerable states. Just like hazards lead to safety incidents in STPA, vulnerabilities lead to security incidents. The main advantage of STPA-Sec is that the focus lies on controlling vulnerabilities instead of trying to avoid threats like it is usually done because these threats are mostly beyond the control of the security specialist. Applying STPA-Sec consists of the same four basic steps as STPA:

1. Establish the foundation for the security analysis by identifying the losses that should be prevented and defining the vulnerabilities that lead to the losses under worst-case environmental conditions. Furthermore, a High Level Control Structure (HLCS) is modeled, providing a graphical specification of the functional controls

2. Identify unsecure control actions that are vulnerable under certain conditions. The unsecure control actions are divided into four types: Providing a control action leads to a hazard or exploits the vulnerability, not providing a control action leads to a hazard or exploits a vulnerability, providing control actions too late, too early, or in the wrong order leads to a hazard or exploits a vulnerability, and stopping a control action too soon or continuing it too long leads to a hazard or exploits a vulnerability.

3. Create security requirements and constraints based on the identified unsecure control actions.

4. Identify causal scenarios that violate the safety constraints by using the guide words STPA-Sec provides. The scenarios can be used to create protection against the occurrence of them or at least limit the damage from them.

Shapiro [Sha16] proposes additional concepts to STPA-Sec to produce STPA-Priv. It addresses privacy in two ways: defining losses and capturing the control structure. The author hopes that the STPA-Priv is as successful for privacy as STPA has proven to be successful for identifying safety risks. The method consists of four steps: Identify potential adverse privacy consequences, identify vulnerabilities that can lead to those consequences, create privacy constraints and functional control structure, and identify privacy-compromising control actions.

A unified approach to safety and security analysis based on STPA and STPA-Sec is developed by Friedberg et al. [FMS+17] and is called STPA-SafeSec. The guidance for identifying causal factors for safety is extended to also include the security domain. Moreover, the method allows linking the control structure to the physical system design. This allows to complement STPA-SafeSec with traditional security analysis techniques. Additionally, some shortcomings of STPA and STPA-Sec are taken care of in this new method.

STPA-DFSec is a data-flow based STPA-Sec developed by Yu et al. [YWL21]. The main difference to STPA-Sec is that the guide words were adjusted and a Functional Interaction Structure (FIS) is used instead of a control structure. An advantage of the new method is that information-related problems can be identified more directly and it can be used together with other STPA-based approaches. However, two limitations were identified: lack of evaluation of identified scenarios and missing information about the data processing of the target system. Concluding, the authors state that STPA-DFSec can reveal more details in information security aspects, not directly addressed by STPA-Sec. However, the new method should not replace the original one but complement it.

France [Fra17] proposes the method STPA-Engineering for Humans in her master thesis. Its goal is to guide the analyst in the identification of causal scenarios related to human operator behavior. The method was originally proposed by John Thomas in order to handle the complexity of human automation interactions. In contrast to other extensions, this one creates a new model with focus on improving the characterization of the operator's model instead of modifying an existing model. This new human controller model can easily be applied as part of the original STPA process. It consists of three components as shown in Figure 4.9:

Control Action Selection This component contains the goals of the controller and how decisions are made.

Mental Models Beliefs of the human about the system and the environment are contained here. It is divided into process state, process behavior, and environment

Mental Model Updates Human experiences and expectations influence the processing of sensory input. This influence is captured in this component.

The model's goal is to provide a better understanding of how UCAs may arise. The complete analysis is the same as in STPA except the identification of loss scenarios. In this step the new introduced model should be used for UCAs performed by a human controller

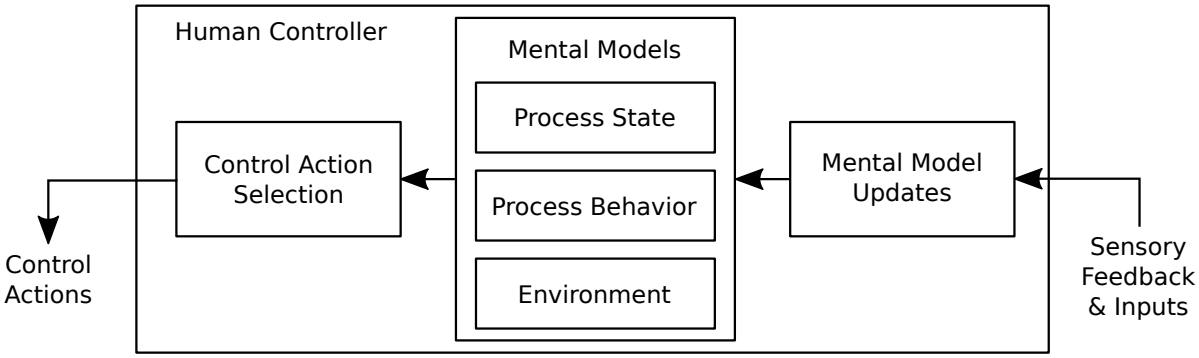


Figure 4.9. The human controller defined by France [Fra17].

in order to identify richer scenarios, which will lead to additional requirements. The author applied the extension to an Automated Park Assist system and concludes that it was feasible and valuable.

Another extension of the human-controller in STPA is done by Thornberry in his master thesis [Tho14]. Originally, the human controller just consists of the control action generation, a model of automation, and a model of the controlled process. In the extension the different components of the controller are the following: detection and interpretation, model of automated process, model of controlled process, decision-making, and affordance. Additionally, Thornberry extends the causal factor analysis, and hence the identification of scenarios, by proposing five categories for them: Flawed feedback, flawed detection and interpretation of feedback, inconsistent process models, flawed decision-making, and inappropriate affordance. The goal of these categories is to make conflicts between feedback more easily identifiable and help the analyst to better identify loss scenarios related to a human operator.

Based on Thornberry's extension, Montes developed in his dissertation STPA-RC [Mon16]. The goal is to capture the important attributes of previous research regarding incorporating human behavior and address existing research gaps. STPA-RC is build on Thornberry's extension and adds new parts. Moreover, it refines his original guidance and introduces a method for identifying outside influences on the controller. The categories for causal scenarios introduced by Thornberry are refined with more details.

4.5 STPA Improvements

STPA may be better than other analysis techniques but it still can be improved. Zou wrote a master thesis with the goal to bring new insights and suggestions for the practical application of STPA [Zou18]. Therefore, he used a case study in which the operation of a fully autonomous vessel was analyzed. The first improvement he found concerns the UCAs. FMEA and Control HAZOP (CHAZOP) use guide words that can be redefined for the UCAs in STPA for example *intermittent*, *unintentional*, or *unintended*. Testing the modified UCAs approach identified four UCAs that were not found before. The second improvement concerns the identification of loss

scenarios. Originally, they are found by group brainstorming and hence, for an individual analysis it is unavoidable to omit scenarios. In order to improve this step a systematic analyzing framework is proposed that groups scenarios in categories: Communication, digital hardware, software, and mechanical item. For each category the corresponding scenarios are defined together with the causal factors and constraints. There is one big drawback: The modified approach cannot deal with the repetition and complexity of causal factors and safety constraints. Zou concludes that this problem must be addressed in order to reduce the analysis time and that the severity and priority of an UCA is still difficult to determine in contrast to FMEA and HAZOP.

Another improvement regarding the generation of scenario is proposed by John Thomas as stated by Summers in his master thesis [Sum18]. The improvement is a new method in which the scenarios are divided into four categories based on their location in the control structure as seen in Figure 4.10: command not followed or followed inadequately, inappropriate decision, inadequate feedback or other inputs, and inadequate process behavior. The goal of this categorization is to ensure more coverage across the control structure just as the categorization of the UCAs ensures each type is considered.

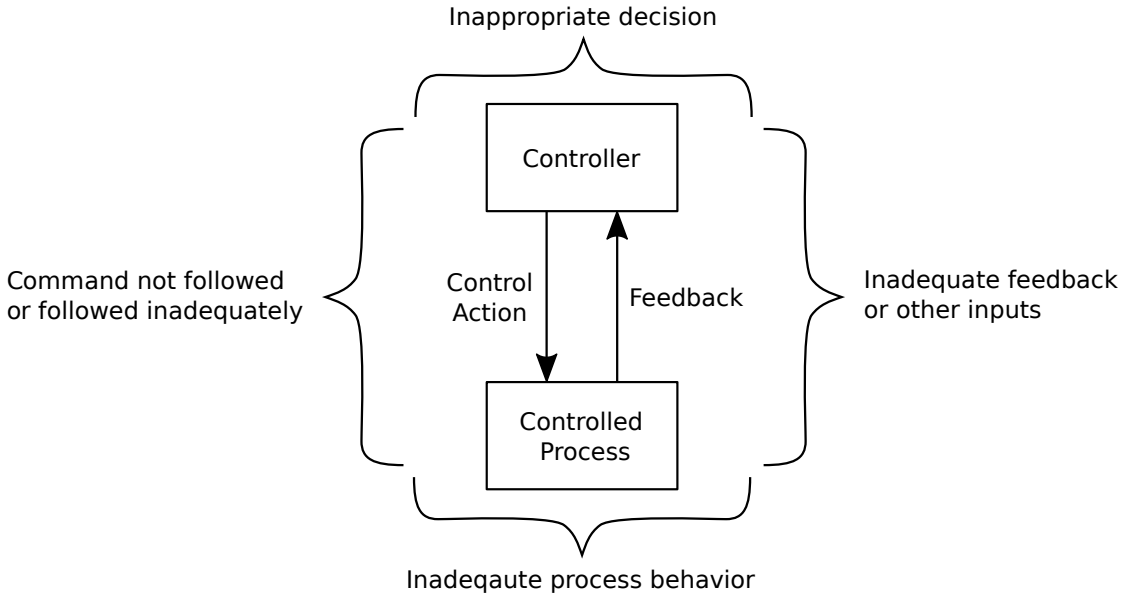


Figure 4.10. The location of the scenario types [Sum18].

