Proceedings of the 35th European Safety and Reliability & the 33rd Society for Risk Analysis Europe Conference
Edited by Eirik Bjorheim Abrahamsen, Terje Aven, Frederic Bouder, Roger Flage, Marja Ylönen
 Published by Research Publishing, Singapore.
doi: 10.3850/978-981-94-3281-3_ESREL-SRA-E2025-P1440-cd

Making Systems of Systems Orchestration Safer

Julieth Patricia Castellanos Ardila, Nazakat Ali, Sasikumar Punnekkat and Jakob Axelsson
School of Innovation, Design and Engineering, Mälardalen University, Sweden.
E-mail: (julieth.castellanos, nazakat.ali, sasikumar.punnekkat, jakob.axelsson)@mdu.se

Orchestration, an approach to service composition, has emerged as a promising solution to integrate independent constituent systems (CS) in a System of Systems (SoS). However, safety in SoS orchestrations remains unexplored. In this paper, we introduce SOSoS (Safe Orchestration of Systems of Systems), a process that utilizes the System-Theoretic Process Analysis (STPA) steps extended with the features proposed in the software product line engineering (SPLE) approach to cope with safety in the inherent SoS variability. We also demonstrate SOSoS in action by considering a case study from the construction domain. As a result, we define SoS-level safety constraints that could lead to actionable technical recommendations for making systems of systems orchestrations safer.

Keywords: System of Systems, Orchestration, Safety Analysis, System-theoretic Process Analysis, Variability.

1. Introduction

A System of Systems (SoS) (ISO/IEC JTC 1/SC 7, 2019) is an assembly of constituent systems (CS) that, with individual capabilities, achieve a more general goal. SoS can be formed in different topologies (Maier, 1998) (Dahmann and Baldwin, 2008), i.e., directed and acknowledged SoS, which have centralized authority, and collaborative and virtual SoS, where CS loosely collaborate. The CS collaboration facilitate the formation of SoS that are diverse (Axelsson and Svenson, 2022), adaptable (Fisher, 2006), and reconfigurable (Axelsson and Kobetski, 2018). However, if not properly managed, the SoS may also have undesirable effects (Beland and Miller, 2007).

A promising solution for SoS management is orchestration (Klein and McGregor, 2013) and (Nordstrom et al., 2024)). Orchestration is a composition approach in which a central module collects, coordinates, and distributes services (Josuttis, 2007). For SoS, this module, called the orchestrator, is meant to be a service hub providing explicit instructions to CS in direct and acknowledged SoS. We also see the orchestrator as a neutral facilitator for collaborative and virtual SoS. However, safety in SoS orchestrations has not yet been explicitly considered. Thus, safety analysis approaches tailored to the unique characteristics of SoS orchestrations are imperative.

In this paper, we introduce a process for Safe Orchestration of Systems of Systems (SOSoS). In particular, SOSoS considers the steps proposed by the System-Theoretic Process Analysis (STPA) (Leveson and Thomas, 2018), an analysis method based on the System-Theoretic Accident Model and Processes (STAMP) (Leveson, 2016). STPA facilitates the management of safety risks in complex systems by considering the principles of systems thinking, i.e., interactions and control structures. However, the orchestrator is a platform with core characteristics, where the independent nature of CS leads to variable configurations (Botterweck, 2013). Thus, we extend the STPA analysis in a novel fashion to include such variability by using the features proposed in the software product line engineering (SPLE) approach (Pohl et al., 2005). As a final contribution, we demonstrate SOSoS in action with a case study from the construction domain. As a result, we define SoS-level safety constraints that can be used to embed a safety concept into the SoS orchestration at design time to enable confident SoS deployments.

This paper is organized as follows. Section 2 presents the required background. Section 3 introduces our process called SOSoS. Section 4 presents a case study to illustrate SOSoS in action. Section 5 presents a discussion of our findings. Section 6 presents related work. Finally, Section 7 presents conclusions and future work.

2. Background

2.1. *Variability in SoS*

An SoS orchestrator (Klein and McGregor, 2013) targets missions while integrating CS capabilities. During such integration, it also enable large-scale service reuse (Botterweck, 2013). For example, CS with different technology need to have means to communicate with the SoS via common interfaces. A mechanism for enabling flexible variant binding to a common core is SPLE.

