Electronic Communications of the EASST
Volume 82 (2022)

11th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation

-

Doctoral Symposium, 2022

Tool Support for System-Theoretic Process Analysis

Jette Petzold and Reinhard von Hanxleden

xiii pages

Guest Editors: Sven Jörges, Salim Saay, Steven Smyth
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Tool Support for System-Theoretic Process Analysis

Jette Petzold and Reinhard von Hanxleden

{jep, rvh}@informatik.uni-kiel.de
Department of Computer Science
Kiel University, Kiel, Germany

Abstract: Hazard analysis techniques such as System-Theoretic Process Analysis (STPA) are used to guarantee the safety of safety-critical systems. Our goal is to improve the tool support for STPA. The preliminary result is the PASTA Visual Studio Code (VSCode) Extension that provides verification checks and diagrams. PASTA uses elkjs to layout the diagrams and Sprotty to draw them. We evaluate PASTA by recreating the ROLFER analysis. In the future we plan to further evaluate whether PASTA improves upon existing tools and to add more features such as reevaluation suggestions, model checking, and support for other risk analysis techniques.

Keywords: Automatic Visualization, Software Safety, STPA, VSCode Extension

1 Introduction

System-Theoretic Process Analysis (STPA) is a relatively new hazard analysis technique for safety-critical systems, based on the System-Theoretic Accident Model and Processes (STAMP) [Lev04]. It is applied manually by the safety analyst and consists of four phases. The results are a control structure and components of seven different aspects, which are: Losses, hazards, system-level constraints, responsibilities, unsafe control actions (UCAs), controller constraints, and scenarios. Components of these aspects reference each other as seen in Fig. 1 [LT18]. Hazards reference the losses they cause, system-level constraints are defined for hazards, UCAs lead to hazards, and so on. The control structure is composed of controllers, controlled systems, control actions, and feedback. Extensions of STPA have a similar approach. One example is STPA for Security (STPA-Sec) [YL13], which goal is to ensure security instead of safety. In this extension, the analyst defines vulnerabilities instead of hazards and Unsecure Control Actions instead of unsafe control actions. In contrast to traditional techniques, such as Fault Tree Analysis (FTA) [RS15], STPA considers not only component failures but also unsafe interactions among system components. That is why STPA identifies more risks than traditional techniques [Lev16]. However, STPA takes more time to apply than these traditional techniques, which is why the need for tools that provide systematization and automation has been widely acknowledged [SPP+19]. One project that would benefit from such a tool is Förde 5G[1]. It is a project of the Clean Autonomous Public Transport Network (CAPTN)[2] initiative with the goal to develop an unmanned autonomous ferry for the Kieler Förde. Since a ferry is a safety-critical system, risk analysis must be performed to verify the safety of the system and it is planned to use STPA for this.

[1] https://captn.sh/foerde-5g/
[2] https://captn.sh/

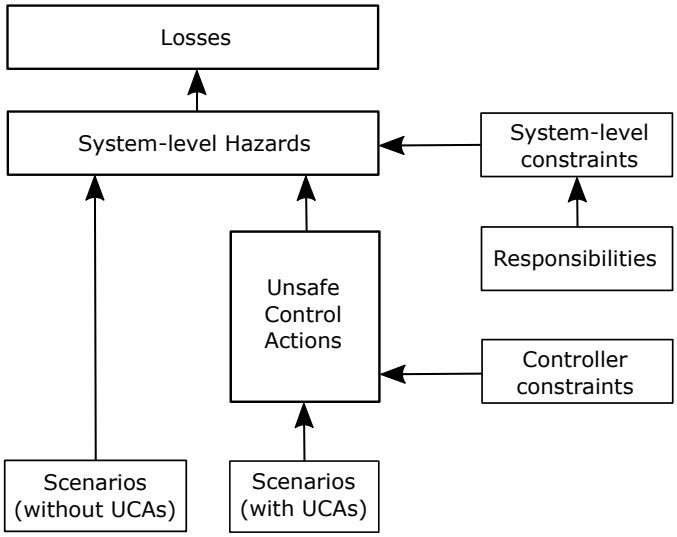


Figure 1: The STPA aspects and their connections as visualized by Leveson et al. [LT18]. A component belonging to an aspect that is the origin of an edge must reference a component belonging to the target aspect of this edge.

We want to improve the support for applying STPA on safety-critical systems by harnessing the benefits of automatically generated diagrams and automatic layout in the STPA context. Textual graph definitions provide benefits in comparison to Drag-and-Drop such as version control and faster diagram creation [FH10]. Since the target group, the analysts, may be more familiar with the creation of diagrams via Drag-and-Drop, a way must be found to introduce textual graph definition. Besides the diagram for STPA, it may also be useful to support other analysis techniques in the same tool. Antoine states that hardware issues found with STPA could be further analyzed with traditional techniques, whose focus lies on these issues [Ant13]. According to him, FTA would be the best candidate for this. This is why, we plan to look further into a possible combination of STPA with FTA.