Antoine also proposes an improvement for identification of loss scenarios in his dissertation [Ant13]. He wanted to eliminate the repetition of similar information caused by UCAs that share common causal factors. Therefore, a *Step 2 Tree* was constructed as seen in Figure 4.11 in order to highlight common pathways through which causal factors lead to UCAs. The tree starts with two general categories of hazardous scenarios: *Inadequate command generation* and *safe command not followed*. Each of them is attached to three lower-level categories. The former one to *inadequate goal/input*, *flawed control algorithm*, and *flawed process model*. The latter one to

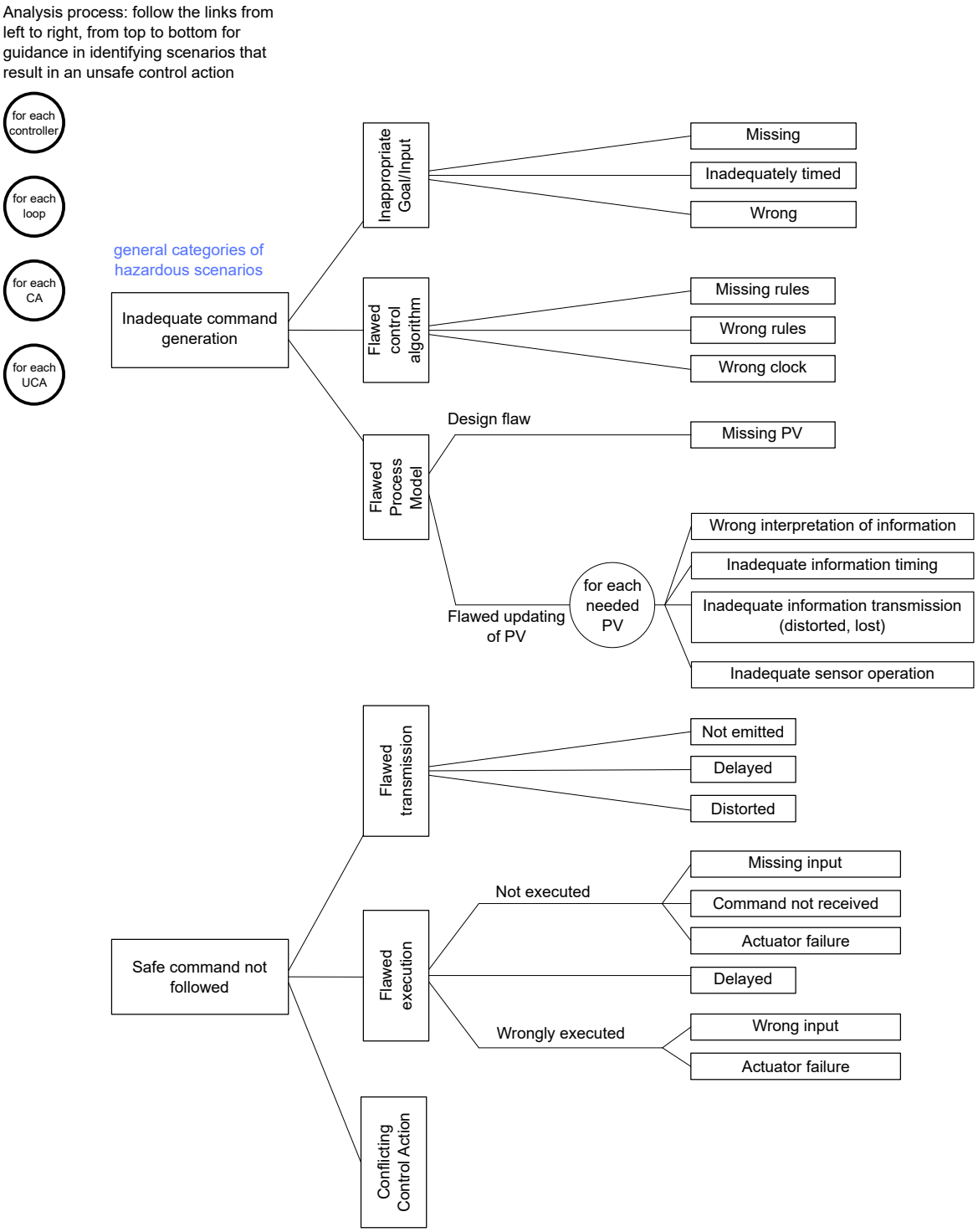


Figure 4.11. The Step 2 Tree [Ant13].

flawed transmission, *flawed execution*, and *conflicting control action*. In order to identify the loss scenarios for an UCA, the tree can be followed left to right, from top to bottom. The goal of the tree is to provide structure for the process of identifying loss scenarios by making the connections between the causal factors explicit. In three of five examples of the PROSCAN STPA project the Step 2 Tree was applied successfully.

Besides the tree, Antoine also organizes the results of the analysis in order to be able to visualize them. This can help to easier grasp the data and the effectiveness of elimination and mitigation strategies for hazards. The visualization Antoine presents can be seen in Figure 4.12. System protection measures are visualized along with the hazards, causal scenarios, and other aspects. Hazards are represented by circles, scenarios by hexagons, UCAs by rectangles, control actions by triangles, and controllers by diamonds. The protection measures are visualized by dots on the links between different elements in the visualization.

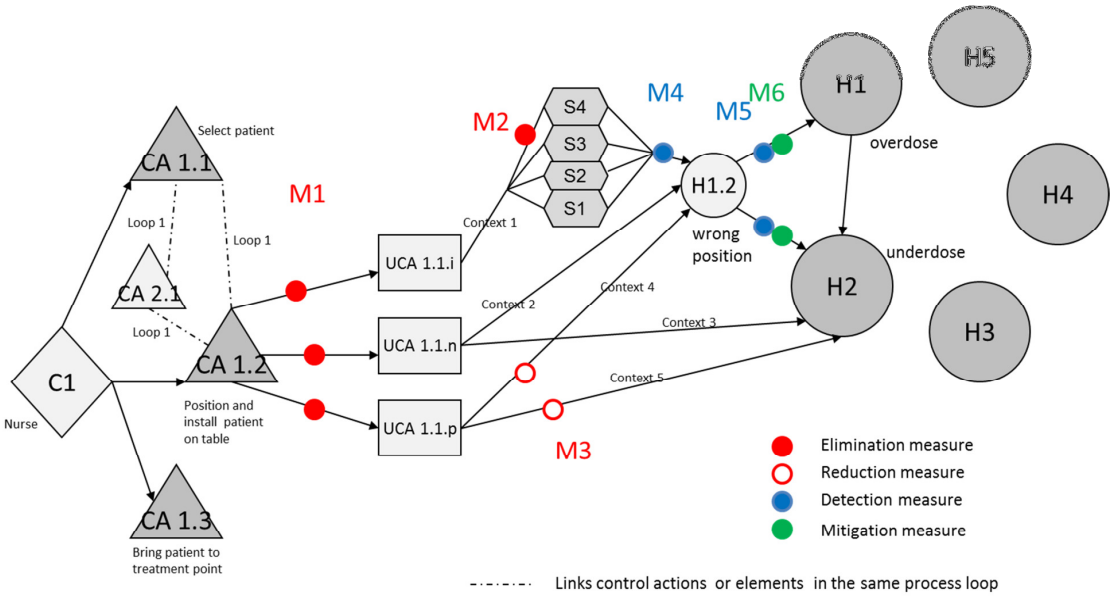


Figure 4.12. Visualization of system protection measures [Ant13].

Regarding tools to support the application of STPA, Antoine states that computer databases and searches can benefit from the data management of STPA. Moreover, software tools that assist in the analysis itself as well as enhance the readability of the analysis and visualize the results can be helpful.

4.6 Leading Indicators

After risk analysis is done for a system and necessary safety requirements are implemented, it can still happen that accidents occur. In order to identify the potential for an accident before it occurs, leading indicators for safety can be used [Lev15]. Particularly in the petrochemical

industry, a lot of effort has been spent on trying to identify such leading indicators [Lev14]. Early attempts to develop them began in the mid-1900's. In the past, the identification of leading indicators involved mostly finding a set of generally applicable signals that presage an accident [Lev15]. For example the quality of maintenance and inspection or equipment failure rates are identified as leading indicators. There also exist standards that propose to start from a hazard analysis to identify leading indicators but they assume accidents can be explained by sequential models. In the paper of Leveson, a new approach is proposed. The design of systems is based on safety-related assumptions. A leading indicator is a warning sign used in monitoring system processes to detect when such assumptions are broken or their validity is changing. In such cases, an action is required to prevent an accident. Those actions can be *shaping actions*, which are used to maintain assumptions, or *hedging actions*, which prepare for the possibility that assumption will fail.

When determining a set of leading indicators, the goal is that the set is complete, consistent, effective, traceable, minimal, continually improving, and unbiased [Lev15]. A structured process can help to achieve this. The basis of a leading indicator program can be safety critical assumptions. In order to guide the identification of them, Leveson presents six types of assumptions. Assumption can be about:

1. System hazards and the causes of them
2. The effectiveness of shaping and hedging actions
3. The operation of the system and the environment
4. Development environment and processes
5. The control structure during operations
6. Severity in risk assessment

STPA can help to identify safety-critical assumptions [Lev15]. First of all they should include that hazards will not occur in a properly designed and operated system. Moreover, the assumptions underlying the identified hazardous scenarios should be determined. Further assumptions are related to the responsibilities assigned to the controllers and whether these are properly enforced. Another source for assumptions are limitations in the design. These are related to basic functional requirements or other environmental assumptions and should be documented. They can relate to hazards that could not be eliminated completely and hence represent accepted risks. Additionally, limitations can be related to trade offs made during the system design. Assumptions can also arise from the safety culture, which is a set of values and cultural assumptions upon which safety-related actions are taken and decisions are made. Ball presents an extension of STPA in his master thesis in order to derive leading indicators from assumptions underlying causality of inadequate control [Bal15]. He concludes that the extension is valid and provides recommendations for implementation of a leading indicator monitoring program. In order to integrate the assumptions into the system

engineering documentation, several methods can be used, for example Intent Specifications [Lev15].

A leading indicator program can be based on the identified assumptions and has three aspects: Identifying leading indicators, creating a safety indicator monitoring program, and embedding this program within a risk management system [Lev15]. The first step is to create leading indicators that will detect when the identified safety-critical assumptions no longer hold. For each leading indicator several aspects should be documented: The associated assumption(s), how and when it will be checked, and the (hedging) action(s) to take if the assumption is violated and hence the indicator is true. After the creation of the leading indicators, the monitoring program must be designed. Many of the assumptions can be handled by shaping and hedging actions and signposts. *Signposts* are points in the future where shaping and hedging actions may be necessary. In these cases it is sufficient to check that these actions are effective. Several designs for the monitoring program are possible. An example is Early Warning Sign Analysis using STPA (EWaSAP) developed by Dokas et al., which is an addition to STPA [DFI13]. It allows to define data indicating the violation of safety constraints and to design assumptions.

In order that leading indicators are effective, they have to be integrated into the risk management program [Lev15]. Detailed actions plans for critical scenarios should be developed and triggers for them specified. Furthermore, responsibilities should be assigned for checking the existence of leading indicators.

Chapter 5

Concept for the STPA DSL

The goal of this master thesis is to develop a DSL for STPA. Since several tools for STPA already exist, the aim is to combine the advantages while minimizing disadvantages. This also includes the requirements stated by Ludvigson [Lud18] and Souza et al. [SPP+19] as used in Section 3.12. Generally, there are two approaches for a DSL: textual and graphical. Both have their pros and cons [GKR+14]. A graphical approach is suitable to give the user an overview of the different elements and how they are connected. However, only being able to add graphical elements as a user via drag and drop or pop-ups in the diagram is tedious, and arranging the elements manually for a good layout is time consuming. Hence, a textual approach is more efficient and less time consuming for the user. Additionally, version control is much easier for text than for graphs. The disadvantage of the textual approach is, like already said, that graphs are better suited to give an overview. In conclusion, a combination of both approaches combines the advantages of both and eliminates the disadvantages of only using one of them. That is why the goal of this thesis is to develop a textual DSL for STPA for which a visualization is generated and layouted automatically. In this way the user has the advantages of both approaches: An overview is given due to the visualization, but no time is consumed by creating and layouting it.

One main benefit of this approach is that the relationships of the STPA components can be visualized, which is so far only provided by the tool SAHRA and no other tool. However, the disadvantages of SAHRA are that it is tedious to only work graphical and large graphs can lead to losing the overview the graph is supposed to give. The DSL approach does not necessarily contain these disadvantages. The remaining chapter further explains the concepts for the DSL (Section 5.1) and the visualization (Section 5.2).