2.2. *SPLE*

SPLE-Software Product Line Engineering (Pohl et al., 2005) is an approach to ensure that while shared components or their properties are standardized, their differences are intentionally planned to adapt to specific contexts. This approach comprises two distinct processes. First, the domain engineering, which is responsible for establishing the reusable platform, i.e., the commonality and the variability of the product line. Second, the application engineering, where specific products are derived from the platform, i.e., the variants. In summary, the strategy is to focus on identifying and leveraging three key aspects:

A1. **Commonalities.** Shared features that serve as the stable foundation of the system.
A2. **Variability.** Features that differ.
A3. **Variant.** Instances selected from the associated variability to form different configurations.

2.3. *STPA*

STPA-System Theoretic Process Analysis (Leveson and Thomas, 2018) is a safety analysis method that focuses on interactions and control structures. STPA comprises the following four steps:

S1. **Define the purpose of the analysis.** Describe the system of interest (SoI) together with potential losses, hazards and application scenarios.
S2. **Model the control structure.** The SoI is modeled considering the set of functional components together with their control loops.
S3. **Identify unsafe control actions (UCA).** Examine the control structure against the following control action type (CAT), i.e., a) A required control action (CA) *is not provided*, b) An unsafe CA *is provided*. c) A required CA is provided *too early* or *too late* or *out of sequence*. d) A required CA *is stopped too soon* or *applied too long*.
S4. **Identify loss scenarios.** Develop scenarios describing how a UCA occurs, considering factors like unsafe control input (UCI), inadequate control algorithm (ICA), inadequate process model (IPM) and inadequate or missing feedback (IF).

3. SOSoS

An SoS platform enables orchestration by providing services for the CS (see Section 2.1). However, poorly managed orchestrators may hide hazards that could impair SoS safety. To address this, we introduce SOSoS, a process for Safe Orchestration of Systems of Systems (see Fig. 1). SOSoS uses STPA steps (see Section 2.3), for safety analysis (tasks in blue), intertwined with SPLE aspects (see Section 2.2) for feature reasoning (tasks in pink). Verification tasks (depicted in white) are also suggested so that the system behavior can also be checked. SOSoS is an iterative process, i.e., it refines the orchestrator control logic and its associated safety concept from the verified model (see the blue arrows depicted in the figure). This iterative process also opens room for SoS evolution, i.e., when the orchestrator capabilities change, or new types of CS are incorporated. SOSoS is conformed by three activities as follows.

3.1. *Design*

In this activity, three tasks are performed to design the control logic of the orchestrator featuring the core functionality:

(a) **SoS Description.** Provide a concise overview of the SoS mission, focusing on the core purpose without going into technical or implementation details. The description shall include the losses to be avoided, the derived hazards, and associated safety requirements to facilitate the safety analysis, as proposed by STPA-S1. The output is a description of the System of Interest (SoI).
(b) **Core Identification.** Identify core features needed across all CS in the SoS, based on the SoI description, as proposed by the SPLE-A1. Core

features are SoS aspects that refer to the shared elements or properties common to all CSs, enabling them to work together to achieve the SoS mission. The output is the description of the orchestrator core features that is used to form the foundation for control logic.

(c) **Control Structure Modeling.** Create a control structure including the identified core features, as proposed by STPA-S2. The output is a structure that graphically describe how those features hierarchically align in the control logic.

3.2. *Safety Analysis*

In this activity, four tasks are carried out to define the orchestrator safety concept:

(a) **UCA Identification.** Examine the unsafe control actions (UCA) in the control structure arising from deficiencies, as proposed by STPA-S3. The output is a list of UCA, which describes the actions (or lack of actions) by the orchestrator that can lead to hazardous SoS states.

(b) **Causal Analysis.** Analyze the causal factors that could lead UCA to a loss scenarios, as proposed by STPA-S4. The output is a list of loss scenarios, which describes how the orchestrator could make the SoS transition into a unsafe state, resulting in a loss (e.g., workers' death).

(c) **Constraints Definition.** Establish constraints based on the list of loss scenarios previously identified. The output is a list of safety constraints which describe what shall or shall not happen to prevent hazardous conditions.