Sec. 2 presents related work. The preliminary results covered in Sec. 3 are the Pragmatic Automated System-Theoretic Process Analysis (PASTA) Visual Studio Code (VSCode) extension and its features. Sec. 4 introduces the used technologies and planned evaluation strategies. The main contributions are concepts that are planned to be finalized and implemented in the future, which are presented in Sec. 5. Finally, Sec. 6 concludes with a summary.

2 Related Work

Tools that support the application of STPA already exist. STPA based Hazard and Risk Analysis (SAHRA) is a graphical editor which provides graphical elements for STPA components [KRR16]. An extract of an example diagram can be seen in Fig. 2. A component is represented as a rectangle containing an ID, a description, and an icon in the upper-right corner representing the aspect. The connections between components are represented by arrows, and additional

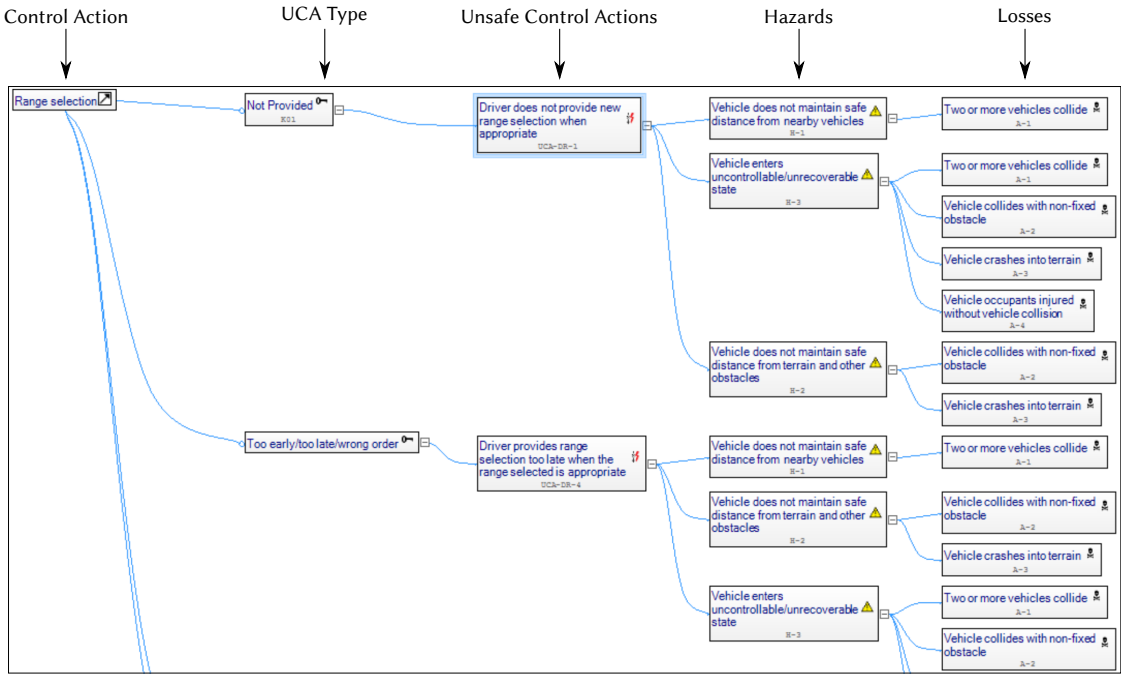


Figure 2: An extract of an example diagram in SAHRA [KRR16].

element types are used for the control structure. SAHRA provides a mind map style that helps to see relationships at a glance and provides flexibility regarding documentation details [KRSH16]. However, unlike the work presented here, SAHRA does not provide automatic layout and therefore requires potentially tedious and time-consuming manual layout. Furthermore, SAHRA does not provide filtering options that would improve the navigation for large diagrams.

The Extensible STAMP Platform (XSTAMPP), which is open source and based on the Eclipse Rich Client Platform (RCP), provides a graphical editor for creating the control structure [AW16b]. The other aspects of STPA are maintained in several views and tables. Furthermore, a subset of the STPA process is automated. The authors also developed a plug-in called Extended Approach to STPA (XSTPA), which implements context-tables proposed by Thomas [Tho13]. These context-tables systematize and partly automate the identification of UCAs. Based on the results of that phase and Safe Behavioral Models (SBMs), modeled manually by the user in another tool e. g. Simulink[3], test-cases can be generated automatically [AW16a]. Additionally, model checking is provided for these models. The model checker uses Linear Temporal Logic (LTL) formulas that are automatically generated based on the UCAs. Failing checks are displayed together with a textual counterexample. In contrast to SAHRA and the tool we develop, XSTAMPP has no visualization of the STPA aspects. This feature can help to get a better overview, which makes seeing connections between components easier. Furthermore, the counterexamples XSTAMPP provides during model checking can be improved by simulating them in the SBM.

