SAFETY AND RELIABILITY
2022, VOL. 41, NO. 4, 197–224
https://doi.org/10.1080/09617353.2022.2145647

Taylor & Francis
Taylor & Francis Group

OPEN ACCESS Check for updates

STPA enabled safety assessment in the architecting of complex systems

Sergio Jimeno Altelarrea, Atif Riaz and Marin D. Guenov

School of Aerospace, Transport, and Manufacturing, Cranfield University, Cranfield, UK

ABSTRACT

STPA is a hazard assessment technique that represents systems as hierarchical control structures composed of feedback control loops. Existing computational support focuses on creating the diagrams that depict these hierarchies. However, the elements in the loops and the signals exchanged must be determined manually. This impedes safety assessment, thus reducing the number of designs that can potentially be explored. Furthermore, the manual approach does not guarantee the correct update of the architecture with changes resulting from safety assessment, which can make the architecture inconsistent with the safety assessment. To overcome these limitations, proposed for the first time are two methods that automate the creation of: (1) hierarchical control structures and (2) detailed control loops. The methods create STPA models by analysing the architecture, which is modelled as a graph. The concept is illustrated with a representative example of a wheel brake system. The resulting models are compared with those obtained manually by the authors of STPA. The automation is shown to significantly reduce the required time and effort. It was also found to ensure consistency among the safety analysis and the architecture definition as it requires safety features to be included in the architecture before being considered in STPA analysis.

ARTICLE HISTORY Received 4 April 2022; Revised 13 October 2022; Accepted 17 October 2022

KEYWORDS Design for safety; Systems-Theoretic Process Analysis (STPA); Hazard assessment; model-based systems engineering

1. Introduction

Safety refers to the avoidance of catastrophic effects such as injuries, loss of life and environmental damage. It is a crucial analytical element of systems design, as it rules out those options that fulfil the design goals but do not guarantee the elimination of behaviours that can lead to catastrophic effects.

CONTACT Sergio Jimeno Altelarrea s.jimeno@cranfield.ac.uk School of Aerospace, Transport, and Manufacturing, Cranfield University, Cranfield, Bedfordshire, UK

This article has been republished with minor changes. These changes do not impact the academic content of the article.

© 2022 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Safety is particularly important for civil aircraft where accidents could result in many fatalities. This has motivated the establishment of regulatory agencies, such as the European Aviation Safety Agency (EASA), which define the minimum safety requirements aircraft must meet to obtain a certificate of airworthiness. Their efforts have succeeded in reducing the number of fatalities involving passenger and cargo operations of large aeroplanes worldwide (European Aviation Safety Agency, 2020a). However, the growing complexity of systems and the use of innovative solutions, such as novel configurations and increasingly more electric aircraft, introduce new challenges. Traditional safety methods might be insufficient for complex systems and much of the past experience regarding safety might not apply to the new designs (Leveson, 2012, pp. 3–6). Within this context, the broader aim of this research is to facilitate the application of safety methods that can improve the efficiency of the safety assessment process of complex, innovative systems.

1.1. Hazard assessment and the system theoretic process analysis STPA

Two standards, the ARP 4754 A (S-18 Aircraft and Sys Dev and Safety Assessment Committee, 2010) and ARP 4761 (S-18 Aircraft and Sys Dev and Safety Assessment Committee, 1996), describe the safety assessment process, which aims to show compliance with safety regulations such as EASA's CS-25 (European Aviation Safety Agency, 2020b). As defined by these standards, Functional Hazard Assessment (FHA) is the stage in the safety assessment process that identifies potential functional failures and classifies the associated hazards. Hazard assessment is a part of the FHA process where the system is examined to identify safety-related risks (Federal Aviation Administration, 2000), which can be used to derive safety requirements for the system.

Traditionally, hazard assessment revolves around the processes and methods described in the ARP 4761 such as fault tree analysis (FTA) and failure modes and effects criticality analysis (FMECA), and other traditional methods such as event tree analysis (ETA) and hazard and operability analysis (HAZOP). According to Leveson et al. (2014), these methods have been effective when applied to relatively simple electromechanical designs, but they are no longer suitable for more complex and software-intensive systems. This is because in complex systems accidents may result from unsafe interactions among the components and not just from component failures.

To overcome these limitations, alternative hazard assessment methods such as the System Theoretic Process Analysis STPA (Leveson, 2012) have been proposed. As shown by a review by Patriarca et al. (2022), the method has gained popularity in recent years, especially in the fields of aviation, maritime and automotive, which account for the highest number of publications. STPA is based on STAMP, an accident model that is founded on basic systems theory concepts (as opposed to traditional methods, which are based on reliability theory) and includes additional causes for accidents such as system design errors, human error and various types of systemic accident causes (Leveson, 2004). Leveson and Thomas (2018) argue that STPA hazard analysis can find the same failure scenarios found by traditional analyses but also many more software-related and non-failure scenarios. Additionally, STPA is claimed to be less costly in terms of time and resources than traditional methods (Leveson & Thomas, 2018). STPA-based safety assessment (Leveson, 2012; Leveson & Thomas, 2018) consists of four steps:

1. **Define the purpose of the analysis** identifying system-level hazards (states that lead to a catastrophic effect) and constraints (system-level requirements to prevent hazards).
2. **Model the control structure**, which is composed of feedback control loops ordered by a decreasing level of control authority. The structure includes controllers and controlled processes, which exchange control actions, feedback signals and other inputs and outputs.
3. Examine the control structure to **identify unsafe control actions**, which can lead to a hazard in a particular context, as they are not provided, provided improperly, or with inadequate timing. Controller constraints (safety requirements) are defined to prevent such actions.
4. Examine detailed control loops to **identify loss scenarios**, such as unsafe controller behaviour, inadequate feedback or component failures that can lead to unsafe control actions.

1.2. Existing STPA support methods and computation tools

A total of eleven state-of-the-art software tools that support STPA and two methods, which use STPA to automate the generation of safety requirements, were reviewed. Table 1 summarises the results from the literature review. It provides information about how the tools and the method support the STPA methodology based on three capabilities:

Table 1. Review of STPA methods.

Method	Diagrams & Tables	Traceability	Automation
A-STPA (Abdulkhaleq, 2014)	Yes	No	No
XSTAMPP (SE-Stuttgart, 2019)	Yes	No	No
An STPA tool (Thomas & Suo, 2015)	Yes	No	Context table, conflicts, requirements
SAHRA (Krauss, Rejzek, Senn, et al., 2016) (Krauss, Rejzek, Reif, et al., 2016)	Yes	Yes	No
Risk Management Studio (Risk Management Studio, 2019) (Björnsdóttir & Rejzek, 2017)	Yes	Yes	List of control actions
STAMP Workbench (Information-technology Promotion Agency, Japan, 2018)	Yes	No	No
Astah System Safety (Astah, 2021)	Yes	No	No
CAIRIS (Shamal Faily, 2021)	Yes	Yes	No
SafetyHAT (Volpe National Transportation Systems Center, 2014)	Yes	No	No
Abdellatif and Holzapfel (Abdellatif & Holzapfel, 2021)	Yes	No	Components in hierarchy, some verification
WebSTAMP (Souza, Pereira, et al., 2019)	Yes	Yes	Context table, guidewords for scenarios
Method to Automate Generation of Safety Requirements (Thomas, 2013) (Thomas & Leveson, 2013)	No	No	Context table, conflicts, requirements
Rule-Based Approach for STPA (Gurgel et al., 2015)	No	No	Context table, guidewords for scenarios

- **Diagrams & Tables** refers to the ability of a tool to provide a user interface to edit STPA analysis data, draw the control structure diagrams and edit tables such as the losses, hazards or unsafe control actions tables.
- **Traceability** implies that the tool is able to automatically indicate which elements, such as losses, hazards and unsafe control actions, relate to others. Traceability is obtained by automatically following links between elements, which are created manually by the analysts.
- **Automation** refers to additional support that automates any part of the STPA process except for traceability (The right column of Table 1 also lists particular automated capabilities where applicable).