5.1 DSL

As the main goal, the grammar of the DSL should be easily readable and understandable as well as clearly structured such that the different aspects of STPA are separated. This is achieved by using captions for each aspect as shown in Listing 5.1. The aspects are introduced in Section 2.1 and are the following: losses, hazards, system-level constraints, control structure, responsibilities, UCAs, controller constraints, loss scenarios, and safety requirements. For each aspect, several components can be defined. In order to support the tracing shown in Figure 2.2, each component needs to have an ID and a list containing the references to other components. Moreover, a component should have a description, which summarizes the component, leading

to the following general syntax: ⟨ID⟩ ⟨String⟩ [⟨Ref⟩, . . .]. One exception are the losses, which do not have tracing to other components and thus no list at the end. Example components for losses, hazards, and system-level constraints can be seen in Listing 5.2. For the hazards and system-level constraints also sub-hazards and sub-constraints can be defined as shown in Listing 5.3. They are stated using brackets after the parent component. While the sub-hazards do not need a reference list — the tracing is inherit from the parent — the sub-constraints still need a list since they could be used to specify the tracing to sub-hazards. Nevertheless, sub-hazards are allowed to specify a list in which only the references the parent has, are allowed. The sub-hazards can be grouped by defining headers but it is not necessary. Tracing to sub-hazards and -constraints work the same way as for the top-level hazards and constraints.

```
Losses

Hazards

SystemConstraints

ControlStructure

Responsibilities

UCAs

ControllerConstraints

LossScenarios

SafetyRequirements
```

Listing 5.1. The captions for the aspects in STPA.

```
Losses
L1 "Loss of life or serious injury to people"
L2 "Damage to aircraft or objects outside the aircraft"

Hazards
H1 "Loss of aircraft control" [L1, L2]
H2 "Aircraft comes too close to other objects" [L1, L2]

SystemConstraints
SC2 "Aircraft must have safe distance to other objects" [H2]
```

Listing 5.2. Component definitions for losses, hazards, and system-level constraints.

The control structure must be defined alongside the other components as seen in Listing 5.4. It is done by stating a name for the overall structure, which contains the system components of the control structure. Each system component must have a name that is its identifier and it can contain the following elements: hierarchyLevel, processModel, controlActions, and feedback. The *hierarchyLevel* determines the position in the visualization as further explained in Section 5.2. In the *processModel* property, process model variables are listed with their possible values. Control actions that can be send from the current system component to another are stated in the *controlActions* property. They are stated using an ID, a label, ->, and the target. Multiple control actions that are send to the same target are stated together. This also applies to the feedback, which are stated in *feedback*.

Responsibilities are normally stated together with the system component they are assigned to. This is the reason why in the DSL a system component must be stated prior to the

```
Hazards
H2 "Aircraft comes too close to other objects" [L1, L2] {
      H2.1 "Deceleration is insufficient"
   "Acceleration"
      H2.2 "Asymmetric acceleration"
      H2.3 "Excessive acceleration provided"
}

SystemConstraints
SC2 "Aircraft must have safe distance to other objects" [H2] {
   SC2.1 "Deceleration must occur within ..." [H2.1]
   SC2.2 "Asymmetric acceleration must not ..." [H2.2]
}
```

Listing 5.3. Definition of subcomponents for hazards and system-level constraints.

```
ControlStructure
Aircraft {
   FlightCrew {
      hierarchyLevel 0
      processModel {
         BSCUmode: ["on", "off"]
      }
      controlActions {
         [powerOff "Power Off BSCU", powerOn "Power On BSCU"] -> BSCU
      }
   }
   BSCU {
      hierarchyLevel 1
      feedback {
         [mode "BSCU mode"] -> FlightCrew
      }
   }
}
```

Listing 5.4. Control structure definition.

```
Responsibilities
BSCU {
   R1 "Actuate brakes when requested" [SC2.1]
   R2 "Pulse brakes in case of a skid" [SC2.2]
}
FlightCrew {
   R3 "Mnually brake in case of malfunction" [SC2.1, SC2.2]
}
```

Listing 5.5. Responsibilities definition.

```
UCAs
FlightCrew.powerOff {
   notProviding {
      UCA1 "Crew does not provide BSCU Power Off when abnormal WBS behavior
          occurs" [H2.1]
   }
   providing {
      UCA2 "Crew provides BSCU Power Off when Anti-Skid functionality is needed
          and WBS is functioning normally" [H2.3]
   }
   tooEarly/Late {}
   stoppedTooSoon {}
}

ControllerConstraints
C1 "Crew must provide the BSCU Power Off control action during abnormal WBS
    behavior" [UCA1]
C2 "Crew must not provide the BSCU Power Off control action when Anti-SKid
    functionality is needed" [UCA2]
```

Listing 5.6. UCAs and controller constraints definition.

responsibilities as shown in Listing 5.5. The responsibilities itself follow the standard definition of components.

The definition of UCAs is done by first stating the corresponding system component and its control action, as seen in Listing 5.6, in order to simplify the connection of an UCA to a component. Moreover, in this way the UCAs are properly grouped, which improves the readability of the document. For each control action, the four categories of UCAs must be written down. This ensures that no category is forgotten. In the categories, the appropriate UCAs can be defined. However, it is possible to leave a category empty for the case that there is no such UCA. The corresponding constraints can be defined in the `ControllerConstraints` section.

Loss scenarios can either be connected to a UCA or directly to a hazard. That is why there are two possibilities for defining loss scenarios as shown in Listing 5.7. When the scenario is

```
LossScenarios
Scenario1 for UCA1 "Abnormal WBS behavior occurs. Crew does not power off ..." [H2.1]
Scenario2 "Insufficient braking is applied due to ..." [H2.1]

SafetyRequirements
SR1 "Sufficient braking must be applied when ..." [Scenario2]
```

Listing 5.7. Loss scenarios and safety requirements definition.

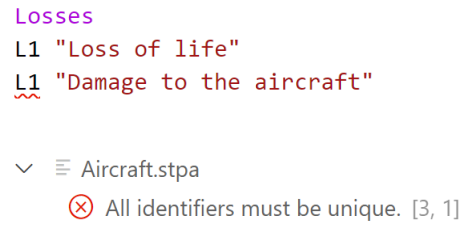
linked to a UCA, it is stated together with the ID of the UCA, otherwise this is omitted. Even if the scenario is linked to a UCA, hazards can still be referenced at the end of the component. However, it is not necessary since the UCA already requires for them to be referenced. It is only for use-cases where the user wants to be able to directly see the connected hazard. Additionally to the aspects in STPA as introduced by Leveson, the DSL supports the definition of concluding safety requirements (also shown in Listing 5.7), which can be linked to the scenarios.

Since STPA does not need to be performed linearly, non-linear usage is also supported. This means that changes can be done to previously defined components. Moreover, it should be usable in every development stage and thus it is not required that components for every aspect are defined. In order to aid the user even more, the references that are stated in the list at the end of a component are checked to be valid IDs and to be a component of the correct aspect. Which aspects are allowed for a component can be seen in Figure 2.2. For the responsibilities and UCAs it is also checked whether the referenced system component exist and in case of the UCAs whether they contain the stated control action. Furthermore, references need to be unique in order to be useful. Therefore, another check validates the uniqueness of all IDs. If one of the checks fails, an error is shown with a corresponding message as seen in Figure 5.1a. Besides this essential characteristics, there are also checks that only lead to a warning as seen in Figure 5.1b. These include controlling that in a reference list an ID is not stated more than once and that the IDs of sub-components start with the same ID as their parent. At last there is a check that examines whether all aspects are defined. If this is not the case, the missing aspects are listed in an info as shown in Figure 5.1c.

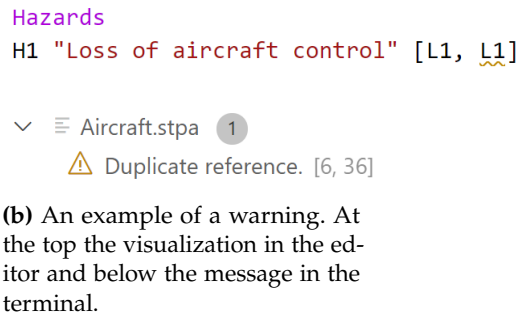
5.2 Visualization

For a better overview of the tracing and thus the relationship between the defined components, the user can request the automatic generation of a visualization. The visualization tab contains two graphs: one representing the control structure and one representing the tracing between the other components of STPA. Both graphs are automatically layouted with the layered algorithm (Section 2.3.2), which assigns nodes to several layers.

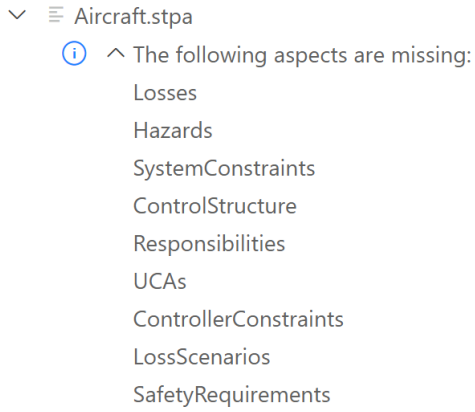
The *control structure graph* is shown in Figure 5.2. It has a node for each system component and is layouted in a way that a hierarchy level is represented by a layer. This means, system components that are in the same hierarchy level are also in the same layer in the visualization.



(a) An example of an error message. At the top the visualization in the editor and below the message in the terminal.



(b) An example of a warning. At the top the visualization in the editor and below the message in the terminal.



(c) An example of an info in the terminal.

Figure 5.1. Visualization of failed checks.

In order to show the typical flow of control actions going from top to bottom and feedback going from the bottom to the top, the layout direction is also top to bottom. This leads to the representation of the hierarchy level 0 as the top layer. The control actions and feedback are visualized as edges. Process models are currently not visualized, they have no influence on the graph.

In the graph for the other STPA components, each component is represented by a node drawn as a rectangle. An example for the defined components in the previous section can be seen in Figure 5.3a. In order to easily be linked to the textual description, the label of the node is the ID of the component. The layer of a node is set based on the aspect it belongs to. Each aspect has its own layer in order to offer a clearly structured visualization and improve the distinguishability of the aspects. The tracing between the components is visualized as edges. For example, if a component $H1$ references $L1$, an edge is shown starting by $H1$ and going to $L1$. This is done for every reference with the goal of giving a better overview of the connections between the components. Loss scenarios are an exception. If they are stated with an UCA and a reference list to hazards, they only have an edge to the UCA since the hazards are already referenced by the UCA. Sub-components are visualized hierarchical in the graph. The node representing the parent contains the nodes representing the sub-components. Unfortunately,

```
ControlStructure
Aircraft {
   FlightCrew {
      hierarchyLevel 0
      processModel {
         BCSUmode: ["on", "off"]
      }
      controlActions {
         [mc "Manual Controls" ]-> OtherSubsystems
         [powerOff "Power Off BSCU", powerOn "Power On BSCU"] -> BSCU
         [manual "Manual Braking"] -> Wheels
      }
   }
   OtherSubsystems {
      hierarchyLevel 1
      feedback {
         [modes "Other system modes", states "states"] -> FlightCrew
      }
   }
   BSCU {
      hierarchyLevel 1
      controlActions {
         [brake "Brake"] -> Wheels
      }
      feedback {
         [mode "BSCU mode", faults "BSCU faults"] -> FlightCrew
      }
   }
   Wheels {
      hierarchyLevel 2
      feedback {
         [speed "Wheel speed"] -> BSCU
      }
   }
}
```