[3] https://www.mathworks.com/products/simulink.html

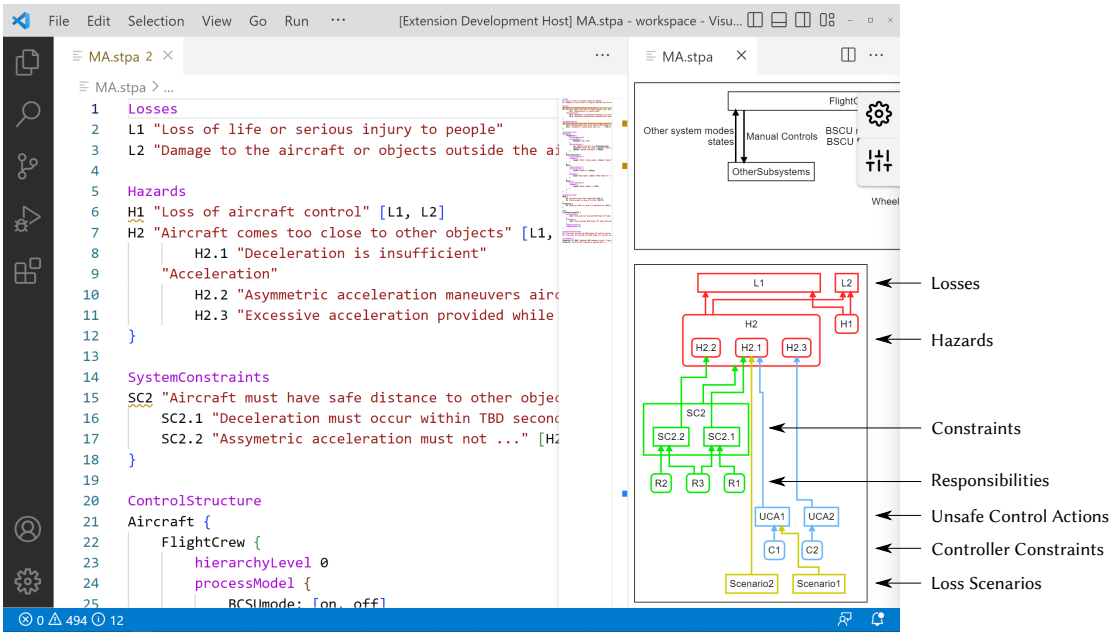


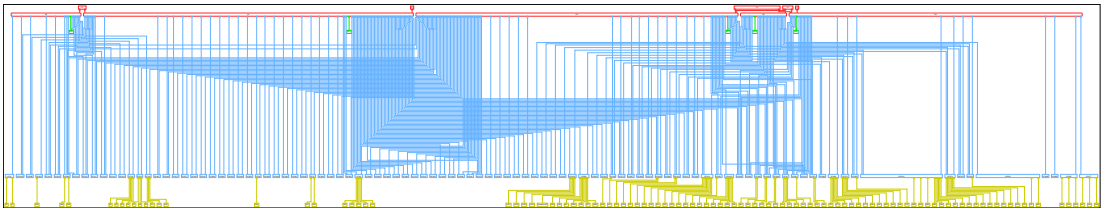
Figure 3: The PASTA VSCode extension.

An example of a web application is WebSTAMP [SPP+19]. It provides a web-based collaborative environment, security, and (context) tables for the STPA aspects. However, it focuses on only two of the four STPA steps. UCAs and scenarios can be defined but losses, hazards, responsibilities and the control structure must be modeled elsewhere. In contrast, PASTA aims to support the entire STPA process.

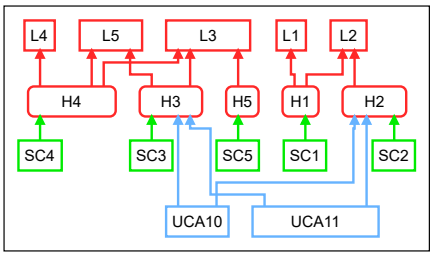
3 Preliminary Results

So far we concentrated on how visualizations can help when performing STPA. The result is a Domain Specific Language (DSL) for STPA as a VSCode Extension that provides the option to automatically generate a diagram [PKH23]. The syntax of the DSL is based on the notation used in the STPA handbook [LT18] to make access for safety analysts as easy as possible. An overview of the PASTA extension can be seen in Fig. 3. On the left is the editor in which the components of the STPA aspects and the control structure can be defined. On the right, the automatically generated and laid out diagram is shown.