All the computational tools provide a user interface which users can employ to create control loops and hierarchy diagrams, and populate tables with losses, hazards and control actions among others. The methods (last two rows of the table) are not computational tools and therefore do not have a user interface.

The most common automation feature is traceability, which is explicitly supported by *SAHRA*, *Risk Management Studio* and *CAIRIS*. *SAHRA* goes beyond the standard tabular format and introduces mind maps, a directed graph that enables traceability and presents elements such as losses, hazards and unsafe control actions among others.

A greater degree of automation is provided by the method for automated generation of formal model-based safety requirements (Thomas, 2013; Thomas & Leveson, 2013) and *An STPA Tool*, as it implements such a method. The method explains how to generate a list of potential hazardous control actions by combining controllers, types of action, control actions and contexts. Requirements are generated by determining the combinations that prevent hazardous behaviour. This can be automated depending on the availability of behavioural models. The method is also able to check the lack of consistency which arises when both providing or not providing an action within the same context is considered hazardous or when a necessary function of the system results in a hazard.

Gurgel et al. (2015) extend the method by Thomas (2013) with a rule-based approach that assists in the identification of hazardous contexts. *WebSTAMP* is a web-based STPA tool that implements this approach.

The tool by Abdellatif and Holzapfel (2021), which links Simulink to STPA, is the only one that automatically determines the components in the STPA control hierarchy. However, it requires manual linking of the components of the hierarchy. It also provides automated verification of missing links (control or feedback) between controllers and controlled processes, and sensor compatibility.

As the review shows (Table 1), the degree of automation provided by the existing tools is generally low, especially regarding the creation of STPA diagrams, which model the control hierarchy and control loops. This can impact the total time required for STPA analysis, which is likely to result in common problems such as safety assessment lagging behind other aspects of the design (Delange & Feiler, 2014) and sometimes taking place later on in the product development process after the design is finalised (Mhenni et al., 2014). Manual application of safety methods is susceptible to errors when extracting safety models from the system definition, which has been found to bring unnecessary subjectivity and inconsistency to the analyses (Joshi et al., 2007). The lack of consistency

between different design abstractions and outdated models as the design progresses has been identified as one of the primary sources of inconsistency in safety analyses (Papadopoulos et al., 2001). Errors when creating STPA models and inconsistent design abstractions might lead to problems such as missing or misplacing links or omitting elements in the control hierarchies and loops.

To overcome the above-presented limitations, two novel methods have been developed, enabling the automatic creation of STPA control hierarchies and control loops. The effect of such automation is twofold: it accelerates the STPA process while ensuring its consistency with the evolving system design. That is, safety can be performed concurrently with system architecting. The scope of the research is restricted to the conceptual design stage, which is critical in its impact on the whole product development process. For example, safety deficiencies discovered at later design stages might need to be corrected at the expense of performance or may require substantial costly redesign.

The rest of the paper is organised as follows. The proposed methods are presented in Section 2. Section 3 demonstrates the application of the proposed approach to a representative use case and compares it to the manual application of the method. Finally, conclusions are drawn and future work is suggested in Section 4.

2. Proposed methods

This section describes two novel methods developed to support the STPA hazard assessment process in the identification of unsafe control actions and loss scenarios. Specifically, the methods enable for the first time the automatic creation of the models required for steps 2, 3 and 4 of the STPA hazard assessment process (see Section 1.1.). The first method automatically generates the high-level view of the hierarchical control structures including all controllers. This is necessary to identify unsafe control actions during Step 2 of STPA. The second method automatically creates more detailed control loops, whose additional information supports the identification of loss scenarios during Step 4 of STPA. Both methods use the information in the (system) architecture to elaborate such models, and therefore, the quality and level of detail of the STPA analysis will be influenced by the quality and level of detail of the architecture definition.

2.1. Input to the methods, the logical view

The input to the methods is the logical view of the system, which is the part of the architecture definition that displays the solutions and the way they are interconnected. Solutions are the elements (such as parts, components or subsystems) that perform (fulfil) the required functions of the system (Guenov et al., 2020). As defined in the STPA methodology, certain solutions have the role of controllers in the systems, whereas the rest correspond to the controlled processes.

Solutions exchange energy, matter and signal flows through their input and output ports. These relations are referred to as flow relations. Flow relations can be modelled as a directed graph (Guenov et al., 2020). This graph can be used to determine which solutions are affected by the flow originating at one solution or which solutions affect a particular solution. Relations of this kind are used within the context of STPA to identify the flow exchange among controllers and processes. An example of a logical flow view is shown in Figure 1, which illustrates the architecture of a wheel brake system. The hydraulics, which provides the power to the wheel brake, can be controlled either directly by the crew

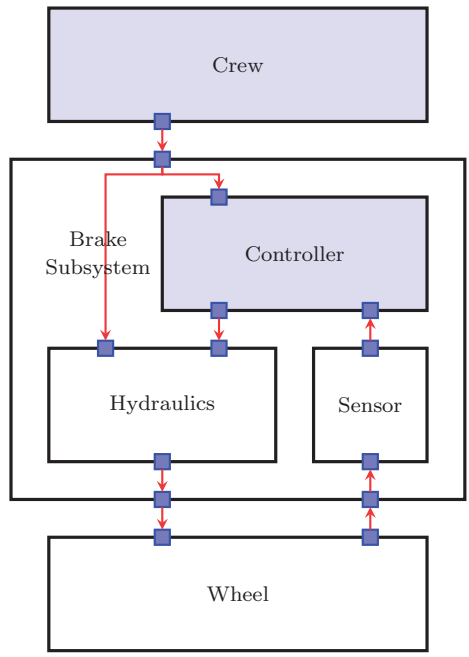


Figure 1. Example of a logical flow view.

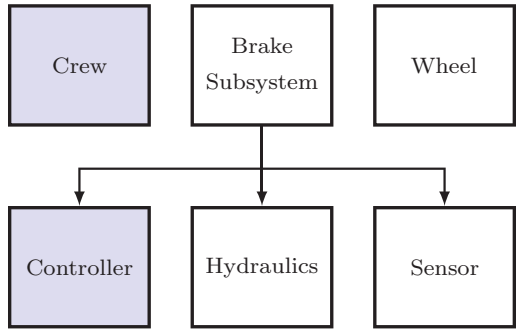


Figure 2. Example of a hierarchical view.

or via a controller. The controller receives feedback from a sensor that measures the status of the wheel.

Solutions can be formed by the aggregation of other solutions. For example, several simple components might form a subsystem. These relations are referred to as hierarchical relations. Hierarchies can be modelled as a tree — a directed acyclic graph. This graph can be used to determine the subsystem a component belongs to or the actual components that form a subsystem at the various levels of the hierarchy. Hierarchical relations provide a means of determining the position of the controllers in the STPA hierarchical control structure. Figure 2 shows the same architecture as in Figure 1 but emphasises how solutions are aggregated into subsystems. In particular, the figure highlights that the controller, the hydraulics, and the sensor are defined as children of the brake subsystem.

The methods were developed with the assumption that a logical view of the system exists, as part of the system architecting process. The logical view is used for defining the solutions (components) of the system and their interrelations and is not tied to any particular analysis. In this research, the input to the presented methods is an RFLP model of the architecture defined according to the framework introduced in (Jimeno Altelarrea, 2021). The RFLP paradigm, based on the German Guideline Design methodology for Mechatronic Systems (VDI 2206, VDI Department of Product Development and Mechatronics, 2004) assumes that systems architecting is distributed over four notional domains: Requirements, Functional, Logical and Physical. It must be noted that the proposed methods are not restricted to the RFLP paradigm and should be easily adaptable to other kinds of architectural representations, if they include equivalent concepts

to flow and hierarchical relations and provide a means to identify controllers.

2.2. Creation of hierarchical control structures

The hierarchical control structure is a graph formed by controller nodes and one or more process nodes at the bottom of the hierarchy. Each controller solution in the logical view is mapped to a controller node in the hierarchy, and the rest of the solutions are grouped into process nodes. By default, only one process node is created in the presented method for creating hierarchical control structures. However, the user can specify more process nodes by indicating which solutions are to be included. Solutions for which no process node is specified will be grouped in the default node.