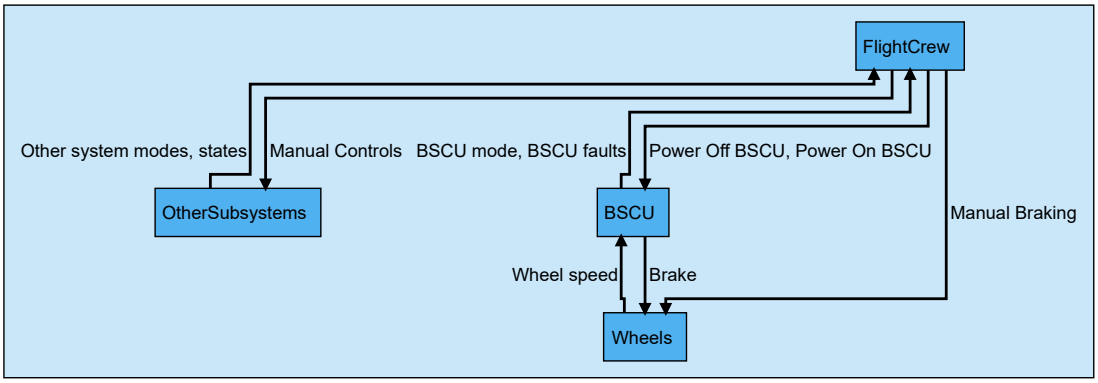
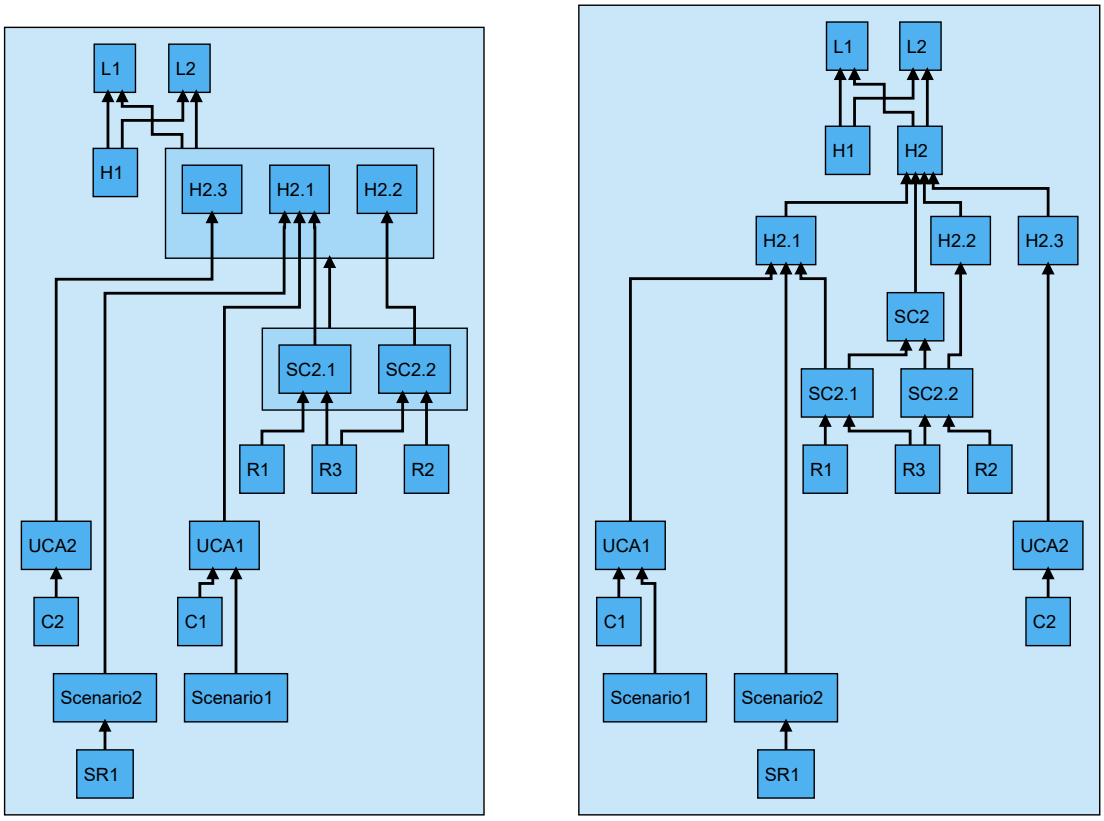


Figure 5.2. Visualization of the control structure defined above.



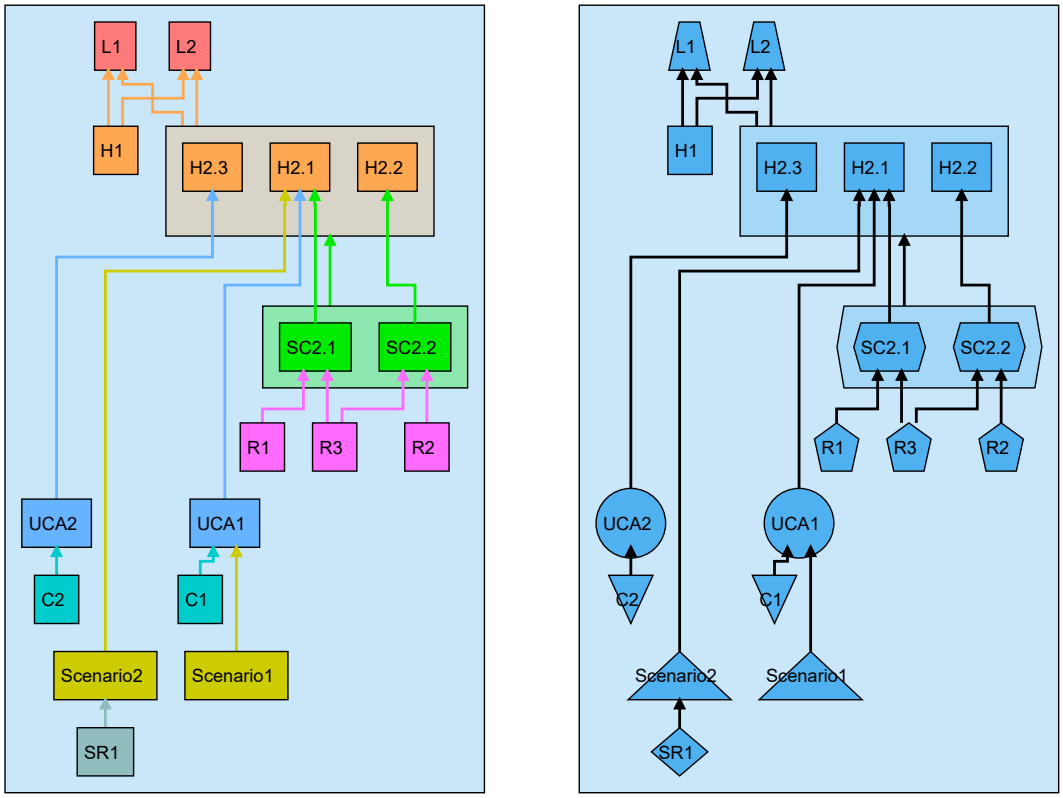
(a) The standard visualization.

(b) The optional visualization of sub-components.

Figure 5.3. The visualization of the STPA graph.

hierarchy crossing edges are not always layouted nicely. That is why the user can change the visualization to the one shown in Figure 5.3b. Instead of entailing the sub-components in the parent, the relationship is shown by edges starting from the sub-component and going to the parent. Additionally, the sub-components get their own layer. The disadvantage is that in this visualization edges have different meanings. Some edges represent the relationship between parent and sub-component while others represent the tracing of a component. In both visualizations, references stated by sub-hazards are ignored. They always inherit the tracing from the parent.

Although each aspect has its own layer, the distinguishability can still be improved. This is done by offering a *colorful style* where each aspect gets its own color as shown in Figure 5.4a. These colors depend on the color theme or, to be more precise, on the color of the label. If the label is white/gray, the brightness of the colors are reduced in order to still provide a good contrast. In larger graphs users possibly have to zoom in to identify individual elements. Other elements that are connected to the ones in the view are possibly not visible any more. Still, it could be helpful to know which aspects and how many of it are defined for and hence connected to a certain element. That is why, in the colorful style, the edges are colored in the



(a) The colorful visualization.

(b) The visualization with different forms.

Figure 5.4. Different options for improving the distinguishability of aspects.

same color as their source. Additionally, this way the edges might be easier to follow as well. For the case that the user do not like the colorful visualization or it is not as helpful as hoped, there is an alternative option to improve the distinguishability: different forms. As seen in Figure 5.4b instead of drawing each node as a rectangle each aspect gets its own form.

Besides the colorful and standard-color styles, there also exist the option of a *print-style* (Figure 5.5). It is supposed to be used if the user wants to print a diagram or do not like the colors at all. Naturally, the color, form, and hierarchy options are independent and thus can be combined as the user prefers.

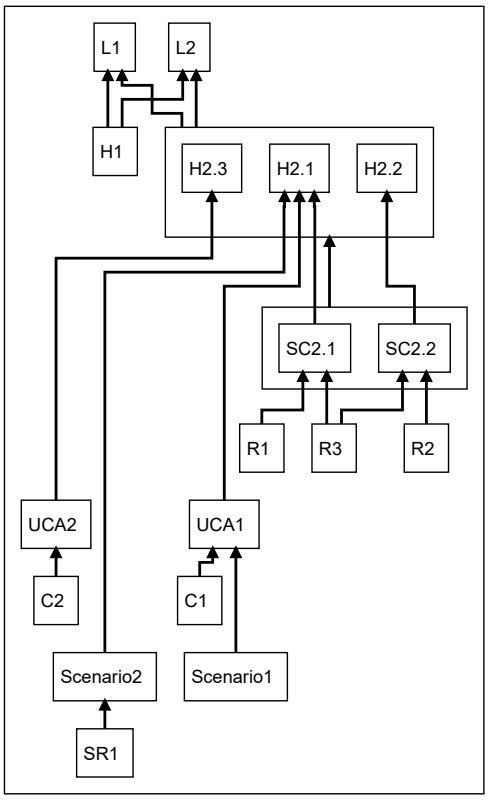


Figure 5.5. The print-stlye visualization.

Chapter 6

Implementation

The approach presented in the previous chapter is implemented as a VS Code Extension, since the VS Code Extension API became increasingly relevant in the past years[1]. It uses Langium (Section 2.3.4) for the DSL and Sprotty (Section 2.3.3) for generating the visualization. There already exists a sprotty-vscode example[2] that is a VS Code Extension combining a Langium-based language server with a Sprotty diagram. This is used as basis for the implementation presented in this section.

The general structure consists of three components: extension, language server, and webview. The *extension* component is the entry point of the extension that combines the language server with the visualization. In the *language server*, the actual DSL, the generation of a graph, and the layout configuration for the layout of the graph is defined. The *webview* contains the generation of the visualization for the graph that can be shown to the user alongside the editor for the DSL. In the remaining chapter the implementation of the extension (Section 6.1), the language-server (Section 6.2), and the webview (Section 6.3) are further explained.

6.1 Extension

As mentioned in Section 2.3.1 the extension needs an entry point. In this case it is the *extension.ts* file in the *extension* folder, which creates an `StpaLspVscodeExtension` in the `activate` function. The commands that should be provided to the user are defined in the *package.json*. The implementation of these is done in the `StpaLspVscodeExtension`. Depending on the command, either a notification is send to the language server or an action is send to the webview. The reactions to these are explained respectively in Section 6.2.4 and Section 6.3.3. `StpaLspVscodeExtension` is also responsible for starting the language client and creating the webview when requested.

6.2 Language Server

The language server is based on Langium. An alternative would have been Xtext. However, when using Xtext with Sprotty, two different programming languages are used: Java and

[1] `https://www.typefox.io/blog/langium-the-new-language-engineering-tool`

[2] `https://github.com/eclipse/sprotty-vscode/tree/master/examples`

TypeScript. This complicates the development as well as the maintenance[3]. Langium solves this problem by enabling to implement the language server in TypeScript as well. Additionally, this leads to a direct integration with the VS Code Extension API[4].