Most of the STPA components should be referenced by another component. With a large number of components, unreferenced components are not directly obvious. Highlighting these components improves the overview of the analysis completeness. The DSL does this by showing a warning in the editor [PKH23]. Another verification check verifies that for each control action, at least one UCA is defined to ensure that the user does not overlook an action. These checks can be enabled or disabled by the user.



(a) Unfiltered Diagram.



(b) Filtered Diagram.

Figure 4: Effect of filtering on a very large diagram. In this case, filtering is done based on the control action of the UCAs.

The purpose of the two generated graphs is to provide an overview and help the user to better understand the relationships. However, the graph for the STPA aspects can grow to a considerable size. In order to still provide a clearly structured diagram, the STPA aspects should be distinguishable. For this purpose, each aspect of STPA has its own row in the graph and different colors are used (Fig. 3). The visualization can help to inspect specific components. The component the user is interested in and the connected components can be highlighted. PASTA implements this by highlighting selected nodes and their connected nodes. A further provided option, to focus on the currently important components, is the ability to filter components based on their aspects and filtering within an aspect. These filters are also helpful when a large number of STPA components are defined. An example application of filtering within the UCAs is shown in Fig. 4. With a growing number of components, the diagram also grows in size and hence the overview the diagram is supposed to give is impaired as can be seen in Fig. 4a. Using the filters leads to a smaller diagram (Fig. 4b), which can still provide a good overview of the currently important components.

Besides the diagrams, another view can be opened showing a context table as seen in Fig. 5 [PKH23]. Context tables are generated to guide the identification of components of a specific STPA aspect: unsafe control actions (UCAs). UCAs are divided into four types: provided, not provided, wrong timing, and applied too long or stopped too soon. In the context table view, the user can select for which control action and type the context table is shown. When clicking on a cell in the table that contains a UCA, the corresponding component in the editor and in the diagram is highlighted making the connection between the views easier.

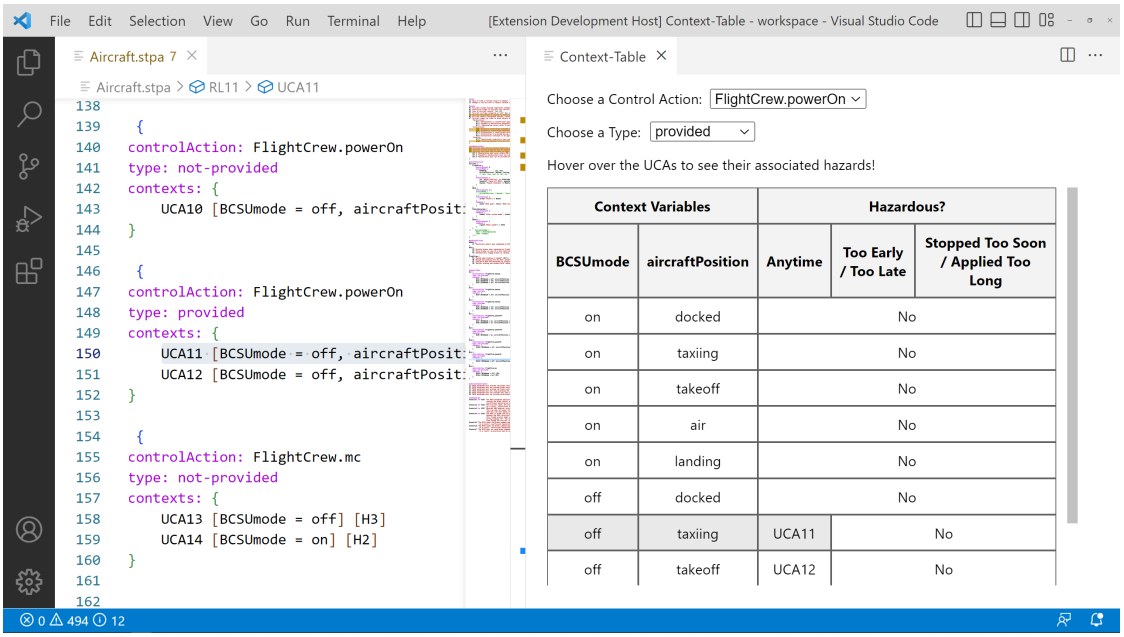


Figure 5: The context table in PASTA.