(d) **Variability Analysis.** Analyze variable features across CS, as in SPLE-A2, to include all possible SoS forms and their associated constraints. The output is a set of constraints targeting variability that can be used to refine the previously identified safety constraints.

3.3. *Configuration Verification*

In this activity, three tasks are performed to verify a model of the orchestrator by considering the associated control logic and the safety concept.

(a) **Variant Selection.** Select specific CS instances or variants, as proposed by the SPLE-A3, to concretize the control logic. The output is a set of SoS Configurations.

(b) **Behavior Modeling.** Represent the behavior of the identified SoS configurations. The output is a behavioral model of the SoS in the form of state machines or agent-based models.

(c) **Model Verification.** Perform procedures that permit dynamic verification of the behavioral models representing SoS configurations (e.g., execute model checking or simulations). The resulting test reports accounting for requirements satisfaction and counterexamples can be used for further analysis.

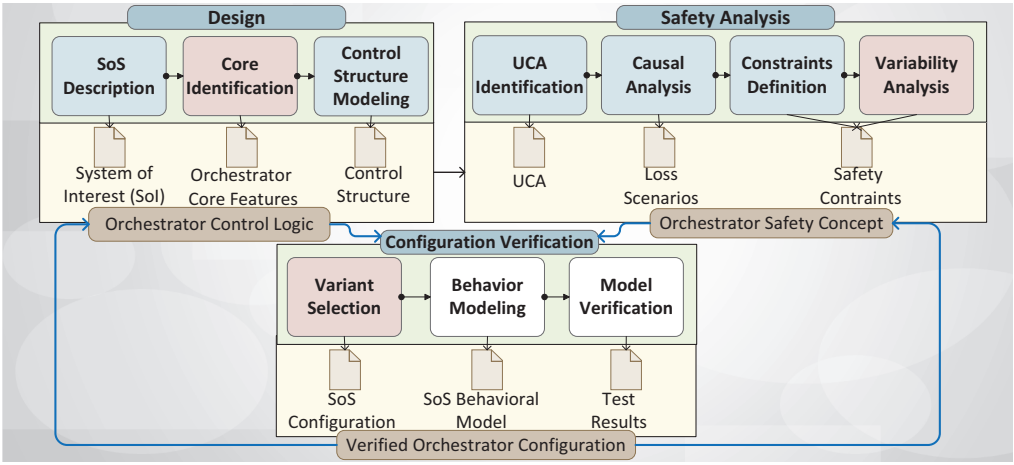

Fig. 1. SOSoS Process.

4. SOSoS in Action

In this section, we illustrate in detail the first two activities included in SOSoS, i.e., design and safety analysis (Sections 4.1 and 4.2) and briefly describe the safety verification (Section 4.3).

4.1. *Design*

This section presents the steps required to design the orchestrator control logic (see Section 3.1).

(a) **SoS description.** The mission of the SoS in this case study is to facilitate mass removal at a construction site. To achieve this mission, the orchestrator shall integrate various constituent systems (CS), such as excavators, trucks, and loaders. A critical aspect of this process is *task allocation*, an action where the orchestrator assigns specific tasks to CS based on operational needs. However, during task execution, CS may operate in dangerously close proximity, increasing the risk of collisions. For example, an excavator moving soil with its arm may hit a truck that approaches to load and transport such soil. This event poses serious safety hazards, potentially injuring the truck operator or causing death. To mitigate this risk, the orchestrator shall enforce a minimum safe distance between CS, preventing dangerous interactions and ensuring safer operations. A summary of the System of Interest (SoI) is provided in Table 1.

Table 1. System of Interest (SoI).

Aspect	Description
Mission	Mass removal in a construction site.
Action	Task allocation.
Loss	Workers dead or severe injuries.
Hazard	CS dangerous proximity.
Safety Req.	Maintain a safe distance between CS.

(b) **Core Identification.** As outlined in the previous section, the orchestrator's primary role is task allocation, a common function to all CS. To perform this task, the orchestrator must understand the overarching SoS mission, the surrounding environment, and the CS capabilities and availability. Since the SoS mission is broad, it is first broken down into smaller, manageable sub-missions that can be assigned to different CS. However, not all CS may interpret them directly, especially if some are fully automated. Therefore, the orchestrator must translate sub-missions into specific tasks, schedule them by considering the safe distances required to prevent collisions, and allocate them accordingly. This process requires that the orchestrator acquire specific information with the help of well-defined interfaces to receive mission, gather environmental data, and access CS information. A summary of the orchestrator core description is provided in Table 2.