As explained in Section 2.3.4, to create a Langium-based language server a grammar definition, a module and a main file is needed. The module for the STPA DSL, `StpaModule`, is defined in the *stpa-module* file. The services registered in this module are the following: `StpaScopeProvider`, `StpaValidator`, `StpaDiagramGenerator`, `StpaLayoutConfigurator`, and `StpaOptions`. The first two improve the utility while the following two are needed for the combination with Sprotty. The last one is used to store options needed by the language server. In the following, the implementation of the DSL, including the grammar, the scope provider and the validator, are explained (Section 6.2.1) followed by the implementation of the diagram generation (Section 6.2.2). Afterwards, Section 6.2.3 explains the layout options and Section 6.2.4 the `StpaOptions` service.

6.2.1 DSL

The language server contains the actual grammar for the STPA DSL, which is defined in Langium. The entry rule is named `Model` and matches parser rules for each STPA aspect. Since parser rules are defined using EBNF, this form is used to define the rules for every STPA aspect based on the concept for the grammar (Section 5.1). For example, the rule for losses is defined as follows: `Loss: name=ID description=STRING;`. For the references at the end of components, *cross references* are used. The same applies for referenced system components and control actions.

As already mentioned, `StpaScopeProvider` and `StpaValidator` that override the default implementation of the services provided by Langium, are used as scope provider and validator. In order for the `StpaValidator` to be used, the `ValidationRegistry` must be overridden. In the `StpaValidationRegistry`, for each STPA aspect a method is registered that contains the relevant checks for each aspect. These checks are implemented in the `StpaValidator`. There are two checks generating an error message when failing. The first one ensures that all IDs of components, including control actions and feedback, are unique. The other one checks that losses referenced by sub-hazards are also referenced by the parent. A warning message is generated when IDs of sub-components do not start with the ID of the parent or if references in the reference lists are not unique. Furthermore, an info message is shown if not all STPA aspects are defined. However, these checks only cover a subset of the wanted controls explained in Section 5.1. In order to guarantee that references to other components are correct in the STPA context, the `StpaScopeProvider` is implemented.

When the user defines a component that has a cross-reference in the grammar definition, `StpaScopeProvider` provides the objects that are in scope and hence referenceable. In order to create the scope, the declarations of the components that should be available for referencing, are collected. These are found based on the current component that is referencing and the

[3] https://www.typefox.io/blog/langium-the-new-language-engineering-tool

[4] https://www.typefox.io/language-engineering/

type the reference should have, which is determined by the grammar. After that, the set of declarations is used to create the scope. This way it is guaranteed that only correct components can be referenced, as described in Section 5.1.

6.2.2 Diagram Generation

In order to visualize the relationships of the STPA components and the control structure, an `SGraph` must be created. This is done in the `StpaDiagramGenerator` class. It contains the `generateRoot` method that gets the context, which includes the document the user created, and creates the `SModelRoot`. This generated root contains two children: one for the control structure and one for the relationships of the other components. Figure 6.1 shows an overview of the classes used to generate the SGraph.

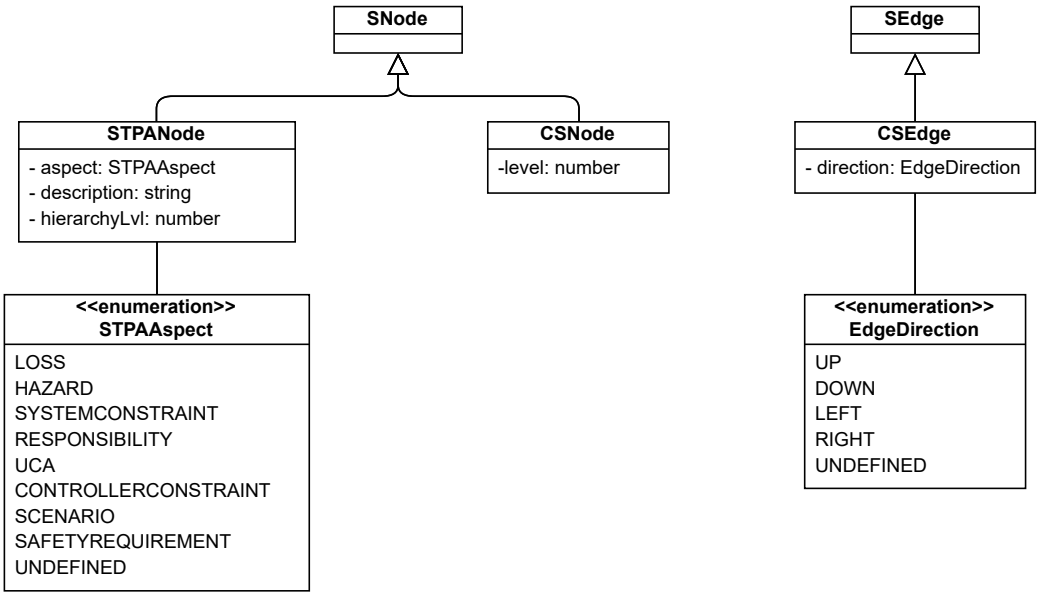


Figure 6.1. The classes for the graph elements.

For the *control structure graph*, each system component is translated to a `CSNode`. Thereby, the `CS` stands for *control structure*. It is an `SNode` that also has the property `level`, which determines the layer the node should be in. This property is set according to the *hierarchyLevel* the user stated for the system component. In order to ensure that the nodes are assigned to the correct layer, the positions of them are set based on the `level`. The control actions and feedback are translated to control structure edges (`CSEdges`), which extend `SEdge` with a `direction` property. At this moment this property is set dependent on whether the edge represents a control action or feedback but it is not used afterwards. The idea is to automatically determine the layer a `CSNode` should be in based on the edges. However, this is not implemented yet.

The *relationship graph* is created by translating each defined component, except the control structure, into an `STPANode`. It extends the `SNode` class with three properties: `aspect`,

description, and hierarchyLvl. The aspect is determined by examining to which STPA aspect the component belongs to. It has the type STPAAspect, which is an enum containing a value for each aspect in STPA. The description contains the description stated by the user. This is not used at the moment, but offers the opportunity to allow a more detailed visualization in the future. The hierarchyLvl is only important for sub-components. For every other component the value is 0. It states how many parents a sub-component has in order to be able to calculate the layer for each component correctly. This calculation is done by determining the maximal hazard and system constraint hierarchy depth. Each aspect gets its own layer by setting the position of the components accordingly. If the hierarchy option is false, also each hierarchy level of sub-components gets its own layer. Besides these STPANodes, SEdges are created representing the tracing of each component.

6.2.3 Layout

After the graph is generated, it still needs layouting. Therefore elkjs (Section 2.3.2) is used. Langium already offers a DefaultLayoutConfigurator, which can be extended to configure the layout done by elkjs. StpaLayoutConfigurator contains the layout options wanted for STPA graphs. Most important is that in the parent node of the control structure and relationship graph the strategies of the layout phases are set to INTERACTIVE as seen in Listing 6.1. As explained in Section 2.3.2, these strategies assign layers and positions in the layer based on the position coordinates of the nodes and updates these coordinates. Therefore, the positions of the nodes must be set manually beforehand, which is already done in StpaDiagramGenerator.

Another option that is set on the parent is the hierarchyHandling. The INCLUDE_CHILDREN value ensures that hierarchy-crossing edges get laid out properly. However, the control structure does not need it, since there is only one hierarchy level. The general layout direction depends on the graph. For the control structure graph the direction is DOWN while for the relationship graph it is UP. That is because in the relationship graph the layers are set based on the ordering of the aspects, meaning the losses are at the top. This leads to edges going from bottom to top.

6.2.4 STPA Options

The *hierarchy option* is provided by the StpaOptions class. It contains a variable determining whether sub-components should be contained in their parent or be in a separate layer. In order that the StpaDiagramGenerator can access this variable, StpaOptions is registered as a new service in the StpaModule. The value of the variable can be toggled by sending a notification to the language server. The reaction to this notification is implemented in the *main* file.

6.3 Visualization

The visualization is done by using Sprotty for the webview. As explained in Section 2.3.3, a container module is needed. Hence, stpaDiagramModule is implemented in which the model

```
protected parentNodeOptions(snode: SNode, index: SModelIndex): LayoutOptions {
   // in the STPA graph this is necessary for hierarchy-crossing edges to be
       better layouted
   let hierarchyHandling = 'INCLUDE_CHILDREN'
   let direction = 'UP'

   if (snode.children && snode.children[0].type == CS_NODE_TYPE) {
      // options for the control structure
      hierarchyHandling = 'SEPARATE_CHILDREN'
      direction = 'DOWN'
   }
   return {
      'org.eclipse.elk.direction': direction,
      'org.eclipse.elk.algorithm': 'layered',
      'org.eclipse.elk.hierarchyHandling': hierarchyHandling,
      // interactive strategies are used to be able to assign layers to nodes
          through positioning
      'org.eclipse.elk.separateConnectedComponents': 'false',
      'org.eclipse.elk.layered.crossingMinimization.semiInteractive': 'true',
      'cycleBreaking.strategy': 'INTERACTIVE',
      'layering.strategy': 'INTERACTIVE'
   };
}
```

Listing 6.1. Layout options fot the parent nodes.

elements are configured. Thereby, all diagram elements shown in Figure 6.1 have their own view they are bound to. The container is created by `StpaPASprottyStarter`, which extends `SprottyStarter`. In the following, at first, Section 6.3.1 explains the styling of the elements using CSS followed by the implementation of the different views in Section 6.3.2. Conclusively, Section 6.3.3 outlines the implementation of diagram options.

6.3.1 CSS

In the CSS file *diagram.css* the colors of the nodes and edges are defined based on the color theme of VS Code. Therefore, mainly class selectors are used, which are introduced in Section 2.3.3. If an element has the `stpa-node` class, the fill color is set based on the aspect the element represents and the color theme. Additionally, the stroke color is set to black. In order to set the stroke color of edges, the class `stpa-edge` is used. This is needed to color the edges accordingly to the aspect of their source node. For the arrows of the edges, the stroke as well as the fill color must be set, which is done with the class `stpa-edge-arrow`. Besides this coloring, there are also classes for coloring the nodes, edges, and arrows in the print style. For the standard coloring the standard Sprotty classes are used. Furthermore, the opacity of a `parent-node` is set to 0.3 so a unified color experience is provided.

6.3.2 Diagram

The actual visualization of each element is implemented in *views*. For the edges `path` is used for the line as well as for the arrow. Both is already provided by the sprotty-vscode example. However, I adjusted the classes that are set. Based on the color-style the user chose, the class is set to `print`, `stpa`, or `sprotty` and thus the wanted style is used. The element created for an `STPANode` is usually a rectangle. However, if the user activated different forms for the STPA aspects, the form of the element is based on the aspect. For each aspect, and hence for each form, a render method exists in *view-rendering*. This contains methods for creating a trapeze, rectangle, hexagon, pentagon, circle, mirrored triangle, triangle, and diamond. Otherwise, just rectangles are created for each node. Additionally, classes are set based on the chosen color-style. The `CSNodes` are simply translated to rectangles and can only contain classes for print-style or sprotty-stlye. For both, `CSNode` and `STPANode`, the parent class is set for the created element if the node contains other nodes. The two nodes containing the control structure and the relationship graph are visualized like `CSNodes`.