In order to simplify the access to text-based graphs, diagram snippets are provided for the control structure. They are shown in a menu in the activity bar as seen in Fig. 6. A diagram snippet is a textual control structure definition for which a preview is generated that shows how it will be visualized. Clicking on one of them inserts the corresponding textual definition in the editor, which adds the selected diagram as a subgraph to the control structure. This way users unfamiliar with the syntax of the control structure can easily construct the desired graph and can learn the syntax. It is also possible to create new snippets by selecting the corresponding textual definition, opening the context menu, and selecting the action to add a snippet. This can reduce the time needed to construct control structures, especially if several project analyses contain control structures with identical substructures.

4 Methods

PASTA[4] is open-source and part of the Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER)[5] project. It uses elkjs[6], the javascript version of Eclipse Layout Kernel (ELK)[7], to layout diagrams automatically. We use the open-source diagram framework Sprotty[8] to draw the laid out diagrams. Sprotty is well suited for our use case because it has built-in support for

[4] https://marketplace.visualstudio.com/items?itemName=kieler.pasta
[5] https://github.com/kieler
[6] https://github.com/kieler/elkjs
[7] https://www.eclipse.org/elk/
[8] https://github.com/eclipse/sprotty

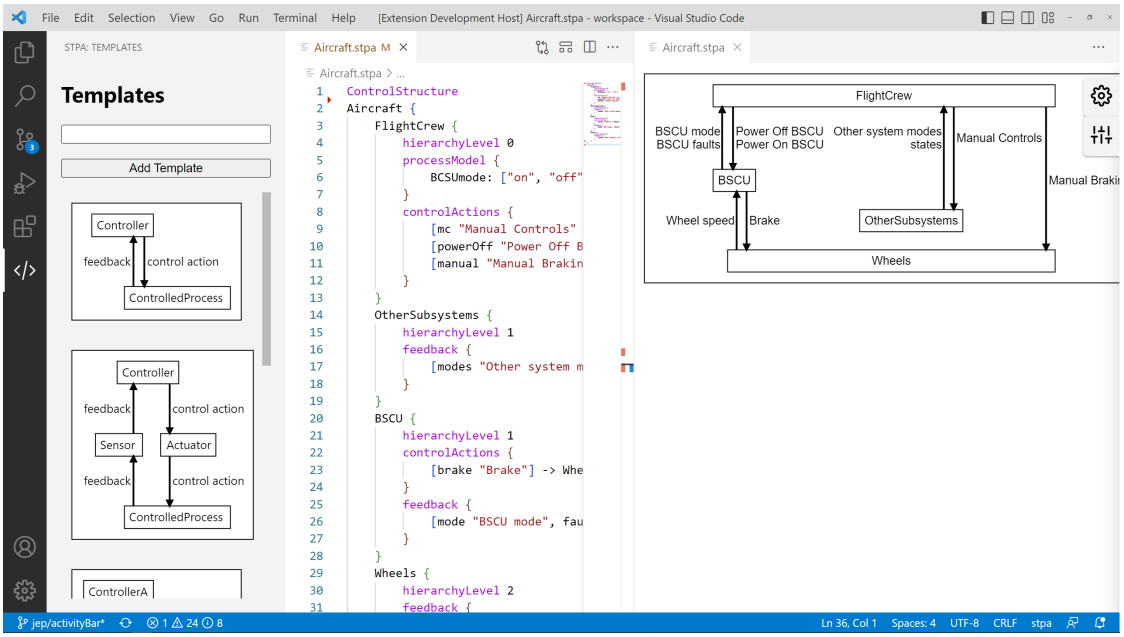


Figure 6: The diagram snippets in PASTA.

elkjs. The DSL in PASTA is defined with Langium[9] since it already provides an integration with Sprotty. Langium already provides language support such as renaming and auto-completion as well as default services such as a scope provider and validator which can be further customized. The verification checks provided by PASTA are realized by implementing a custom validator. In order to create a diagram, we extended the `DiagramGenerator` provided by Langium. The diagram generator takes an Abstract Syntax Tree (AST) as input and creates an `SGraph`. Sprotty provides the SGraph structure and default rendering for it. We defined custom rendering to visualize the diagram as described in the previous section. The different views for the diagram, context table, and diagram snippets are generated using VSCode webviews.