Table 2. Orchestrator Core Description.

Aspect	Description
Functionality	Mission understanding.
	Mission breakdown.
	Safe scheduling.
	Task assignment.
Inputs	SoS Mission.
	Environmental data (world view).
	CS capability and availability.

(c) **Control Structure Modeling.** We take as inputs the information presented in Table 2 to model the control structure of the orchestrator. The resulting model includes four blocks with their corresponding interactions (see Fig. 2).

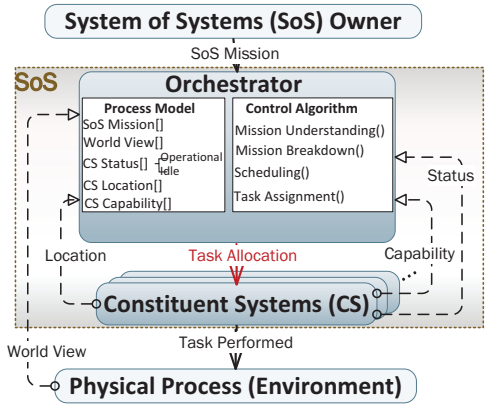

Fig. 2. Control Structure.

The outmost blocks are the *SoS Owner* (at the top), in charge of describing missions, and the *Physical Process* (at the bottom), affected by the

executed tasks. Those blocks provide inputs to the SoS, which contains the *orchestrator* and the *orchestrated CS*. The orchestrator takes the missions from the SoS owner, the world view from the environment, and the CS capability, location, and status. This information is maintained in the orchestrator's process model and processed by the control algorithm to produce the required control action, i.e., task allocation, which is critical. The functions previously identified, i.e., mission understanding, mission breakdown, safe scheduling (which shall enforce the safety requirement included in Table 1), and task assignment, are included in the algorithm of the control structure.

4.2. *Safety Analysis*

This section presents the steps required to derive the orchestrator safety concept (see Section 3.2).

(a) **UCA Identification.** The control action (CA) under analysis is task allocation, represented by the red arrow leaving the orchestrator in Fig. 2. To identify unsafe control actions (UCA), we apply the control action types (CAT) defined in the STPA process (see Section 2.3-S3). This structured approach helps uncover seven UCAs that may contribute to specific hazards, requiring further detailed examination. The results of this analysis are summarized in Table 3.

(b) **Causal Analysis.** All unsafe control actions (UCA) identified in Table 3 are analyzed based on the four possible loss scenarios outlined in Section 2.3-S4. To keep the discussion concise, we illustrate only the analysis for UCA-1 in Table 4. However, a similar analysis should be conducted for all identified UCAs.

Table 4. Causal Analysis for UCA-1.

Cause	Scenario
UCI-1	Unsafe parameters, e.g., wrong localization or capability are input into the orchestrator, leading to the allocation of dangerous tasks.
ICA-1	The algorithm does not consider the associated safety constraint when determining task allocations, leading to unsafe choices.
IPM-1	The model does not include needed parameters, e.g., machine limitations or task completion, leading to unsafe task allocation.
IF-1	Inadequate real-time feedback from the CS or incorrect status updates leading the orchestrator to make unsafe allocations.

(c) **Constraints Definition.** The causal analysis identifies 28 scenarios, which are categorized based on their root causes: UCI, ICA, IPM, and IF (see abbreviations description in Section 2.3-S4). After organizing them, root causes are reviewed to eliminate redundancies, and corresponding safety constraints are proposed (see Table 5).

Table 3. Unsafe Control Action (UCA).