The method uses Algorithm 1 to create the hierarchy and add the links from controller and non-controller solutions. Specifically, Algorithm 1 calls the function ADD-LINKS-FROM-CONTROLLER, which is described in Algorithm 2, to create the links in the hierarchy, starting from or ending at a controller κ. The connections are determined by applying the following rules:

Algorithm 1: Creation of Hierarchical Control Structures

```
1  Function CREATE-CONTROL-HIERARCHY(logical-view, process-nodes)
       Inputs : The logical view of the system (logical-view).
                The desired grouping for process nodes (process-nodes).
       Output : The hierarchical control structure (H) corresponding to logical-view
2      H ← new HIERARCHY()
3      for each κ ∈ GET-CONTROLLERS(logical-view)
4          ADD-LINKS-FROM-CONTROLLER(logical-view, process-nodes, H, κ)
5      end
6      for each σ ∈ GET-NON-CONTROLLERS(logical-view)
7          ADD-LINKS-FROM-NON-CONTROLLER(logical-view, process-nodes, H, σ)
8      end
9      return H
10 end
```

1. Traverse the logical view of the architecture starting from a port p in controller κ including every component that can be reached by following flow links in the direction of the flow. The traversal through a particular link ends when a controller solution is found, immediately after including such controller. This forms the set of components Σ_{l_κ} influenced by κ.

2. The controller solutions in Σ_{l_κ} represent the set of controllers K_{l_κ} influenced by κ.
3. A link in the hierarchical control structure is formed between the nodes representing κ and each controller κ_i in K_{l_κ}, excluding the initial controller κ. The type of the link is determined by comparing the controllers' position in the logical hierarchy:
   - If κ is higher than κ_i a link of type 'ControlAction' is added.
   - If κ is lower than κ_i a link of type 'Feedback' is added.
   - Otherwise, the link is of type 'SameLevelInputOutput'.

   The initial port in the traversal p is associated with the link.

4. The non-controller solutions in Σ_{l_κ} represent the set of non-controller solutions S_{l_κ} influenced by κ.
5. If $S_{l_\kappa} \neq \emptyset$, which represents the case where non-controller solutions are directly controlled by κ, a link of type 'ControlAction' is added between the node representing κ and each process node π_i that includes a solution in S_{l_κ}.
6. Traverse the logical view starting from p including every component that can be reached by following flow links in the opposite direction to the flow. The traversal through a particular link ends when a controller solution is found, immediately after including such controller. This forms the set of components $\Sigma_{l'_\kappa}$ that influence κ.
7. The controller solutions in $\Sigma_{l'_\kappa}$ represent the set controllers $K_{l'_\kappa}$ that influence κ.
8. A link is formed in the hierarchical control structure between the nodes representing κ and each controller κ_i in $K_{l'_\kappa}$. The type of the link is determined by comparing the controllers' position in the logical hierarchy:
   - If κ_i is higher than κ a link of type 'ControlAction' is added.
   - If κ_i is lower than κ a link of type 'Feedback' is added.
   - Otherwise, the link is of type 'SameLevelInputOutput'.

   The initial port in the traversal p is associated with the link.

9. The non-controller solutions in $\Sigma_{l'_\kappa}$ represent the set of non-controller solutions $S_{l'_\kappa}$ that influence κ.
10. If $S_{l'_\kappa} \neq \emptyset$, a link of type 'Feedback' is added between the node representing κ and each process node π_i that includes a solution in $S_{l'_\kappa}$.

Algorithm 2: Creation of links from/to controllers

```
1  Function ADD-LINKS-FROM-CONTROLLER(logical-view, process-nodes, H, κ)
       Inpus : The logical view of the system (logical-view).
               The desired grouping for process nodes (process-nodes).
               The hierarchical control structure (H).
               A controller in the logical view (κ).
2      for each p ∈ GET-PORTS(κ)
3          Σ_{I_κ} ← TAVERSE-FLOW-WISE(logical-view, p) // Set of solutions
            influenced by κ
4          K_{I_κ} ← GET-CONTROLLERS(Σ_{I_κ}) // Controllers in Σ_{I_κ}
5          for each κ_i ∈ K_{I_κ}
6              if IS-HIGHER(logical-view, κ, κ_i) then
7                  ADD-CONTROL-ACTION-LINK(H, κ, κ_i, p)
8              else if IS-LOWER(logical-view, κ, κ_i) then
9                  ADD-FEEDBACK-LINK(H, κ, κ_i, p)
10             else
11                 ADD-SAME-LEVEL-LINK(H, κ, κ_i, p)
12             end
13         end
14         S_{I_κ} ← Σ_{I_κ} \ K_{I_κ} // Non-controller solutions in Σ_{I_κ}
15         if S_{I_κ} ≠ ∅ then
16             for each π_i ∈ GET-PROCESS-NODES(S_{I_κ}, process-nodes)
17                 ADD-CONTROL-ACTION-LINK(H, κ, π_i, p)
18             end
19         end
20         Σ_{I'_κ} ← TAVERSE-COUNTER-FLOW-WISE(logical-view, p) // Set of
            solutions that influence κ
21         K_{I'_κ} ← GET-CONTROLLERS(Σ_{I'_κ}) // Controllers in Σ_{I'_κ}
22         for each κ_i ∈ K_{I'_κ}
23             if IS-HIGHER(logical-view, κ_i, κ) then
24                 ADD-CONTROL-ACTION-LINK(H, κ_i, κ, p)
25             else if IS-LOWER(logical-view, κ_i, κ) then
26                 ADD-FEEDBACK-LINK(H, κ_i, κ, p)
27             else
28                 ADD-SAME-LEVEL-LINK(H, κ_i, κ, p)
29             end
30         end
31         S_{I'_κ} ← Σ_{I'_κ} \ K_{I'_κ} // Non-controller in Σ_{I'_κ}
32         if S_{I'_κ} ≠ ∅ then
33             for each π_i ∈ GET-PROCESS-NODES(S_{I_κ}, process-nodes)
34                 ADD-CONTROL-ACTION-LINK(H, π_i, κ, p)
35             end
36         end
37     end
38 end
```

The function ADD-LINKS-FROM-NON-CONTROLLER, described in Algorithm 3, creates the links in the hierarchy, starting from or ending at a process node associated with a non-controller σ. The connections are determined by applying the following rules:

1. Traverse the logical view of the architecture starting from a port p in solution σ including every component that is directly linked to p by following flow links in the direction of the flow. This forms the set of components Σ_{I_σ} influenced by σ.
2. For each σ_i in Σ_{I_σ}, a link is formed in the hierarchical control structure between the process node that contains σ and the process node that contains σ_i. The link is of type 'SameLevelInputOutput'. Duplicated links are omitted.
3. Traverse the logical view starting from p including every component that is directly linked to p by following flow links in the opposite direction to the flow. This forms the set of components $\Sigma_{I'_\sigma}$ that influence σ.
4. For each σ_i in $\Sigma_{I'_\sigma}$, a link is formed in the hierarchical control structure between the process node that contains σ and the process node that contains σ_i. The link is of type 'SameLevelInputOutput'. Duplicated links are omitted.

Algorithm 3: Creation of links from/to non-controller solutions

```
1  Function ADD-LINKS-FROM-NON-CONTROLLER(logical-view, process-nodes, H, σ)
       Inpus : The logical view of the system (logical-view).
               The desired grouping for process nodes (process-nodes).
               The hierarchical control structure (H).
               A non-controller solution in the logical view (σ).
2      for each p ∈ GET-PORTS(σ)
3          σ_{I_σ} ← TAVERSE-FLOW-WISE(logical-view, p) // Set of solutions
            influenced by σ
4          for each π_i ∈ GET-PROCESS-NODES(σ_{I_σ}, process-nodes)
5              ADD-SAME-LEVEL-LINK(H, σ, π_i, p)
6          end
7          σ_{I'_σ} ← TAVERSE-COUNTER-FLOW-WISE(logical-view, p) // Set of
            solutions that influence σ
8          for each π_i ∈ GET-PROCESS-NODES(σ_{I'_σ}, process-nodes)
9              ADD-SAME-LEVEL-LINK(H, π_i, σ, p)
10         end
11     end
12 end
```

The analysis of the algorithm reveals that its asymptotic complexity is $O(P_K L + P_S)$, where P_K is the number of ports that belong to controllers, L is the number of links in the logical view (which represent the worst-case for a traversal originating in one port), and P_S is the number of ports that belong to non-controllers. Provided that all ports are connected, the values of P_K, P_S, and L might be of similar magnitude, which implies a running time that increases quadratically with respect to the number of links in the

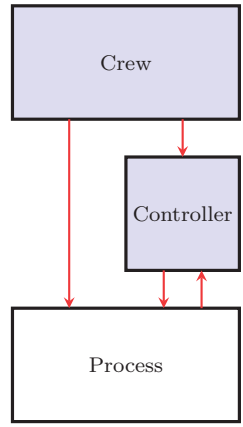


Figure 3. Example of a hierarchical control model.

logical view. This indicates that the worst-case asymptotic complexity is $O(L^2)$.