Evaluation of the concepts for the STPA support can be done by using PASTA in the Förde 5G context. This way, analysts can give feedback regarding the workflow and the implemented concepts. An exemplary analysis was done by using the aircraft example of the STPA handbook [LT18], which showed that the whole STPA process is supported. Furthermore, we used PASTA to recreate the entire Robotic Lifeguard For Emergency Rescue (ROLFER) analysis [CLD+20]. In addition to the support of all defined components, the recreation revealed that filtering is needed since the graph can grow to a considerable size [PKH23]. A comparison to other tools was already done based on the suggestions of Ludvigsen [Lud18] and Souza et al. [SPP+19]. It revealed that the main advantage of PASTA is the visualization of the relationships between the STPA components and that our tool can keep up with XSTAMPP, which fulfills the most suggestions of the compared tools [PKH23]. However, a full user study should be conducted in which the participants use XSTAMPP as well as PASTA to apply STPA.

[9] https://langium.org/

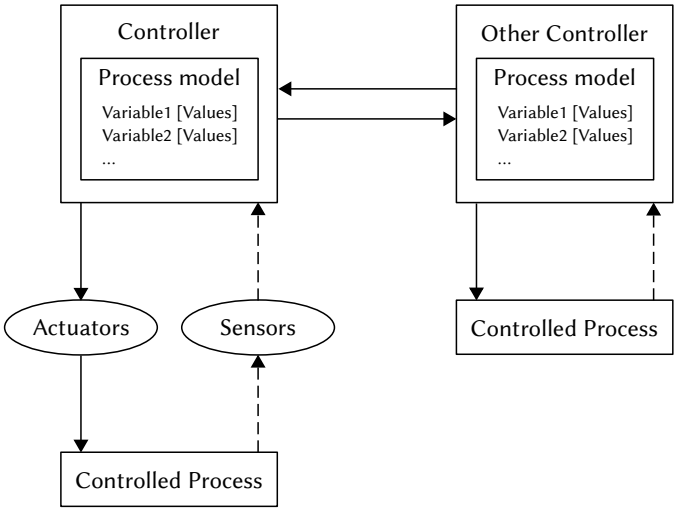


Figure 7: The planned visualization of the control structure in PASTA. The nodes represent system components while the edges represent communication between them.

5 Next Steps

We plan to include reevaluation suggestions as proposed by Ludvigsen [Lud18]. STPA does not need to be a linear process, so if components are adjusted in later steps, reevaluation suggestions can show which other components need to be updated as well. This way inconsistencies that emerge because not all affected components are updated could be reduced. We plan to inspect how such suggestions can be realized and to what extent visualization can help. One possible realization is a workflow similar to renaming a component: The user right-clicks on the component that should be changed and selects the *reevaluate* action. Subsequently, the component can be edited and affected components highlighted in the editor or diagram. An option to jump to the next affected component in the editor may be useful as well.

Furthermore, the DSL will be evaluated, especially the usefulness of the visualization. Additionally, we intend to improve the diagram further. For example, the layout of the control structure can be adjusted to be more similar to manually drawn ones. This may lead to better orientation in the graph as it is more in line with the mental map of the user. The orientation and overview can be improved even further by drawing actuators and sensors differently from other system components and using dashed lines for feedback transitions. The process model of a controller, which contains information about other system components and the environment, can be visualized as well. The resulting visualization is shown in Fig. 7. In order to evaluate these concepts, we plan to conduct a survey.

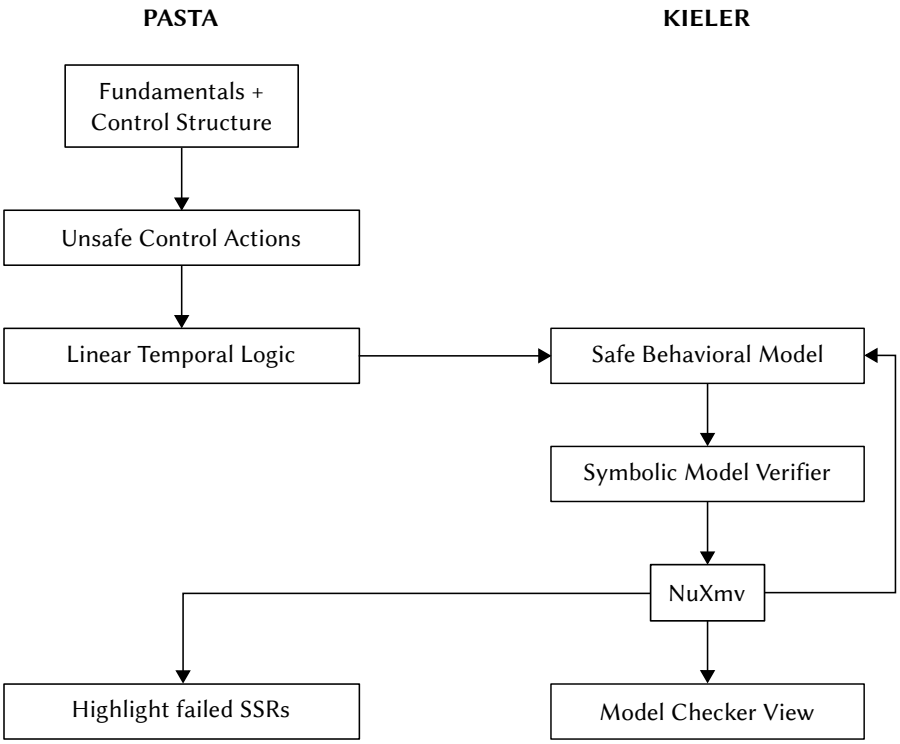


Figure 8: The workflow for model checking with PASTA.