Id	CAT	UCA	Hazard Description
UCA-1	Unsafe CA-provided	The orchestrator allocates tasks that are unsafe.	CS may operate too closely, leading to collisions between them.
UCA-2	CA-not provided	The orchestrator does not allocate tasks when required.	Gaps in task allocation can create overlapping zones or improper system placements.
UCA-3	CA-provided too early	The orchestrator allocate the tasks prematurely.	CS may begin tasks before the operational environment is ready, increasing the collision risk.
UCA-4	CA-provided too late	The orchestrator delays the allocation of tasks.	Workers may attempt to act manually to compensate for delays, increasing human errors.
UCA-5	CA-out of sequence	The orchestrator allocates tasks in the wrong sequence or to the wrong CS.	Wrong sequencing may cause multiple CS to operate in overlapping zones, increasing the likelihood of collisions.
UCA-6	CA-stopped too soon	The orchestrator prematurely stops the allocated tasks (e.g., provides an unexpected stop).	Tasks left unfinished (e.g., partially dug areas or unremoved debris) can create hazards in the operational environment.
UCA-7	CA-applied too long	The orchestrator allocates tasks repetitively to the same CS or assigns redundant tasks.	Repetitive execution of the same task can overstrain CS, causing equipment to fail unexpectedly, potentially leading to accidents.

Table 5. Constraints Definition for the Orchestrator.

Cause	Safety Constraints *The orchestrator shall:*
UCI	**SSR-1:** validate all input data by cross-checking against predefined criteria or supplementary sources.
	SSR-2: include a monitoring mechanism that tracks the timing of expected inputs.
	SSR-3: notify operators to prevent unsafe task generation if inputs are found to be incorrect or delayed.
ICA	**SSR-4:** include a monitoring mechanism to ensure the safety distance between CS.
	SSR-5: have the ability to stop a CS in case of safety distance violation.
	SSR-6: allocate tasks within a specified time frame to prevent delays and bottlenecks in the operation.
	SSR-7: ensure that all tasks allocated to CS are within their operational capabilities.
	SSR-8: ensure CS dependencies are satisfied before task allocation.
	SSR-9: incorporate mission completion criteria to ensure that tasks are allocated adequately.
	SSR-10: monitor its own operational integrity to detect faults that may lead to unsafe task allocation.
IPM	**SSR-11:** maintain an updated process model of each CS, i.e., location, capability, status, and constraints.
	SSR-12: tracks ongoing tasks, preventing their premature terminations.
	SSR-13: flag tasks that exceed expected timeframes without completion.
	SSR-14: update the process model to reflect real-time situations, i.e., changes in the environment.
IF	**SSR-15:** periodically collect feedback from CS and physical process about task execution outcomes.
	SSR-16: periodically require CS to report readiness states.
	SSR-17: detect system unresponsiveness or missing updates.

(d) **Variability Analysis.** Several sources of variability stem from the environment (e.g., different weather conditions), the CS (e.g., different versions of a truck), and their status (e.g., iddle or operational). This variability is expected during task allocation and can impact the established constraints (see Table 5). For this reason, it is essential to analyze every constraint in the context of each source of variability. For illustrations purposes, we focus on two key safety requirements, i.e., SSR-4, which ensures continuous monitoring of safe distances, and SSR-5, which ensures the existence of collision avoidance functions. Together, these requirements enforce the initially identified safety requirement of the System of Interest (SoI) (see Table 1). However, their effectiveness depends on the specific characteristics of each CS, such as speed, payload capacity, and maneuverability. Additionally, CS status is dynamic, not only in terms of availability but also in operational efficiency and condition (e.g., breakdowns, performance degradation). Furthermore, CS may operate at different levels of automation, ranging from fully autonomous to manual control. These variations require adapting and refining safety requirements to accommodate different CS capabilities and operational conditions. The necessary adjustments are outlined in Table 6.

Table 6. Constraints Refinement to include Variability.

ID	Refinement *The orchestrator shall:*
SSR-4	**SSR-4.1:** calculate safety distances dynamically based on CS capabilities, e.g., speed and the environmental factors, e.g., weather.
	SSR-4.2: adjust safety distances in real-time to account for changes in CS status (e.g., malfunctions, low power and automation mode).
	SSR-4.3: schedule tasks to avoid overlapping CS operations within unsafe proximity.
SSR-5	**SSR-5.1:** issue an immediate stop command when there are unsafe distances between CS.
	SSR-5.2: reallocate CS with degraded status, minimizing their interaction with other CS.
	SSR-5.3: define a priority hierarchy during emergencies, where safety-critical tasks take precedence over other tasks.