Figure 3 shows a simple example of a hierarchical control structure that is automatically created from the logical view of the system. The control hierarchy corresponds to the wheel brake system in Figure 1. The hierarchy is composed of controllers (solutions highlighted in blue) and a process node where the rest of the solutions are lumped together. Since the controller is lower than the crew in the solution hierarchy, this node is also situated lower in the control hierarchy. The hierarchy shows how the crew controls the process both directly and via the controller, which receives feedback from the process.

2.3. Creation of detailed control loops

Creating a detailed control loop requires the identification of four sets of components: the controller κ, actuators Σ_{A_κ}, sensors Σ_{S_κ} and process solutions Σ_{P_κ}. Given a controller, the developed method can automatically identify the actuators, sensors and process solutions. The process consists of the following steps:

1. Traverse the logical view starting from controller κ including every solution that can be reached by following flow links in the direction of the flow. This forms the set of components Σ_{I_κ} influenced by κ.
2. The set of solutions in Σ_{I_κ} with a direct flow link from κ corresponds to Σ_{A_κ}.
3. Traverse the logical view starting from controller κ including every solution that can be reached by following flow links in the opposite

direction to the flow. This forms the set of components $\Sigma_{I'_\kappa}$ that influence κ.

4. The set of solutions in $\Sigma_{I'_\kappa}$ with a direct flow link to κ corresponds to Σ_{S_κ}.
5. Finally, use the following equation to obtain the set of process solutions

$$\Sigma_{P_\kappa} = (\Sigma_{I_\kappa} \cap \Sigma_{I'_\kappa}) \setminus (\Sigma_{A_\kappa} \cup \Sigma_{S_\kappa} \cup \kappa)$$

The union of the four sets forms the set of components in the loop

$$\Sigma_{L_\kappa} = \Sigma_{I_\kappa} \cap \Sigma_{I'_\kappa} = \Sigma_{P_\kappa} \cup \Sigma_{A_\kappa} \cup \Sigma_{S_\kappa} \cup \kappa$$

In reality the analyst can also provide different values for Σ_{A_κ} and Σ_{S_κ}, which will override the values obtained automatically and remove components from Σ_{P_κ} as required. After the sets are identified, the method will classify the links among the various items in the control loop according to the links in the generic control loop provided in the STPA handbook (Leveson & Thomas, 2018, pp. 44, 49). These links cover the most common loss scenarios (unsafe controller behaviour, inadequate feedback and information, issues involving the control path and scenarios related to the controlled process). The method helps to produce the information required by STPA's Step 4 *Identify Loss Scenarios* with little input from the architect (only the controller κ).

In practice, some links do not fall under any of the categories proposed in Leveson and Thomas (2018, pp. 44, 49) such as links from other components (outside the loop) to sensors and actuators and between them. The proposed algorithm does not discard any link a priori and allows the analysts to decide which links are relevant for them. Figure 4 displays the extended generic control loop, which includes all possible kinds of links. These links are tagged with a number corresponding to the description of the links presented in Table 2. The latter contains the criteria employed for link classification and marks the additional kinds of links with an asterisk (*). A link is named using one of these two ways: (1) *Destination* Inputs from *Origin* or (2) *Origin* Outputs to *Destination*, both of which indicate that a link exists from *Origin* to *Destination*.

The analysis of the algorithm reveals that its asymptotic complexity is $O(L + S)$, where L is the number of links and S is the number of solutions in the logical view. The first summand appears because there are two traversals that will include every link in the architecture in the worst-case scenario. The second summand appears because every solution might be involved in the set operations. Set operations are performed with the help of hash tables, which yield an average time $O(1)$ to search for an element in a set (Cormen et al., 2009). In the case of the architectures employed in this research, where all ports are connected, the number of links is of similar

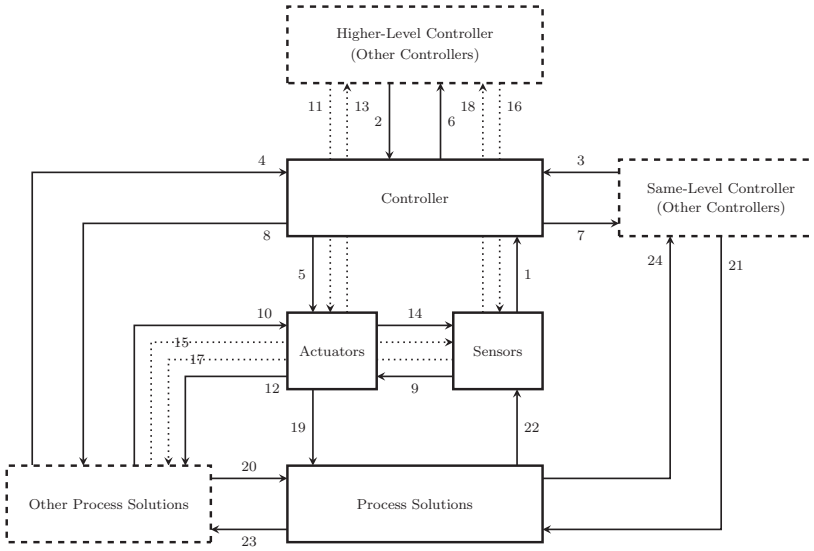


Figure 4. Links in the extended generic control loop.

magnitude to the number of ports and solutions. This brings the complexity to linear $O(L)$.

Figure 5 shows a simple example of the creation of a detailed control loop, which corresponds to the solution labelled as 'Controller'. It is created from the logical view of the wheel braking system in Figure 1. All solutions are classified according to their role in the control loop. Σ_{I_κ} is composed of the hydraulics, the wheel, the sensor and the controller. Σ_{A_κ} contains the hydraulics as they are the only component in Σ_{I_κ} with a direct link from κ. $\Sigma_{I'_\kappa}$ is composed of the crew, the sensor, the wheel, the hydraulics and the controller. Σ_{S_κ} contains the sensor as it is the only component in $\Sigma_{I'_\kappa}$ with a direct link to κ. Σ_{P_κ} corresponds to the wheel. The crew, which does not belong to the loop, is included in the figure as a direct influence on the controller.

3. Evaluation

Reported in this section are several studies which were performed to test and evaluate the two novel methods. The section starts with a brief introduction of AirCADia Architect, a prototype tool which incorporates the software implementation of the proposed techniques. This is followed by the description of the wheel brake use case (S-18 Aircraft and Sys Dev and Safety Assessment Committee, 1996) employed to evaluate the methods. Finally, the comparison between the automated and the manual application of STPA is discussed.

Table 2. Classification of control loop links.