6.3.3 Diagram Options

Both options for changing the visualization, the color and the form of `STPANodes`, are stored in the `DiagramOptions` class. In order to be injectable and guarantee that there is only one instance of the class, it is bound in the `stpaDiagramModule` to itself in a singleton scope. The commands that can change the values of the options are defined in *commands*. There exist commands for setting each color-style. Another command, `FormToggleCommand`, toggles the value for the `forms` option. For each command also an action is defined, determining the `KIND`. All commands are registered by configuring them in the `StpaSprottyStarter`. This way the extension can send actions with the according `KIND` in order to trigger a command.

Chapter 7

Evaluation

In this chapter the DSL is used for an exemplary development with STPA (Section 7.1). Therefore, the aircraft example in the STPA handbook [LT18] is used. The usage of the DSL for that example can be seen in Figure 7.1. Furthermore, the DSL is compared to other tools supporting STPA. For that, the suggestions of Ludvigsen and Souza et al. are used, seen in Table 3.1 and Table 3.2. These concern among others: list management for the components, traceability, UCA/context table, version control, systematization, and automation of STPA steps.

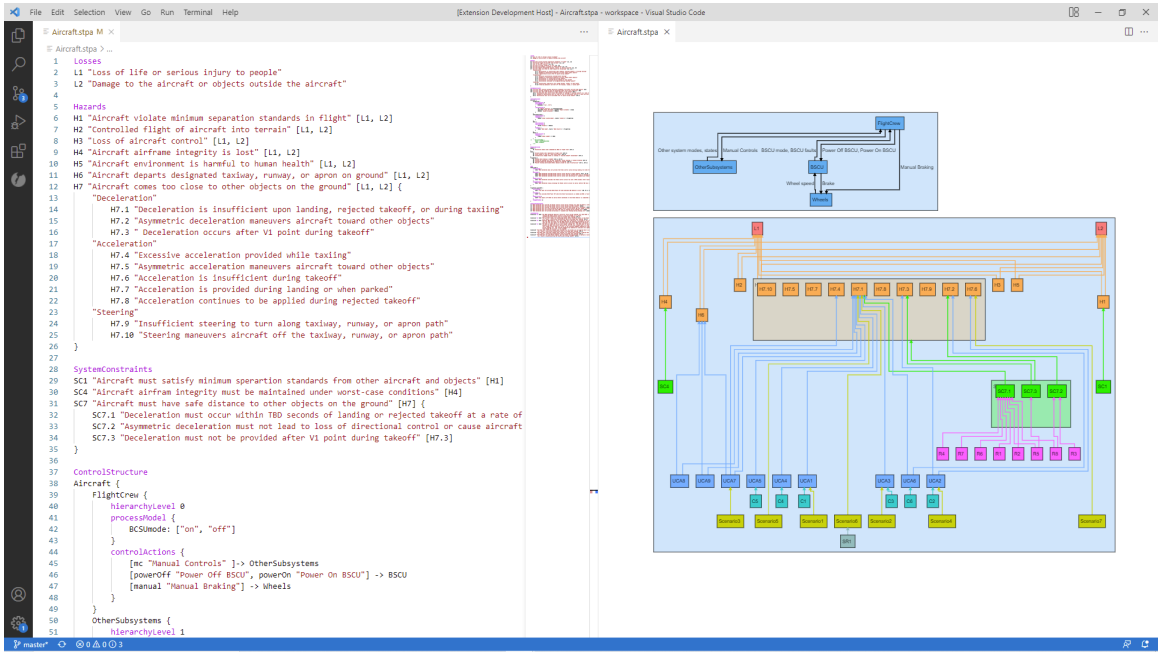


Figure 7.1. The aircraft example modeled in the DSL.

7.1 Exemplary Development

The STPA handbook explains the STPA process by using an aircraft system as example. This example is used to evaluate the process of using the DSL for STPA. The fundamentals, meaning losses, hazards, and system-level constraints, of the aircraft system can be defined in the DSL using a similar syntax to the one in the handbook. Additionally, the definition of sub-hazards

and sub-constraints stated for the aircraft system can also be done in the DSL. Thereby, the sub-hazards can be grouped by headers as suggested in the handbook.

In general, the control structure for the aircraft can be modeled in the DSL. However, hierarchy is not allowed. Hence, grouping of system components cannot be done in the proposed DSL. Moreover, communication between components at the same level is not definable. This restriction is caused by the layered layout algorithm, which does not allow edges between components in the same layer. In later steps of the STPA process, actuators and sensors are added to the control structure. The DSL does not support them directly. Actuators as well as sensors must be defined as standard system components. The visualization of the control structure could also be further improved to look similar to the one in the handbook. Nevertheless, responsibilities can be defined for each system component as done in the handbook.

The UCAs are grouped by four categories as explained earlier (Section 2.1). In the handbook this is done using a table. The first column states the related control action and the other four columns are filled with the UCAs, whereby their category determines their column. Whereas in the DSL, no table is used. Instead, the related control action must be stated followed by the UCAs grouped by their category. The evaluation of the differences between those two approaches is left for future work. However, both approaches offer the same feature scope. Additionally, defining constraints for the UCAs is also supported.

Loss scenarios are generally identified without guidance. However, some guide is offered by dividing them into two types as explained in Section 2.1. In the handbook, the scenarios for the aircraft system are grouped by these types. This is not done directly in the DSL. The user can state scenarios that involve a UCA and scenarios that do not, but ordering or grouping of them is not required. Nevertheless, the user can manually order the definitions of the scenarios based on their type. Even more guidance is provided by causal factors introduced by Leveson [Lev16], which is not supported in the DSL yet.

All in all, the whole STPA process is supported. Additionally, in the DSL the user can define concluding safety requirements. However, improvements regarding the control structure and the guidance for identification of loss scenarios can be done.

7.2 Comparison

For the comparison of the DSL with the other tools, the suggestions from Ludvigsen [Lud18] and Souza et al. [SPP+19] are used.

Regarding the suggestions from Ludvigsen, only few are fulfilled by the DSL. These are traceability, linking UCA to hazards, and version control. Traceability is supported by the reference lists in the DSL as well as by the visualization. This visualization is an advantage in contrast to the other tools. Only SAHRA also visualizes the relationship between the components. Nevertheless, it can be tedious to only work graphically and large graphs can lead to losing the overview the graph is supposed to give. This disadvantage does not exist in the DSL. Moreover, the textual approach naturally allows for an easy version control. Since the

DSL does not contain lists or tables, neither list management nor a UCA table are supported. Furthermore, the control structure cannot be hierarchical and re-evaluation suggestions are not supported. Nevertheless, both can be provided in future work. A relatively big disadvantage of the DSL is that context tables are not used and hence logical simplification and hazard rules are not supported, either. These features are quite helpful to reduce the time needed by the analyst to perform STPA. However, they still can be added in future work.

In contrast to WebSTAMP and XSTAMPP only safety analyses are supported and not security. The other requirements stated by Souza et al. are only partially provided. Identification of UCAs is systematized in the DSL by using the keywords, but as already mentioned no context table is provided. The identification of loss scenarios however is not systematized. Change management is provided by the traceability between the components. The requirements of collaborative analysis and support for verification of the analysis are not provided. In conclusion, two of the six functional requirements are fulfilled by the DSL. From the four non functional requirements two are provided: analysis reusability and portability. For a proper classificaiton of the user experience, an evaluation with real use-cases would be needed.

All in all, the main advantage of the DSL is the visualization of the relationships between the STPA components. However, there is also room for further improvements. For example when the user selects a component in the visualization, the connected components could be highlighted. At the moment, this advantage may not compensate the disadvantage of not providing and automating a context table. This feature should be included in future work in order for the DSL to be useful in real use-cases. Nevertheless, more evaluation should be done by letting analyst experts use the DSL in real use-cases.

Chapter 8

Conclusion

This chapter summarizes the thesis and gives an outlook at future work for improving the DSL.

8.1 Summary

The exploration of the risk analysis topic presented the basis for risk analysis techniques: the causality models. These models are divided into three categories: sequential, epidemiological, and systemic. In the sequential model safety is a reliability problem and accidents are the result of individual component failures. Whereas in the systemic model, safety is a control problem and accidents can also be caused by unsafe interactions between system components. Traditional techniques such as FTA and FMEA are based on the sequential model while STPA is based on the systemic one. There are several more techniques, including combinations of them. In comparison, STPA provides advantages over other techniques. STPA is used in many use cases and organizations such as Google. Furthermore, several extensions, for example STPA-Sec, and improvements are proposed for STPA.

In this thesis a DSL for STPA is implemented as a VS Code Extension using Langium and Sprotty. It provides an editor in which components for each aspect of STPA can be defined. A scope provider and a validator are implemented for the DSL in order to improve the utility. Additionally, a visualization is generated based on the defined components. The visualization shows two graphs: the control structure as well as the relationships of the other components by representing the tracing as edges. Both graphs are automatically layouted using ELK. Moreover, options are provided to change the color-style of the graphs and to determine how sub-components are visualized.

The evaluation of the DSL revealed that generally the whole STPA process is supported, but there is still room for further improvements. Its main advantage in comparison to other tools is the visualization of the relationships between the components while the main disadvantage lays in the missing context tables. Additionally, some suggested features from Ludvigsen and Souza et al. are not provided by the DSL.

In conclusion, the DSL supports the application of STPA. The visualization of the relationships of the components can help the analysts and is so far only supported by the tool SAHRA. Nevertheless, the disadvantages of SAHRA are resolved by using textual definitions of components. The DSL has potential to be a good alternative to other tools. However, further improvements are necessary such as providing context tables.

8.2 Future Work

There are several features that can be added to the DSL, to offer even more support for the application of STPA. One of them is the support of leading indicators introduced in Section 4.6. For example, the definition of assumptions, on which the leading indicators are based, could be allowed and maybe also visualized. Another feature that could be implemented is that definitions of STPA components can be split onto several files and still are visible to other components. This way, the analysts can order the definitions the way they want to. Furthermore, an automatic update of the visualization when the user change the file in the editor could be implemented. The following sections present further future work for the DSL (Section 8.2.1) and the visualization (Section 8.2.2).

8.2.1 DSL

The DSL could further inspect the tracing of the STPA components. It should be checked that every component has a reference list and that every component is referenced at least once. Moreover, a check that UCAs are defined for all control actions would be useful. These checks can help the analysts to not overlook important components.

Regarding the definition of losses, ranking or prioritizing of them should be allowed. This way the analyst gets a better overview which safety requirement are more important than others. Another feature that could be implemented is re-evaluation suggestions as proposed by Ludvigsen [Lud18]. The ranking as well as the re-evaluation suggestions can be done in the DSL but it may be reasonable to show them in the visualization, too.

The control structure can also be improved. First of all, it can be helpful to allow the definition of several control structures. This way control structures for different abstraction levels can be defined. Furthermore, actuators and sensors should be definable explicitly and not as system components. Otherwise, it is possible that the analysts mistake them for system components. This can also be supported in the visualization. The definition of controller system components can be further improved by allowing explicit definition of human controllers. There exist some extensions regarding human controllers (see Section 4.4), which could be considered.

Identification of loss scenarios can be guided more. At the moment, there is no guidance at all but, as mentioned in Chapter 7, Leveson provides causal factors, which help in the identification. Supporting the definition of these or at least showing what causal factors exist like some tools do it, would help in the identification of loss scenarios. Another possibility to improve this identification is to implement the Step 2 Tree proposed by Antoine (Section 4.5) and/or the improvements proposed by Thomas (Section 2.2).