4.3. *Configuration Verification*

As presented in Section 3.3(a), specific variants are selected from the planned variability to form a possible configuration. At this stage, we can discover that some variants have not been considered due to access to new information. To solve this, we could either test it to see how much it fits with the configurations or return to the variability analysis. With the configuration in place, as presented in Section 3.3(b), the orchestrator is modeled, for example, as a state machine that con-

siders a set of variables, such as the mission completion, CS status, and CS availability. Constraints like safety distance and the subsequent policies, such as emergency stops, can also be included. This model is created using tools that allow the representation of state transitions to, as presented in Section 3.3(c)), enable the automated verification of specific safety metrics. For example, we could consider two key safety properties. First, the safety distance monitoring shall ensure that no violations of safety distances occur between CS during operations. Second, correctly using the collision avoidance signal to verify that CS stops in case of safety violations. Additionally, predefined safety thresholds can be incorporated, such as maintaining an error margin of ±0.5 meters in the safety distance calculation algorithm. The verification process may reveal counterexamples indicating where safety properties are violated. These counterexamples are analyzed further to refine the control structure and the safety concept by adding necessary inputs and adjusting specific parameters. This process continues until all safety properties are fully satisfied.

5. Discussion

Orchestrating SoS enables the management of the CS's interactions in an effective manner. However, ensuring safety in SoS orchestrations can be challenging due to the natural variability included in an SoS (as presented in Section 3). In particular, CS differ in several aspects, e.g., modes of operations, and automation statuses. The environments in which a SoS operates also vary significantly during operational time. The operational context, for example, evolves as tasks progress, such as new areas being excavated, or structures being erected. These dynamics require the SoS to adapt, posing significant challenges for maintaining consistent safety in the orchestration.

We have identified several key points based on the application scenario (see Section 4). First, the systematic nature of SPLE and STPA ensures consistency in identifying, analyzing, and addressing risks, making the process repeatable and less prone to oversight. Second, an inherent alignment between top-down (STPA) and bottom-up (SPLE) perspectives ensures comprehensive coverage of risks at both granular and systemic levels. For example, in Table 3, we identified hazards beyond the one initially considered, i.e., potential human errors (UCA-4), environmental hazard (UCA-5) and equipment failure (UCA-6). Third, the resulting safety constraints could change the orchestrator's initial view drastically. For example, from the lists presented in Tables 5 and 6, we can see that various new variables and functions must be included in the control logic. If we do an analysis of applicable standards, such as IEC 61508 (IEC, 2010), we identify that the monitoring of safety distances and stop functions (SSR-4 and SSR-5) are very critical and require independence from the SoI. Thus, it is vital to clearly delineate the functional scope of the SoS orchestrator (i.e., as a central services hub) to avoid unnecessary overload. This means that several functions required for safety may need to be allocated outside of the orchestrator, e.g., additional controllers to ensure redundancy or directly on the CS. This trade-off is crucial for the safety analysis to remain relevant and manageable within the orchestrator context.

6. Related Work

Safety in SoS has remained a research gap due to the SoS's inherent nature, e.g., complexity, autonomy, and heterogeneity (Despotou et al., 2009). However, the state of the art has proposed new perspectives to minimize this gap. In particular, the authors in (Baumgart et al., 2021) introduced a model-driven approach to evaluate SoS safety, including failure mode and effects analysis (FMEA) and hazard and operability (HAZOP) analyses. Similarly, in (Muram et al., 2019), the authors contributed to SoS hazard identification by combining HAZOP results with the Fault Tree Analysis (FTA). The Systems perspective has also been introduced in (Axelsson and Kobetski, 2018), where STAMP has been extended for SoS to address risks beyond safety. Derived from STAMP, STPA has also been broadly used for safety analysis in SoS, e.g., (Baumgart et al., 2019; Castellanos-Ardila et al., 2022; Buysse et al., 2022). Ontological approaches are also provided, e.g., (Adach et al., 2023; Ali et al., 2024) to

facilitate safety and security reasoning in SoS. As in the previous work, we are proposing a safety analysis for SoS, which is also based on STPA. However, we extend such analysis in a novel fashion by considering the inherent SoS variability, which in our work is analyzed via the use of SPLE.

7. Conclusions and Future Work

SOSoS is aimed at helping practitioners orchestrate SoS operations safely by leveraging the principles of SPLE and STPA. As shown in our application scenario, SOSoS systematically guides the identification of safety risks relevant to SoS orchestrations that could otherwise be overlooked.