Link type	From	To
(1) Controller Inputs from Sensors	A sensor in the loop Σ_{S_κ}	Controller κ
(2) Controller Inputs from Higher-Level Controllers	A controller in $\Sigma_{l'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to κ and higher than κ in the controller hierarchy	Controller κ
(3) Controller Inputs from Same-Level Controllers	A controller in $\Sigma_{l'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to κ and at the same levels as κ in the controller hierarchy	Controller κ
(4) Controller Inputs from Other Process Solutions	Solutions in $\Sigma_{l'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to κ and that are not controllers	Controller κ
(5) Controller Outputs to Actuators	Controller κ	An actuator in Σ_{A_κ}
(6) Controller Outputs to Higher-Level Controllers	Controller κ	A controller in $\Sigma_{l'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to κ and higher than κ in the controller hierarchy
(7) Controller Outputs to Same-Level Controllers	Controller κ	A controller in $\Sigma_{l'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to κ and at the same levels as κ in the controller hierarchy
(8) Controller Outputs to Other Process Solutions	Controller κ	A solution in $\Sigma_{l_\kappa} \setminus \Sigma_{A_\kappa}$ with a direct link to κ
(9) Actuators Inputs from Sensors*	A sensor in the loop Σ_{S_κ}	An actuator in Σ_{A_κ}
(10) Actuators Inputs from Other Process Solutions*	A solution in $\Sigma_{l'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to an actuator in Σ_{A_κ}	An actuator in Σ_{A_κ}
(11) Actuators Inputs from Other Controllers*	A controller in $\Sigma_{l'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to an actuator in Σ_{A_κ} and that is not a controller	An actuator in Σ_{A_κ}
(12) Actuators Outputs to Other Process Solutions*	An actuator in Σ_{A_κ}	A solution in $\Sigma_{l_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to an actuator in Σ_{A_κ} and that is not a controller
(13) Actuators Outputs to Other Controllers*	An actuator in Σ_{A_κ}	A controller in $\Sigma_{l_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to an actuator in Σ_{A_κ}
(14) Sensors Inputs from Actuators*	An actuator in Σ_{A_κ}	A sensor in Σ_{S_κ}
(15) Sensors Inputs from Other Process Solutions*	A solution in $\Sigma_{l'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a sensor in Σ_{S_κ} and that is not a controller	A sensor in Σ_{S_κ}
(16) Sensors Inputs from Other Controllers*	A controller in $\Sigma_{l'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a sensor in Σ_{S_κ}	A sensor in Σ_{S_κ}
(17) Sensors Outputs to Other Process Solutions*	A sensor in Σ_{S_κ}	A solution in $\Sigma_{l_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a sensor in Σ_{S_κ} and that is not a controller
(18) Sensors Outputs to Other Controllers*	A sensor in Σ_{S_κ}	A controller in $\Sigma_{l_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a sensor in Σ_{S_κ}

(*continued*)

Table 2. Continued.

Link type	From	To
(19) Process Inputs from Actuators	An actuator in Σ_{A_κ}	A solution in Σ_{P_κ}
(20) Process Inputs from Other Process Solutions	A solution in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a solution in Σ_{S_κ} and that is not a controller	A solution in Σ_{P_κ}
(21) Process Inputs from Other Controllers	A controller in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a solution in Σ_{S_κ}	A solution in Σ_{P_κ}
(22) Process Outputs to Sensors	A solution in Σ_{P_κ}	A sensor in Σ_{S_κ}
(23) Process Outputs to Other Process Solutions	A solution in Σ_{P_κ}	A solution in $\Sigma_{I_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a solution in Σ_{S_κ} and that is not a controller
(24) Process Outputs to Other Controllers	A solution in Σ_{P_κ}	A controller in $\Sigma_{I_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a solution in Σ_{S_κ}

3.1. Software prototype

AirCADia Architect (Guenov et al., 2020) is a research prototype tool for rapid model-based system architecting and evaluation. It has been extended for this research according to the object model proposed in Jimeno Altelarrea (2021). The new capabilities include, among others, support for the STPA hazard assessment methods presented in this paper. Additionally, AirCADia Architect allows the interactive explorations of the resulting STPA models. Once a part of the model is selected, the related parts and analysis results are highlighted.

3.2. Wheel brake system case study

The use case consists in the application of the two STPA methods to the Wheel Brake System (WBS) example proposed in the ARP 4761 (S-18 Aircraft and Sys Dev and Safety Assessment Committee, 1996). This example was employed in Leveson et al. (2014) to compare the manual application of STPA to commonly used traditional methods. To facilitate comparison with the results obtained by Leveson et al., the diagram, textual description and assumptions regarding the WBS stated in (Leveson et al., 2014, pp. 68–72) are used to populate the logical view of the architecture in AirCADia Architect. Although many STPA tools exist (see Section 1.2), none of these fully automates the creation of hierarchies and detailed control loops. Therefore, the comparison of the proposed methods with the manual approach is considered a representative comparison of the existing tools.

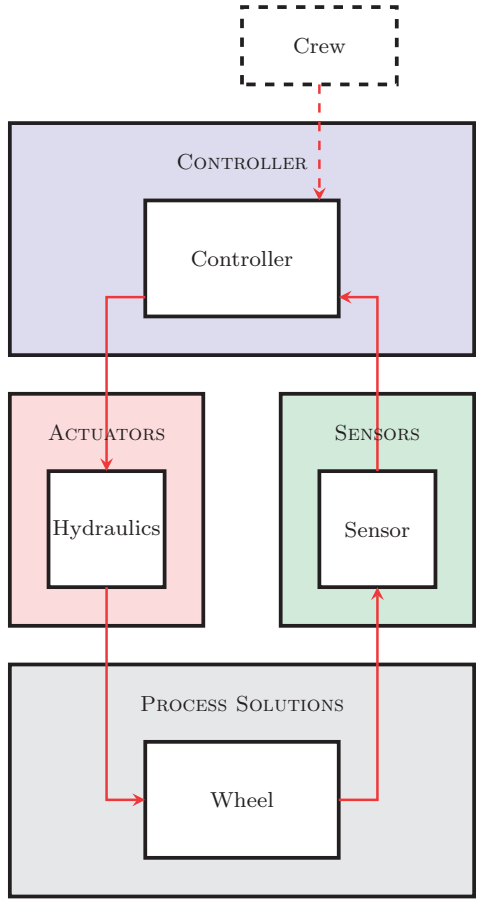


Figure 5. Example of a detailed control loop.

Figure 6 portrays the system implemented in AirCADia Architect. As shown in the diagram the system has two hydraulic channels (green and blue) capable of applying pressure on the brakes of the aircraft wheels. The hydraulic part of the system is controlled by the Brake System Control Unit (BSCU) and by the pedal's mechanical input. The controller uses the manual brake commands from the pedals as well as inputs from the rest of the aircraft such as wheel speed or auto-brake mode. Apart from outputting control commands for various hydraulic valves, it also communicates its status to the Brake System Annunciation. The annunciation informs the crew about the state of the system so they can make the necessary decision to control the WBS through manual brake (using pedals) or automatic brake commands. More details about the system can be found in the original report (Leveson et al., 2014). Some connections and components (such as the autobrake), which were only implicit in the original diagram, but

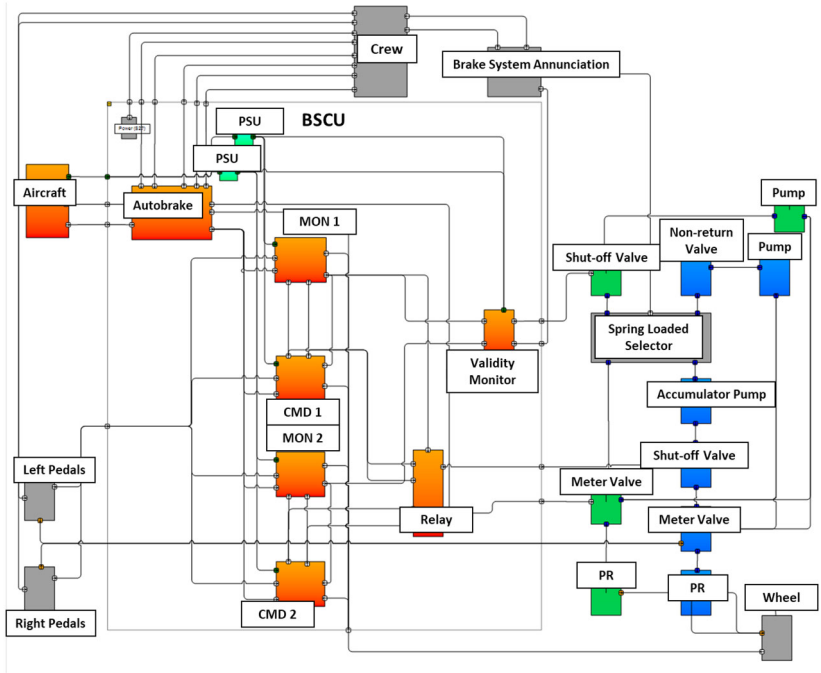


Figure 6. WBS systems architecture in AirCADia Architect.

provided in the system's description, have been made explicit in this implementation. This is because the tool and the methods require links to be included explicitly to work properly.