Another improvement considers automation. For the UCAs, a context table could be generated automatically. The user should be allowed to either edit the table manually to select which contexts are hazardous or to define rules as proposed by Gurgel et al. This can be even further improved by providing logical simplification and consistency checks. If a context table is used, additionally the generation of controller constraints and basic scenarios

could be automated as stated by Thomas. Furthermore, formalization of safety requirements could be supported for verification purposes like other tools already offer. In this case, the transformation of safety requirements to formal specifications such as LTL could also be automated.

8.2.2 Visualization

The visualization can be improved by introducing filters. Such filters can be that the selection of an element highlights the elements connected to it or only showing components that belong to a specific aspect. Moreover, the selection of an UCA or responsibility component could lead to a highlighting of the according control action or system component in the control structure. With these features, analysts could better inspect specific elements. Additionally, the selection of a diagram element could highlight the textual definition in order to provide a clearer connection between diagram and editor.

In order to improve the overview, an option to only show one of the two graphs could be added. Another option could be provided to hide specific aspects that are not relevant to the analyst at the moment. The clarity of the diagram could also be improved by allowing the user to select whether only IDs are shown or also the descriptions of the components.

The visualization of the control structure could generally be improves as stated in Chapter 7. First of all, the layout can be improved to include horizontal edges and look more like it would when drawing it manually. Moreover, the process model of a controller should also be visualized. Determining the layer of system components automatically could also reduce the workload for analysts.

Bibliography

[ADP19] Arie Adriaensen, Wilm Decré, and Liliane Pintelon. "Can Complexity-Thinking Methods Contribute to Improving Occupational Safety in Industry 4.0? A Review of Safety Analysis Methods and Their Concepts". In: *Safety* (Dec. 2019), p. 65.

[AL16] Blake Abrecht and Nancy Leveson. "Systems Theoretic Process Analysis (STPA) of an Offshore Supply Vessel Dynamic Positioning System". In: *Massachusetts Institute of Technology* (2016), p. 79.

[Ant13] Blandine Antoine. "Systems Theoretic Hazard Analysis (STPA) applied to the risk review of complex systems: an example from the medical device industry". PhD thesis. Massachusetts Institute of Technology, 2013.

[AOT10] H. Arabian-Hoseynabadi, H. Oraee, and P. J. Tavner. "Failure Modes and Effects Analysis (FMEA) for wind turbines". In: *International Journal of Electrical Power & Energy Systems* (Sept. 2010), pp. 817–824.

[AR10] Terje Aven and Ortwin Renn. *Risk Management and Governance: Concepts, Guidelines and Applications*. Springer Science & Business Media, Sept. 2010.

[AW13] Asim Abdulkhaleq and Stefan Wagner. "Experiences with applying STPA to software-intensive systems in the automotive domain". In: *STAMP Workshop* (2013).

[AW14] Asim Abdulkhaleq and Stefan Wagner. "Open tool support for System-Theoretic Process Analysis". In: *2014 STAMP Workshop, MIT, Boston, USA*. 2014.

[AW15a] Asim Abdulkhaleq and Stefan Wagner. "Integrated Safety Analysis Using Systems-Theoretic Process Analysis and Software Model Checking". In: *Computer Safety, Reliability, and Security*. Ed. by Floor Koornneef and Coen van Gulijk. 2015, pp. 121–134.

[AW15b] Asim Abdulkhaleq and Stefan Wagner. "XSTAMPP: An eXtensible STAMP platform as tool support for safety engineering". In: *2015 STAMP Workshop, MIT, Boston, USA* (2015).

[AW16] Asim Abdulkhaleq and Stefan Wagner. "XSTAMPP 2.0: New Improvements to XSTAMPP Including CAST Accident Analysis and an Extended Approach to STPA". In: *MIT* (2016).

[Bal15] Allen J. (Allen Joseph) Ball. "Identification of Leading Indicators for Producibility Risk in Early-Stage Aerospace Product Development". MA thesis. Massachusetts Institute of Technology, 2015.

[BFP18] Stephan Baumgart, Joakim Froberg, and Sasikumar Punnekkat. “Can STPA be used for a System-of-Systems? Experiences from an Automated Quarry Site”. In: *2018 IEEE International Systems Engineering Symposium (ISSE)*. Oct. 2018, pp. 1–8.

[BHN09] Kate Branford, Andrew Hopkins, and Neelam Naikar. “Guidelines for AcciMap analysis”. In: *Learning from high reliability organisations*. CCH Australia Ltd, 2009.

[BVJ14] Christopher Becker, Qi Van Eikema Hommes, and John A. Volpe National Transportation Systems Center (U.S.) *Transportation Systems Safety Hazard Analysis Tool (SafetyHAT) User Guide (version 1.0)*. Tech. rep. John A. Volpe National Transportation Systems Center (US), Mar. 2014.

[Cle90] PL Clemens. “Event Tree Analysis”. In: *available at: www. fault-tree. net (accessed 25 January 2012)* (1990), p. 13.

[DFI13] Ioannis M. Dokas, John Feehan, and Syed Imran. “EWaSAP: An early warning sign identification approach based on a systemic hazard analysis”. In: *Safety Science* (Oct. 2013), pp. 11–26.

[Don12] Airong Dong. “Application of CAST and STPA to railroad safety in China”. MA thesis. Massachusetts Institute of Technology, 2012.

[Fil13] Associated Normative Machine Consumable Files. “Omg unified modeling languzage tm (omg uml)”. In: *Object Management Group* (2013), p. 752.

[FM14] Brendon Frost and John P T Mo. “System Hazard Analysis of a Complex Socio-Technical System: The Functional Resonance Analysis Method in Hazard Identification”. In: *Proc. of Australian System Safety Conference, Melbourne Australia*. 2014, p. 14.

[FMS+17] Ivo Friedberg, Kieran McLaughlin, Paul Smith, David Laverty, and Sakir Sezer. “STPA-SafeSec: Safety and security analysis for cyber-physical systems”. In: *Journal of Information Security and Applications* (June 2017), pp. 183–196.

[FMS06] Sanford Friedenthal, Alan Moore, and Rick Steiner. “OMG Systems Modeling Language (OMG SysML™) Tutorial”. In: *INCOSE Intl. Symp*. 2006, p. 122.

[Fra17] Megan Elizabeth France. “Engineering for Humans: A New Extension to STPA”. MA thesis. Massachusetts Institute of Technology, 2017.

[GHD15] Danilo Lopes Gurgel, Celso Massaki Hirata, and Juliana De M. Bezerra. “A rule-based approach for safety analysis using STAMP/STPA”. In: *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. Sept. 2015.

[Gil93] Warren Gilchrist. “Modelling Failure Modes and Effects Analysis”. In: *International Journal of Quality & Reliability Management* (Jan. 1993).

[GKR+14] Hans Grönninger, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. “Textbased Modeling”. In: *arXiv preprint arXiv:1409.6623* (Sept. 2014).

[God93] P.L. Goddard. “Validating the safety of embedded real-time control systems using FMEA”. In: *Annual Reliability and Maintainability Symposium 1993 Proceedings*. Jan. 1993, pp. 227–230.

[HCW01] David Huang, Toly Chen, and Mao-Jiun J. Wang. “A fuzzy set approach for event tree analysis”. In: *Fuzzy Sets and Systems* (Feb. 2001), pp. 153–165.

[HHC14] Erik Hollnagel, Jeanette Hounsgaard, and Lacey Colligan. *FRAM - the Functional Resonance Analysis Method*. Center for Kvalitet. Region Syddanmark, 2014.

[HL12] Xin He and Yun-Jun Li. “Software reliability analysis on embedded system based on SFMEA and SFTA model”. In: *2012 International Conference on Systems and Informatics (ICSAI2012)*. May 2012, pp. 2471–2474.

[Hol10] Erik Hollnagel. “The Changing Nature Of Risks”. In: *Ergonomics Australia Journal* (2010), pp. 33–46.

[Hol16a] Erik Hollnagel. *Barriers and Accident Prevention*. Routledge, Dec. 2016.

[Hol16b] Erik Hollnagel. “Investigation as an Impediment to Learning”. In: *Resilience Engineering Perspectives, Volume 1*. CRC Press, 2016, pp. 273–282.

[Hol99] Erik Hollnagel. “Accidents and barriers”. en. In: *Proceedings of lex valenciennes*. 1999, pp. 175–182.

[Hor17] David C. Horney. *Systems-Theoretic Process Analysis and Safety-Guided Design of Military Systems*. en. Tech. rep. MIT Lincoln LaboratoryMassachusetts Institute of Technology, June 2017.

[HVV+20] Lokmane Hezla, Avdotin V.P, Plyuschicov V.G, Sambros N.B, Nadjla Hezla, and Derouiche L. “The Role of Organizational Failure Mode, Effects & Analysis(FMEA) in Risk Management and Its Impact on the Company’s Performance”. In: *Proceedings of the 2020 International Conference on Big Data in Management*. May 2020, pp. 108–112.

[ILT+10] Takuto Ishimatsu, Nancy G. Leveson, John Thomas, Masa Katahira, Yuko Miyamoto, and Haruka Nakao. “Modeling and Hazard Analysis Using STPA”. In: *International Association for the Advancement of Space Safety (IAASS)* (Sept. 2010).

[ILT+14] Takuto Ishimatsu, Nancy G. Leveson, John P. Thomas, Cody H. Fleming, Masafumi Katahira, Yuko Miyamoto, Ryo Ujiie, Haruka Nakao, and Nobuyuki Hoshino. “Hazard Analysis of Complex Spacecraft Using Systems-Theoretic Process Analysis”. In: *Journal of Spacecraft and Rockets* (2014), pp. 509–522.

[KK21] Marianne Kraut and Ioana Victoria Koglbauer. “STPA-Based Analysis of the Process Involved in Enforcing Road Safety in Austria”. In: *Safety* (June 2021), p. 34.

[KRC15] Mohd Faris Khamidi, Imam Rochani, and Dirta Marina Chamelia. "Hazard and Operability Analysis (HAZOP) of Mobile Mooring System". In: *Procedia Earth and Planetary Science* (Jan. 2015), pp. 208–212.

[KRR16] Sven Stefan Krauss, Martin Rejzek, and Monika Ulrike Reif. "Towards a modeling language for Systems-Theoretic Process Analysis (STPA) : Proposal for a domain specific language (DSL) for model driven Systems-Theoretic Process Analysis (STPA) based on UML". In: *ZHAW Züricher Hochschule für Angewandte Wissenschaften* (Dec. 2016).

[KRS+16] Sven S Krauss, Martin Rejzek, Christoph W Senn, and Christian Hilbes. "SAHRA - An integrated software tool for STPA". In: *4th European STAMP Workshop, Zurich, 13-15 September 2016*. 2016.

[Lev04] Nancy Leveson. "A new accident model for engineering safer systems". In: *Safety Science* (Apr. 2004), pp. 237–270.

[Lev14] Nancy G Leveson. "Using STAMP to Develop Leading Indicators". In: *GI-Jahrestagung*. 2014, pp. 597–600.

[Lev15] Nancy Leveson. "A Systems Approach to Risk Management Through Leading Safety Indicators". In: *Reliability Engineering & System Safety* (Apr. 2015), pp. 17–34.

[Lev16] Nancy G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. The MIT Press, 2016.

[Lev19] Nancy G Leveson. "How to Learn More from Incidents and Accidents". In: *Procedia manufacturing* (2019), p. 148.

[Lev20] Nancy Leveson. "Are You Sure Your Software Will Not Kill Anyone?" In: *Communications of the ACM* (Jan. 2020), pp. 25–28.

[Lev21] Nancy Leveson. "Safety III: A Systems Approach to Safety and Resilience". en. In: *MIT Systems Engineering Lab.* (2021), p. 110.