5.1 Model Checking

Moreover, we plan to adapt the verification and test-case generation done by Abdulkhaleq et al. [AW16a] into KIELER and combine it with the DSL. Abdulkhaleq et al. use the context tables to automatically derive Software Safety Requirements and translate them to LTLs formulas. These formulas can be used to verify the SBM of a software controller of the analyzed system. The SBM is modeled manually by the user and is automatically transformed into a Symbolic Model Verifier (SMV) model. The generated LTL formulas are added to this SMV model and the NuSMV model checker is used to verify the correctness of the SBM regarding the formulas. When the model is correct, a safe test model — an Extended Finite State Machine (EFSM) — is constructed automatically based on the SBM. The EFSM is then used to create test cases. This verification and test case generation is implemented in XSTAMPP. We plan to also provide such model checking with PASTA. The planned workflow can be seen in Fig. 8. The STPA aspects, including the UCAs, are defined by the user in PASTA. Based on the context tables, LTL formulas are generated using Abdulkhaleq et al.'s rules. However, not all UCA types are considered by these rules. It is planned to create rules for the missing types.

The SBM that Abdulkhaleq et al. expect to be modeled elsewhere, will be modeled in KIELER. KIELER already supports a translation to an SMV model and model checking of LTL formulas. The model checking is done by using NuXmv [CCD+14]. In order to use the LTL formulas that are generated by PASTA, KIELER must communicate with PASTA. Since both tools are VSCode

Extensions, this will be done with the Language Server Protocol (LSP). In contrast to XSTAMPP, KIELER can simulate the counterexamples of failed LTL formulas in the SBM, which may help the user to better understand why a formula is not fulfilled.

Additionally, failed LTL formulas are planned to be tracked back to their UCAs to highlight them in the editor and visualization in PASTA. For this, again, the LSP will be used. The finished SBM can be used to generate safe code for the modeled controller. KIELER provides different methodologies and target languages for such a code generation. Furthermore, we want to explore in what detail an SBM can be generated automatically based on the STPA analysis. If a completely automatic generation of an SBM is possible, an STPA analysis would be sufficient to generate code for software controllers. Even a partly automatic generation of an SBM can help to reduce the time needed for modeling the behavior of a controller.

Additionally, we plan to use the SBM to generate scenarios in the OpenSCENARIO[10] format because these scenarios can be used for a simulation in the Unreal Engine[11]. With such a simulation the behavior of a system can be verified. For the Förde 5G project, such a simulation of the autonomous ferry exists and it is planned to use OpenSCENARIO to test the ferry. In order to create the scenarios, at first test cases must be generated. For that, we can create random values for the input variables similar as Abdulkhaleq et al. do. However, to create scenarios in the OpenSCENARIO format, it is not sufficient to only know the behavior of the modeled system in each scenario. In the case of the ferry in the Förde 5G project, we also have to state the behavior of other vessels in each scenario. Hence, we must define a way how the user can specify the environment, such as the number of additional vehicles, their maximal velocity, their mobility, etc. We plan to investigate which parameters are needed and how these parameters can be used to create scenarios in which several vessels participate.

5.2 FTA Integration

We want to support other risk analysis techniques in PASTA and if possible combine them with STPA. For now, we chose FTA for this, since Antoine states that it is the best candidate for a combination with STPA [Ant13]. In FTA the user draws a tree starting with the component failure as the top event. Subsequently, the events leading to the top event are added using one of the following connectors: AND gate, OR gate, k/n gate, or INHIBIT gate. An example tree can be seen in Fig. 9 in which *G1* is an AND gate and *G2* an OR gate. If the events *E1* and *E2* occur and one of the events *E3* or *E4*, the top event *DS1* occurs. To integrate FTA in PASTA a new grammar must be defined in PASTA. Just as for STPA this can be done with Langium. Besides a new grammar, we need to define a diagram generator which translates an FTA AST to an SGraph. This graph can then be visualized by defining custom rendering similar to how it is done for STPA. Once PASTA provides such a FTA DSL, we will look into the combination of STPA with FTA. For example, an option can be provided to import component failure events detected with STPA to an FTA file. We plan to investigate to what extent a fault tree can be automatically generated for such a component failure based on the information given in the STPA file.