3.2.1. Creation of hierarchical control structures

The first part of the evaluation focuses on the method to automatically create the control hierarchies used for STPA analysis. Figure 7 shows the control hierarchy as automatically modelled by AirCADia Architect. The degree of similarity with respect to the control hierarchy obtained by Leveson et al. (2014, p. 28) is high regarding the components which are included and their position in the hierarchy. In both cases, the crew is situated at the top of the hierarchy. At an intermediate level, one can find the brake system control unit's (BSCU) controllers. Finally, the WBS hydraulics and wheel are located at the bottom of the hierarchy. It must be noted that the wheel has to be explicitly declared as a separate process node to make it appear as a separate node in AirCADia Architect. There small difference arises because the original hierarchy merges the two redundant control channels of the BSCU, whereas the automated method considers them individually.

There are also differences regarding the order of controllers within the BSCU. The original report omits the autobrake from the system diagram but

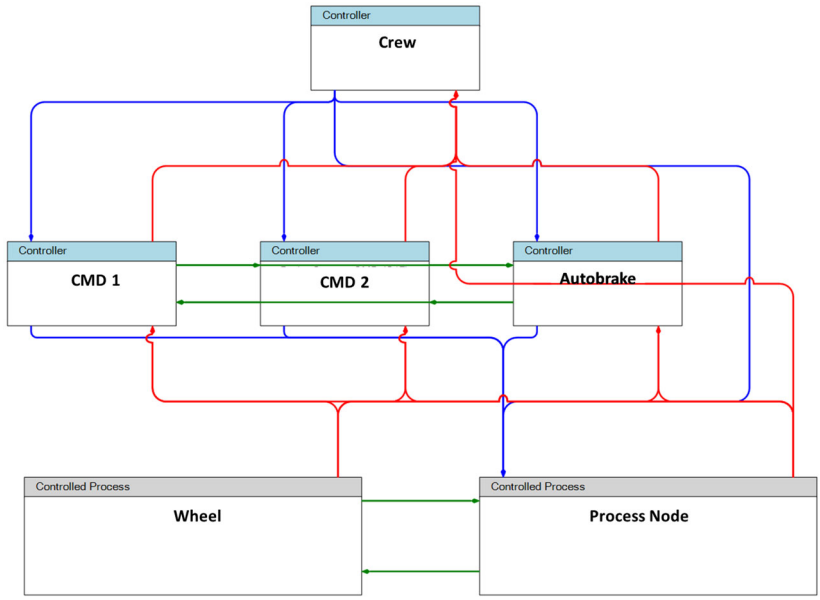


Figure 7. Control hierarchy obtained by AirCADia Architect.

situates it at a higher level than the rest of the BSCU controllers in the hierarchy. The autobrake had to be explicitly included in AirCADia Architect; placing it within the BSCU at the same level seemed the most appropriate option. As a result, the autobrake appears by default at the same level as the rest of the BSCU controllers in Figure 7. The wheels and the WBS hydraulics also appear at the same level in Figure 7. If desired, AirCADia Architect allows to change the type of the links (such as the link between the autobrake and the CMDs, or between the hydraulics and the wheel) to obtain an order identical to the original hierarchy.

Unlike the original hierarchy, Figure 7 does not show the description of the signals carried by the links, to avoid cluttering. Instead, this information is presented in a more convenient tabular format in Table 3. The table presents the comparison results regarding control commands (from a higher to a lower level in the hierarchy) and same level links in the hierarchy. The results show a considerable degree of similarity as the automated method discovers most of the signals and even some additional ones.

Nevertheless, there are some differences regarding which elements are identified by the creation algorithm as process nodes (every component that is not a controller). This includes the hydraulics and wheels, as in the original hierarchy, but also the aircraft and some BSCU components such as the monitors. This fact explains why the algorithm includes an additional command from the autobrake to the process, which corresponds to the

Table 3. Control commands in the control hierarchy.

From	To	Original signals	AirCADia signals
Crew	BSCU	Power on/off	Power on/off
Crew	Autobrake	Arm and Set	Arm and Set
		Disarm	Disarm
Crew	Hydraulic Controller	Brake (pedal)	Brake 1 & 2 (x2)
Crew	WBS Hydraulics	Brake (pedal)	Brake 1 & 2
Autobrake	Hydraulic Controller	Brake command	Control command (x2)
	Process Node	–	Control command
Hydraulic Controller	WBS Hydraulics	Open/Close green valve	–
		Green position command	Brake command
		Open/Close blue valve	Anti-skid command
Hydraulic Controller	Hydraulic Controller	–	Brake command (x2)
		–	Anti-skid command (x2)
WBS Hydraulics	Wheels	Braking Force	Braking Force (x2)
Wheels	WBS Hydraulics	–	Wheel Speed
Aircraft	Autobrake	Touchdown	–
		Rejected take-off	–

flow link between the autobrake and the 2 MONs. It also explains why 'Touchdown' and 'Rejected take-off' are not identified as command links. This is because both signals come from the process node, which is situated lower in the hierarchy, as opposed to the original hierarchy that depicts these links as downward links.

The fact that the STPA method considers the two CMD individually leads to the identification of two additional commands between CMDs ('Brake Command' and 'Anti-skid Command') not considered in the original analysis. This link between CMDs is found by traversing the hydraulics up to the wheel and coming back via the wheel speed feedback signal. The consideration of independent CMDs also leads to the double discovery of many of the command and feedback links, which has been indicated as (x2) in both Tables 3 and 4.

Table 4 presents the feedback signals in the control hierarchy (from a lower to a higher level). The explanations for discrepancies in the case of control commands are also applicable to feedback signals. All feedback signals in the original report are found by the method except the 'Wheel Speed' feedback signal from the hydraulic controller to the autobrake, which is found from the wheel to the autobrake instead. This is because the autobrake is not included in the original diagram of the architecture and neither are all of its inputs and outputs explained in the textual description of the system. Table 4 shows the two missing signals from Table 3, 'Touchdown' and 'Rejected take-off', which are identified as feedback. A few additional feedback signals are identified by the automated methods, namely 'Fault detected', 'Manual brake', 'Validity signal', and 'Electrical power'. Perhaps these links were neglected or not considered relevant for the hierarchy in the original report (Leveson et al., 2014).

Table 4. Feedback signals in the control hierarchy.

From	To	Original signals	AirCADia signals
Autobrake	Crew	Activated status	Activated status
		Armed status	Armed status
		Programmed deceleration	Programmed deceleration
		–	Fault detected
Hydraulic controller	Crew	Fault detected	Fault detected (x2)
WBS Hydraulics	Crew	Braking mode	Braking mode
Process	Crew	–	Fault detected
Hydraulic controller	Autobrake	Manual braking status	Manual braking status
		Wheel speed	–
Wheels	Autobrake	–	Wheel speed
Process	Autobrake	–	Touchdown
		–	Rejected take-off
Wheels	Hydraulic controller	Wheel speed	Wheel speed
Process	Hydraulic controller	–	Manual brake (x2)
		–	Validity signal (x2)
		–	Electrical power (x2)

3.2.2. Creation of detailed control loops

The second part of the STPA support evaluation focuses on the method to model automatically detailed control loops for STPA analysis. The crew was the chosen controller for the evaluation control loop, the brake pedals were identified as the actuators and the annunciation system is the 'sensor' providing feedback to the crew. The controller process is the wheel brake system, including the hydraulic controller and the hydraulics. Table 5 presents the comparison of the links obtained by both the original manual analysis (Leveson et al., 2014, p. 38) and the automated analysis by the methods proposed in this research. There is a high similarity regarding the components which belong to each part of the loop, namely the controller, actuators, sensors and controlled process.