Future work will consider security analysis to account for threats affecting SoS safety. We also will provide more case studies to further assess SOSoS in practice. Finally, we will present the process and the resulting orchestrations to practitioners and safety assessors to evaluate their perceptions regarding the SoS composition.

Acknowledgement

This research is supported by the Vinnova funded SIMCON project and the SSF funded DAISY project.

References

Adach, M., N. Ali, K. Hänninen, and K. Lundqvist (2023). Hazard analysis on a system of systems using the hazard ontology. In *18th Annual System of Systems Engineering Conference*, pp. 1–6.

Ali, N., J. P. Castellanos-Ardila, and S. Punnekkat (2024). Towards an integrated safety-security ontology for system of systems. In *International Symposium on Systems Engineering (ISSE)*, pp. 1–8. IEEE.

Axelsson, J. and A. Kobetski (2018). Towards a risk analysis method for systems-of-systems based on systems thinking. In *Annual IEEE International Systems Conference (SysCon)*.

Axelsson, J. and P. Svenson (2022). On the concepts of capability and constituent system independence in sos. In *17th Annual System of Systems Engineering Conference (SOSE)*, pp. 247–252. IEEE.

Baumgart, S., J. Fröberg, and S. Punnekkat (2019). A state-based extension to stpa for safety-critical system-of-systems. In *4th International Conference on System Reliability and Safety*, pp. 246–254.

Baumgart, S., J. Fröberg, and S. Punnekkat (2021). How to analyze the safety of concepts for a system-of-systems? In *IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–8.

Beland, S. C. and A. Miller (2007). Assuring a complex safety-critical systems of systems. *SAE Transactions*, 974–988.

Botterweck, G. (2013). Variability and evolution in systems of systems. *arXiv preprint arXiv:1311.3627*.

Buysse, L., D. Vanoost, J. Vankeirsbilck, J. Boydens, and D. Pissoort (2022). Case study analysis of stpa as basis for dynamic safety assurance of autonomous systems. In *European Dependable Computing Conference*, pp. 37–45. Springer.

Castellanos-Ardila, J. P., H. Hansson, and S. Punekkat (2022). Safe integration of autonomous machines in underground mining environments. In *International Symposium on Systems Engineering (ISSE)*, pp. 1–8.

Dahmann, J. S. and K. J. Baldwin (2008). Understanding the current state of us defense systems of systems and the implications for systems engineering. In *2nd Annual IEEE Systems Conference*.

Despotou, G., R. Alexander, and T. Kelly (2009). Addressing challenges of hazard analysis in systems of systems. In *3rd Annual IEEE Systems Conference*.

Fisher, D. A. (2006). An emergent perspective on interoperation in systems of systems. *Software Engineering Institute, Carnegie Mellon), March*.

IEC (2010). *IEC 61508 - Functional Safety of Electrical/Electronic/Programmable Electronic (E/E/PE) safety-related systems*.

ISO/IEC JTC 1/SC 7 (2019). *ISO/IEC/IEEE 21841:2019. Systems and Software Engineering — Taxonomy of System of Systems*.

Josuttis, N. M. (2007). *SOA in practice: the art of distributed system design*. O'Reilly Media, Inc.

Klein, J. and J. D. McGregor (2013). System-of-systems platform scoping. In *4th International Workshop on Product LinE Approaches in Software Engineering(PLEASE)*.

Leveson, N. G. (2016). *Engineering a safer world: Systems thinking applied to safety*. The MIT Press.

Leveson, N. G. and J. P. Thomas (2018). *STPA Handbook*.

Maier, M. W. (1998). Architecting Principles for Systems-of-Systems. *The Journal of the International Council on Systems Engineering 1*(4).

Muram, F. U., M. A. Javed, and S. Punnekkat (2019). System of systems hazard analysis using hazop and fta for advanced quarry production. In *4th International Conference on System Reliability and Safety (ICSRS)*, pp. 394–401. IEEE.

Nordstrom, T., L. Sutfeld, and T. Besker (2024). Exploring different actor roles in orchestrations of system of systems. In *19th Annual System of Systems Engineering Conference (SoSE)*.

Pohl, K., G. Böckle, and F. van der Linden (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, Berlin.