[10] https://www.asam.net/standards/detail/openscenario/v200/
[11] https://www.unrealengine.com/de

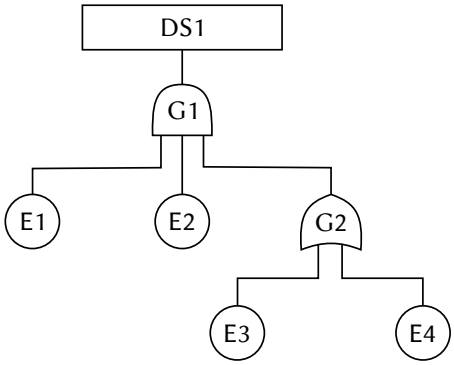


Figure 9: An example Fault Tree.

6 Conclusion

In conclusion, a basic foundation to improve the support for STPA is already implemented in the PASTA VSCode Extension. PASTA provides a DSL, automatic validation checks, automatically generated diagrams, and context tables. A comparison based on the suggestions of Ludvigsen and Souza et al. revealed that PASTA can already keep up with XSTAMPP and has the advantage of the visualization of relationships between STPA components. In the future several features are planned to be finalized and implemented. These features include an expanded control structure definition and visualization as well as model checking and scenario generation based on the STPA result. For the latter, KIELER will be used. Furthermore, we plan to inspect how STPA can be combined with FTA.

Acknowledgements: This research has been partly funded by the Federal Ministry for Digital and Transport (BMDV) within the project "CAPTN Förde 5G".

Bibliography

[Ant13] B. 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.

[AW16a] A. Abdulkhaleq, S. Wagner. An Automatic Safety-Based Test Case Generation Approach Based on Systems-Theoretic Process Analysis. Technical report, University of Stuttgart, 2016.

[AW16b] A. Abdulkhaleq, S. Wagner. XSTAMPP 2.0: New Improvements to XSTAMPP Including CAST Accident Analysis and an Extended Approach to STPA. *2016 STAMP Workshop at Massachusetts Institute of Technology (MIT)*, Mar. 2016.

[CCD+14] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta. The nuXmv symbolic model checker. In *Computer Aided*

Verification: 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings 26. Pp. 334–342. 2014.

[CLD+20] S. Charalampidou, E. Lygouras, I. Dokas, A. Gasteratos, A. Zacharopoulou. A Sociotechnical Approach to UAV Safety for Search and Rescue Missions. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. Pp. 1416–1424. 2020.

[FH10] H. Fuhrmann, R. von Hanxleden. Taming Graphical Modeling. In *Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS '10)*. LNCS 6394, pp. 196–210. Springer, Oct. 2010. doi:10.1007/978-3-642-16145-2

[KRR16] S. S. Krauss, M. Rejzek, M. U. 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. Technical report, ZHAW Züricher Hochschule für Angewandte Wissenschaften, Dec. 2016.

[KRSH16] S. S. Krauss, M. Rejzek, C. W. Senn, C. Hilbes. SAHRA - An integrated software tool for STPA. In *4th European STAMP Workshop, Zurich, 13-15 September 2016*. 2016.

[Lev04] N. Leveson. A New Accident Model for Engineering Safer Systems. *Safety science* 42(4):237–270, 2004.

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

[LT18] N. Leveson, J. P. Thomas. STPA Handbook. *MIT Partnership for Systems Approaches to Safety and Security (PSASS)*, 2018. http://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf.

[Lud18] N. Ludvigsen. Prototyping a digital support tool for an agile implementation of STPA. Master's thesis, Norwegian University of Science and Technology, 2018.

[PKH23] J. Petzold, J. Kreiß, R. von Hanxleden. PASTA: Pragmatic Automated System-Theoretic Process Analysis. In *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2023.

[RS15] E. Ruijters, M. Stoelinga. Fault Tree Analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review* 15:29–62, 2015.

[SPP+19] F. G. Souza, D. P. Pereira, R. M. Pagliares, S. Nadjm-Tehrani, C. M. Hirata. Web-STAMP: a Web Application for STPA & STPA-Sec. In *MATEC Web of Conferences*. Volume 273. 2019.

[Tho13] J. P. Thomas. *Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis*. PhD thesis, Massachusetts Institute of Technology, 2013.

[YL13] W. Young, N. Leveson. Systems thinking for safety and security. In *Proceedings of the 29th Annual Computer Security Applications Conference*. Pp. 1–8. 2013.