The original loop is less detailed than the one provided in this research because of two reasons. The first one is because it did not provide any description for the links starting from or ending in the WBS. The second reason is that it does not distinguish between any of the two pedals or redundancies in the WBS. Because of this lack of detail, Table 5 shows a number of links that are not included in the original analysis. These links include 'Process Inputs From Actuators', 'Process Inputs From Other Process Solutions', 'Process Outputs To Sensors', and the second link in 'Controller Outputs To Actuators'. There are, however, a few links under the category 'Controller Inputs From Other Process Solutions' that appear in the original loop, but have not been identified by the proposed algorithm. This is because these links were not included in the diagram or the textual description of the architecture in the original document (Leveson et al., 2014).

Table 5. Signals in the crew control loop.

From	To	Original signals	AirCADia signals
Controller Inputs from Sensors			
Annunciation	Crew	Fault Detected	Fault Detected
Annunciation	Crew	Braking mode	–
Annunciation	Crew	Autobrake activated	Autobrake activated
Annunciation	Crew	Autobrake armed	Autobrake armed
Annunciation	Crew	Autobrake deccel. rate	Autobrake deccel. rate
Controller Inputs from Other Process Solutions			
Unspecified	Crew	Various signals	–
Controller Outputs to Actuators			
Crew	Pedals Left Seat	Manual Braking	Manual Braking
Crew	Pedals Right Seat	Manual Braking	Manual Braking
Process Inputs from Actuators			
Pedals Left Seat	Blue Metre Valve	–	Mechanical Position
Pedals Right Seat	Blue Metre Valve	–	Mechanical Position
Pedals Left Seat	CMD1	–	Brake Command
Pedals Right Seat	CMD1	–	Brake Command
Pedals Left Seat	MON1	–	Brake Command
Pedals Right Seat	MON1	–	Brake Command
Pedals Left Seat	CMD2	–	Brake Command
Pedals Right Seat	CMD2	–	Brake Command
Pedals Left Seat	MON2	–	Brake Command
Pedals Right Seat	MON2	–	Brake Command
Process Inputs from Other Process Solutions			
PSU	CMD1	–	Electrical Output
PSU	MON1	–	Electrical Output
PSU	CMD2	–	Electrical Output
PSU	MON2	–	Electrical Output
PSU	Validity Monitor	–	Electrical Output
PSU	Validity Monitor	–	Electrical Output
Process Outputs to Sensors			
Aircraft	AutoBrake	–	Touch Down Signal
Aircraft	AutoBrake	–	Rejected Take-Off Signal
Validity Monitor	Annunciation	–	Fault Signal
Spring Loaded Selector	Annunciation	–	Braking Mode Signal

3.2.3. Discussion of results

The comparison shows that the presented novel methods can reproduce the results obtained by manual application of the STPA methodology by Leveson et al. (2014). In fact, the methods are able to find additional links between elements, especially in the control loop. The automated approach was able to obtain this information and present diagrams and tables with link information with no or little input from the user. For example, in the case of the hierarchy, it was only necessary to indicate that the wheel belongs to its own process node. For the control loop, the crew was selected as the controller, while the pedals were selected as the actuators and the annunciation system as the sensor.

To compare the time and effort required by the manual and automated approaches, it is assumed that the process that converts the architecture into a graphical STPA model occurs in two steps. First, the architecture is analysed to determine which components to include and how to link them.

Second, the elements of the graphical model (such as nodes and links) are created and drawn on screen. The second step is assumed to be the same in both cases.

It is difficult to formalise the manual approach to analyse its asymptotic complexity. However, the difference regarding running time is clear. The execution time of the automated methods is in the order of milliseconds on a standard PC (including the debugger overhead), while the manual process rehearsed by the authors takes at least several minutes. This is an indication that the time difference between the manual and the automated approaches will be many orders of magnitude larger as the architecture complexity increases. If the preparation of the inputs (in this case the logical view) is too time consuming, it might offset the benefits of the automation. However, as explained in Section 2.1, it is reasonable to assume that a logical view of the system exists, as a result of the system architecting process. That is, the logical view is not created specifically for the STPA analysis. It can be used for other kinds of analysis such as sizing and performance and, therefore, the cost of creating it is spread out. Additionally, the presented methods ensure that the STPA analysis is updated at a low cost as the design advances.

Some minor discrepancies resulted from the differences between the manual and automated approaches (e.g. grouping of redundant solutions in the BSCU) and from the lack of consistency between the models presented in the original report by Leveson et al. (2014). As explained earlier in this section, the manual approach merged the redundant solutions in the BSCU in one single node in the STPA models whereas the automated method kept them separate. Although the manual approach enables additional functionality such as merging redundant components, it can also be a source of inconsistency as explained next. Considering the use case, the original diagram of the system did not include the autobrake or any of its links, but the textual description of the system did. Other links, such as the BSCU power on/off are only mentioned implicitly in the description. Furthermore, Leveson et al. (2014) included a link between the BSCU controller and the autobrake in the control hierarchy that is not present neither in the diagram nor in the description of the system. This is a good example of how the manual application can be a source of error and inconsistency, both between system definition models (textual and diagram) and between definition and STPA analysis. That is, if the STPA analysis is performed exclusively with the information from the description of the system it will not be possible to reproduce the results from Leveson et al. (2014) as the link between the BSCU controller and the autobrake is missing. Furthermore, other teams working on the system will not be aware of the existence of the link and be able to include it in their analysis. By contrast, the presented

methods ensure that the architecture is updated prior to obtaining the desired safety results, as safety results only change when the architecture is changed. If another team analysed the safety or any other characteristic of the architecture, they would be aware of the extra link, as it would have been necessarily included in the architecture.

4. Summary and conclusions

A review of the current state-of-the-art computational support methods for the creation of STPA models identified two main limitations: a significant amount of time required to create the models manually and the difficulty of ensuring consistency with the evolving architecture definition.

To overcome these limitations, two novel methods enabling the automatic creation of STPA models are proposed. The methods use the logical view of the systems architecture as input. The graphs formed by the hierarchical and flow relations in the logical view are utilised by the methods to create both hierarchical control structures and detailed control loops. These are used during STPA to determine unsafe control actions, loss scenarios and to derive safety requirements.

The proposed methods were implemented within a prototype object-oriented software, AirCADia Architect, to be tested and evaluated. The methods were applied to a wheel brake system use case that was previously used to compare the manual application of the STPA methodology to other hazard assessment methods. The results were compared with those provided by the manual application of STPA (Leveson et al., 2014). The comparison indicates that the proposed methods produce models that are almost identical to those obtained manually. Minor discrepancies were observed due to a different grouping of redundant solutions and an inconsistent system description. It was also demonstrated that the automatic process is orders of magnitude faster and requires less manual input. The automated approach was also found to ensure better consistency among the safety analysis and the architecture definition as it requires safety features to be included in the architecture definition prior to being considered in STPA analysis. In a wider context, this is seen as a step towards a more effective *design for safety (DfS)* process.

Future work, especially from an industrial perspective, will focus on exploring ways for efficient validation and verification of the presented methods. Another research avenue is the seamless integration with STPA methods and tools that automate other parts of the process, such as the method to automate the generation of formal model-based safety requirements (Thomas, 2013), or traditional safety methods such as FHA to utilise existing knowledge about functional hazards.

Disclosure statement

No potential competing interest was reported by the author(s).

Data availabilty statement

Data supporting this study are included within the article.

Notes on contributors

Sergio Jimeno Altelarrea PhD, is a Research Fellow in Aircraft Systems Architecting in Cranfield University. During his PhD he developed safety methods for the conceptual design of complex systems and contributed to the development of advanced Model-based Design tools such as AirCADia.

Atif Riaz PhD, CEng, MRAeS is a lecturer in Engineering Design at Centre for Aeronautics, Cranfield University, UK. His research interests focus on the development of methods and tools for complex systems design and analysis, utilising Model-Based Systems Engineering (MBSE), Multidisciplinary Design and Optimisation (MDAO), and Set-Based Design (SBD).