[LGT+85] W. S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie. "Fault Tree Analysis, Methods, and Applications - A Review". In: *IEEE Transactions on Reliability* (Aug. 1985), pp. 194–203.

[LH83] N.G. Leveson and P.R. Harvey. "Analyzing Software Safety". In: *IEEE Transactions on Software Engineering* (Sept. 1983), pp. 569–579.

[LSS11] Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. "A Guided Tour of the CORAS Method". In: *Model-Driven Risk Analysis: The CORAS Approach*. Ed. by Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. Springer, 2011, pp. 23–43.

[LT18] Nancy Leveson and John P Thomas. "STPA Handbook". In: *MIT Partnership for Systems Approaches to Safety and Security (PSASS)* (2018).

[Lud18] Niklas Ludvigsen. "Prototyping a digital support tool for an agile implementation of STPA". MA thesis. NTNU, 2018.

[LW96] R.R. Lutz and R.M. Woodhouse. "Contributions of SFMEA to requirements analysis". In: *Proceedings of the Second International Conference on Requirements Engineering*. Apr. 1996, pp. 44–51.

[LWF+14] Nancy Leveson, Chris Wilkinson, Cody Fleming, John Thomas, and Ian Tracy. "A Comparison of STPA and the ARP 4761 Safety Assessment Process". In: *MIT PSAS Technical Report, Rev 1* (2014), p. 79.

[McK88] T.C. McKelvey. "How to Improve the Effectiveness of Hazard and Operability Analysis". In: *IEEE Transactions on Reliability* (June 1988), pp. 167–170.

[MN21] Alimeh Mofidi Naeini and Sylvie Nadeau. "FRAM and STAMP new avenue for risk analysis of manufacturing in the context of industry 4.0". In: *GfA-Press*. 2021.

[Mon16] Daniel R. (Daniel Ramon) Montes. "Using STPA to Inform Developmental Product Testing". PhD thesis. Massachusetts Institute of Technology, 2016.

[MZP+06] I. Maglogiannis, E. Zafiropoulos, A. Platis, and C. Lambrinoudakis. "Risk analysis of a patient monitoring system using Bayesian Network modeling". In: *Journal of Biomedical Informatics* (Dec. 2006), pp. 637–647.

[NKM+11] Haruka Nakao, Masa Katahira, Yuko Miyamoto, and Nancy Leveson. "Safety guided design of crew return vehicle in concept design phase using STAMP/STPA". In: *Proc. of the 5: th IAASS Conference*. 2011, pp. 497–501.

[Ono97] K. Onodera. "Effective techniques of FMEA at each life-cycle stage". In: *Annual Reliability and Maintainability Symposium*. ISSN: 0149-144X. 1997, pp. 50–56.

[ORS05] Frank Ortmeier, Wolfgang Reif, and Gerhard Schellhorn. "DEDUCTIVE CAUSE-CONSEQUENCE ANALYSIS (DCCA)". In: *IFAC Proceedings Volumes* (Jan. 2005), pp. 62–67.

[Pop19] G Pope. *Risk Management Using Systemic Theoretic Process Analysis (STPA)*. Tech. rep. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2019.

[Rea90] James Reason. *Human Error*. Cambridge University Press, Oct. 1990.

[RS15] Enno Ruijters and Mariëlle Stoelinga. "Fault Tree Analysis: A survey of the state-of-the-art in modeling, analysis and tools". In: *Computer Science Review* (Feb. 2015), pp. 29–62.

[SBB93] D.S. Savakoor, J.B. Bowles, and R.D. Bonnell. "Combining sneak circuit analysis and failure modes and effects analysis". In: *Annual Reliability and Maintainability Symposium 1993 Proceedings*. Jan. 1993, pp. 199–205.

[SBF+16] Antonio Scappaticci, Rod Benson, Daniel Foley, and Darryl Kellner. "Sneak Circuit Analysis: Lessons Learned For Beginners Based on a Successful Application". In: *2016 Annual Reliability and Maintainability Symposium (RAMS)*. Jan. 2016.

[SBF+19] Sardar Muhammad Sulaman, Armin Beer, Michael Felderer, and Martin Höst. "Comparison of the FMEA and STPA safety analysis methods–a case study". In: *Software Quality Journal* (Mar. 2019), pp. 349–387.

[Sha16] Stuart S. Shapiro. "Privacy Risk Analysis Based on System Control Structures: Adapting System-Theoretic Process Analysis for Privacy Engineering". In: *2016 IEEE Security and Privacy Workshops (SPW)*. May 2016, pp. 17–24.

[Slo20] Hannah M. Slominski. "Using STPA and CAST to Design for Serviceability and Diagnostics". MA thesis. Massachusetts Institute of Technology, 2020.

[Son12] Yao Song. "Applying System-Theoretic Accident Model and Processes (STAMP) to Hazard Analysis". MA thesis. McMaster University, Apr. 2012.

[SPP+19] Fellipe G. R. Souza, Daniel P. Pereira, Rodrigo M. Pagliares, Simin Nadjm-Tehrani, and Celso M. Hirata. "WebSTAMP: a Web Application for STPA & STPA-Sec". In: *MATEC Web of Conferences* (2019).

[SSH14] Christoph Daniel Schulze, Miro Spönemann, and Reinhard von Hanxleden. "Drawing Layered Graphs with Port Constraints". In: *Journal of Visual Languages & Computing* (Apr. 2014), pp. 89–106.

[ST14] Dajing Suo and John Thomas. "An STPA Tool". In: *STAMP 2014 Conference at MIT*. 2014.

[STT81] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. "Methods for Visual Understanding of Hierarchical System Structures". In: *IEEE Transactions on Systems, Man, and Cybernetics* (Feb. 1981), pp. 109–125.

[Sum18] Sarah E. Summers. "Systems Theoretic Process Analysis Applied to Air Force Acquisition Technical Requirements Development". MA thesis. Massachusetts Institute of Technology, 2018.

[Suo16] Dajiang Suo. "Tool-assisted hazard analysis and requirement generation based on STPA". MA thesis. Massachusetts Institute of Technology, 2016.

[SWH13] Sardar Muhammad Sulaman, Kim Weyns, and Martin Höst. "A review of research on risk analysis methods for IT systems | Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering". In: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. 2013, pp. 86–96.

[Tho13] John P. Thomas. "Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis". PhD thesis. Massachusetts Institute of Technology, 2013.

[Tho14] Cameron L Thornberry. "Extending the Human-Controller Methodology in Systems- Theoretic Process Analysis (STPA)". MA thesis. MIT, 2014.

[TMT18] Yoshinari Toda, Yutaka Matsubara, and Hiroaki Takada. "FRAM/STPA: Hazard Analysis Method for FRAM Model". In: *Proceedings of the 2018 FRAM Workshop. Cardiff, Wales*. 2018.

[UW12] P Underwood and P Waterson. "A critical review of the STAMP, FRAM and Accimap systemic accident analysis models". In: *Advances in human aspects of road and rail transportation* (2012), p. 11.

[UW13] Peter Underwood and Patrick Waterson. "Systemic accident analysis: Examining the gap between research and practice". In: *Accident Analysis & Prevention* (June 2013), pp. 154–164.

[WBV+17] Hans Wienen, Faiza Allah Bukhsh, Eelco Vriezekolk, and Roel Wieringa. "Accident Analysis Methods and Models - a Systematic Literature Review". In: *Centre Telematics Inf Technol* (2017).

[WH08] R Woltjer and E Hollnagel. "Functional modeling for risk assessment of automation in a changing air traffic management environment". In: *Proceedings of the 4th international conference working on safety*. 2008.

[Yaz02] Zeki Yazar. "A qualitative risk analysis and management tool – CRAMM". In: *SANS InfoSec Reading Room White Paper* (2002), p. 13.

[YL13] William Young and Nancy Leveson. "Systems Thinking for Safety and Security". In: *Proceedings of the 29th Annual Computer Security Applications Conference*. 2013, pp. 1–8.

[YRL19] Abouzar Yousefi, Manuel Rodriguez Hernandez, and Valentin Lopez Peña. "Systemic accident analysis models: A comparison study between AcciMap, FRAM, and STAMP". In: *Process Safety Progress* (2019).

[YWL21] Jinghua Yu, Stefan Wagner, and Feng Luo. "Data-flow-based adaption of the System-Theoretic Process Analysis for Security (STPA-Sec)". In: *PeerJ Computer Science* (Feb. 2021).

[ZKM+21] Nanda Anugrah Zikrullah, Hyungju Kim, Meine JP van der Meulen, Gunleiv Skofteland, and Mary Ann Lundteigen. "A comparison of hazard analysis methods capability for safety requirements generation". In: *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* (Dec. 2021), pp. 1132–1153.

[Zou18] Jiahui Zou. "Systems-Theoretic Process Analysis (STPA) Applied to the Operation of Fully Autonomous Vessels". MA thesis. NTNU, 2018.

[ZZ14] Mengni Zhu and Zheng Zhou. "System reliability and Sneak Circuit Analysis". In: *2014 10th International Conference on Reliability, Maintainability and Safety (ICRMS)*. Aug. 2014, pp. 369–373.

Abbreviations

DSL	Domain Specific Language
STAMP	System-Theoretic Accident Model and Processes
STPA	System-Theoretic Process Analysis
STPA-Sec	STPA for Security
EWaSAP	Early Warning Sign Analysis using STPA
CAST	Causal Analysis based on System Theory
FTA	Fault Tree Analysis
SFTA	Software FTA
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Modes, Effects and Criticality Analysis
SFMEA	Software FMEA
HAZOP	Hazard and Operability Study
CHAZOP	Control HAZOP
SCA	Sneak Circuit Analysis

FRAM	Functional Resonance Accident Model
DCCA	Deductive Cause-Consequence Analysis
ETA	Event Tree Analysis
FETA	Fuzzy FTA
CRAMM	CTTA Risk Analysis and Management Methodology
FHA	Functional Hazard Analysis
SHA	System Hazard Analysis
UCA	Unsafe Control Action
JAXA	Japan Aerospace Exploration Agency
HTV	H-IIB Transfer vehicle
SAHRA	STPA based Hazard and Risk Analysis
XSTAMPP	Extensible STAMP Platform
SafetyHAT	Safety Hazard Analysis Tool
CREAM	Cognitive Reliability and Error Analysis Method
WYLFIWYG	What-You-Look-For-Is-What-You-Get
WYSIWYG	What-You-See-Is-What-You-Get

RPN	risk priority number
HE	Hazardous event
HLCS	High Level Control Structure
FIS	Functional Interaction Structure
ISS	International Space Station
CV	Crew return Vehicle
SoS	system of systems
CBTC	Communication Based Train Control
OSV	Offshore Supply Vessel
DoD	Department of Defense
SpecTRM	Specification Tools and Requirements Methodology
SpecTRM-RL	SpecTRM Requirements Language
RCP	Rich Client Platform
LTL	Linear Temporal Logic
A-CAST	Automated CAST
XSTPA	Extended Approach to STPA

PDE	Plug-in-Development Environment
UML	Unified Modeling Language
SysML	System Modeling Language
EA	Enterprise Architect
GUI	Graphical User Interface
ELK	Eclipse Layout Kernel
MIT	Massachusetts Institute of Technology
XML	eXtensible Markup Language
EBNF	Extended Backus-Naur Form
CSS	Cascading Style Sheets
LSP	Language Server Protocol
AST	Abstract Syntax Tree
SVG	Scalable Vector Graphics
VS Code	Visual Studio Code
API	Application Programming Interface