Marin D. Guenov PhD, CEng, FRAeS is Emeritus Professor in Engineering Design and Aerospace Engineering, and a former head of the Cranfield Aeronautics Centre. During his academic career he researched methods and led the development of advanced tools such as AirCADia for Model-based Design and Multidisciplinary Optimisation of Complex Systems.

ORCID

Sergio Jimeno Altelarrea http://orcid.org/0000-0002-1882-7183
Atif Riaz http://orcid.org/0000-0001-8383-8826

References

Abdellatif, A. A., & Holzapfel, F. (2021, January). The utilization of STPA techniques for system design safety enhancement. In *AIAA Scitech 2021 Forum*. American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2021-0565

Abdulkhaleq, A. (2014, March). *An open tool support for system-theoretic process analysis*. MIT, Boston. Retrieved May 25, 2021, from http://psas.scripts.mit.edu/home/wp-content/uploads/2014/03/Asim_A-STPA.pdf

Astah. (2021, February). *Astah system safety*. Retrieved March 31, 2021, from https://astah.net/products/astah-system-safety/

Björnsdóttir, S. H., & Rejzek, M. (2017, March). Embedding STPA into a highly successful risk management software application. In *6th MIT STAMP Workshop*. ZHAW Zürcher Hochschule für Angewandte Wissenschaften. https://doi.org/10.21256/ZHAW-3306

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.

Delange, J., & Feiler, P. (2014). Architecture fault modeling with the AADL Error-Model Annex. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, August (pp. 361–368). IEEE. https://doi.org/10.1109/SEAA.2014.20

European Aviation Safety Agency. (2020a). *Annual safety review 2020* (Tech. Rep.). Author. https://doi.org/10.2822/147804

European Aviation Safety Agency. (2020b). *Certification Specifications for Large Aeroplanes CS-25*. Retrieved May 25, 2021, from https://www.easa.europa.eu/certification-specifications/cs-25-large-aeroplanes

Federal Aviation Administration. (2000, December). *System safety handbook*. Retrieved March 15, 2021, from https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/risk_management/ss_handbook/

Guenov, M. D., Riaz, A., Bile, Y. H., Molina-Cristobal, A., & Heerden, A. S. (2020). Computational framework for interactive architecting of complex systems. *Systems Engineering 23*(3), 350–365. https://doi.org/10.1002/sys.21531

Gurgel, D. L., Hirata, C. M., & De M. Bezerra, J. (2015, September). A rule-based approach for safety analysis using STAMP/STPA. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)* (p. 7B2-1-7B2-8). https://doi.org/10.1109/DASC.2015.7311464

Information-technology Promotion Agency, Japan. (2018). *STAMP Workbench 1.0.1 Documentation*. Retrieved March 31, 2021, from https://www.ipa.go.jp/sec/stamp_wb/manual_en/tutorial/basic/basic.html

Jimeno Altelarrea, S. (2021). *Building safety into the conceptual design of complex systems. An Aircraft Systems Perspective*. [Unpublished doctoral dissertation]. Cranfield University.

Joshi, A., Vestal, S., & Binns, P. (2007). *Automatic generation of static fault trees from AADL Models* (Tech. Rep.). Retrieved from http://hdl.handle.net/11299/217313

Krauss, S. S., Rejzek, M., Reif, M. U., & Hilbes, C. (2016). 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. https://doi.org/10.21256/ZHAW-1175

Krauss, S. S., Rejzek, M., Senn, C. W., & Hilbes, C. (2016). SAHRA – An integrated software tool for STPA. In *4th European STAMP Workshop*. Zurich, Switzerland. https://doi.org/10.21256/zhaw-4926

Leveson, N. (2004). A new accident model for engineering safer systems. *Safety Science*, *42*(4), 237–270. https://doi.org/10.1016/S0925-7535(03)00047-X

Leveson, N. (2012). *Engineering a safer world: Systems thinking applied to safety*. MIT Press.

Leveson, N., & Thomas, J. (2018). *STPA Handbook* (Tech. Rep.). Retrieved March 12, 2021, from http://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf

Leveson, N., Wilkinson, C., Fleming, C., Thomas, J., & Tracy, I. (2014). *A comparison of STPA and the ARP 4761 Safety Assessment Process* (Tech. Rep.). Retrieved May 25, 2021, from http://sunnyday.mit.edu/STAMP/ARP4761-Comparison-Report-final-2.pdf

Mhenni, F., Nguyen, N., & Choley, J.-Y. (2014, July). Automatic fault tree generation from SysML system models. In *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics* (pp. 715–720). IEEE. https://doi.org/10.1109/AIM.2014.6878163

Papadopoulos, Y., McDermid, J., Sasse, R., & Heiner, G. (2001). Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering & System Safety*, *71*(3), 229–247. https://doi.org/10.1016/S0951-8320(00)00076-4

Patriarca, R., Chatzimichailidou, M., Karanikas, N., & Di Gravio, G. (2022). The past and present of System-Theoretic Accident Model And Processes (STAMP) and its associated techniques: A scoping review. *Safety Science*, *146*, 105566. https://doi.org/10.1016/j.ssci.2021.105566

Risk Management Studio. (2019). *STPA software solution – Risk management studio*. Retrieved March 31, 2021, from https://www.riskmanagementstudio.com/stpa-software-solution/#

S-18 Aircraft and Sys Dev and Safety Assessment Committee. (1996, December). *Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*. (Tech. Rep.). SAE International. https://doi.org/10.4271/ARP4761

S-18 Aircraft and Sys Dev and Safety Assessment Committee. (2010, December). *Guidelines for development of civil aircraft and systems* (Tech. Rep.). SAE International. https://doi.org/10.4271/ARP4754A

SE-Stuttgart. (2019, June). *XSTAMPP*. Retrieved March 31, 2021, from https://github.com/SE-Stuttgart/XSTAMPP

Shamal Faily. (2021). *CAIRIS 2.3.8 documentation*. Retrieved May 01, 2021, from https://cairis.readthedocs.io/en/latest/stpa.html

Souza, F. G., Pereira, D. P., Pagliares, R. M., Nadjm-Tehrani, S., & Hirata, C. M. (2019). WebSTAMP: A web application for STPA & STPA-Sec. *MATEC Web of Conferences*, *273*, 02010. https://doi.org/10.1051/matecconf/201927302010

Thomas, J. (2013). *Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis*. [Doctoral dissertation]. Massachusetts Institute of Technology. Retrieved May 01, 2021, from https://dspace.mit.edu/handle/1721.1/81055

Thomas, J., & Leveson, N. (2013). Generating formal model-based safety requirements for complex, software- and human-intensive systems. *Safety-Critical Systems Club*. Retrieved March 28, 2021 http://sunnyday.mit.edu/SSS-conference-stpa.pdf

Thomas, J., & Suo, D. (2015, March). *A tool-based STPA process*. Retrieved May 01, 2021, from http://psas.scripts.mit.edu/home/wp-content/uploads/2015/03/Thomas-Suo-Tool-based-STPA-process.pdf

VDI Department of Product Development and Mechatronics. (2004). *Design methodology for mechatronic systems (VDI 2206)*. Retrieved May 05, 2021, from https://www.vdi.de/richtlinien/details/vdi-2206-entwicklungsmethodik-fuer-mechatronische-systeme

Volpe National Transportation Systems Center. (2014). *SafetyHAT: A transportation system safety hazard analysis tool*. Retrieved May 01, 2021, from https://www.volpe.dot.gov/infrastructure-systems-and-technology/advanced-vehicle-technology/safetyhat-transportation-system



https://dspace.org/

STPA enabled safety assessment in the architecting of complex systems

Jimeno Altelarrea, Sergio; Riaz, Atif; Guenov, Marin D.

2022-12-09
Taylor & Francis
https://doi.org/10.1080/09617353.2022.2145647
Jimeno Altelarrea S, Riaz A, Guenov MD. (2022) STPA enabled safety assessment in the architecting of complex systems. Safety and Reliability, Volume 41, Issue 4, pp. 197-224

Downloaded from a DSpace repository. https://dspace.org/