STPD - Operational Design as a Model-Based Extension of STPA

Avi Harel, Ergolight

Keywords: STAMP, STPA, STPD, Scenario-Centered Operation, Operational Design, Operational network, Canonical Scenario Representation, Controller Multi-Service Interaction, Exceptions, Decision Support, Feedback, Feedforward, Proactive Troubleshooting, Model-Based Systems Engineering, System State Consistency, Human Factors, HSI, Socio-Technical Systems, Digital Twin, LLM

Abstract

The paper argues that the growing complexity of distributed socio-technical systems exposes a fundamental limitation of the STPA methodology. Although STPA transformed safety engineering (Leveson 2011; Leveson 2004) by treating safety as a control and constraint problem rather than purely a reliability problem, it still assumes that hazardous scenarios and the constraints needed to prevent them can largely be identified during design time. The paper claims that this assumption no longer holds in modern adaptive systems (Rasmussen 1997; Woods 2015), where operational contexts evolve continuously, AI-based components change behavior dynamically, distributed actors develop inconsistent interpretations of the operational situation, and unexpected interactions emerge during runtime.

According to the paper, safety in such environments cannot rely only on predefined constraints. Instead, systems must continuously validate during runtime whether operational assumptions, coordination structures, and safety constraints still remain valid under changing conditions. This shifts the engineering challenge from specifying static safe behavior toward maintaining operational coherence under uncertainty. To address this challenge, the paper proposes extending STPA into a broader framework called system-theoretic process design, (STPD). STPD treats runtime surprise (Woods 2015; Dekker 2011) as an inherent property of complex adaptive systems (Rasmussen1997; Woods 2015) rather than as an exceptional case outside the model. Under this approach, systems must be capable of detecting deviations from expected operational assumptions, adapting coordination structures, managing degraded modes, recovering from anomalies, and learning from unforeseen situations while still constraining behavior within acceptable operational limits.

The paper emphasizes that this extension substantially increases analytical complexity. The analysis must address not only fixed control loops and predefined unsafe control actions, but also adaptive behaviors, runtime transitions, resilience mechanisms (Hollnagel, Woods, and Leveson 2006; Woods 2015), evolving operational contexts, and partially unknown future states. Resilience itself introduces additional coupling, variability, coordination overhead, and verification difficulty. Consequently, a central tradeoff emerges: systems must remain adaptive

enough to survive operational surprise while simultaneously remaining constrained enough to preserve safety, controllability, productivity, and usability.

A central criticism presented in the paper concerns the scalability limitations of classical STPA. The number of possible socio-technical operational states grows exponentially with the number of system variables, making exhaustive analysis infeasible. The paper also argues that STPA's constraint-based logic requires engineers to identify all hazardous control actions explicitly, whereas the original STAMP philosophy focused more broadly on constraining the system toward correct operation according to an intended operational plan. Based on this critique, the paper proposes combining STPA with the original STAMP concept of operational guidance. The proposed solution proactively defines operational scenarios during design time and enforces corresponding system states during runtime transitions. The system continuously verifies whether ongoing operations remain aligned with the intended coordination structure, detects deviations, and activates emergency or recovery mechanisms when inconsistencies appear. To implement this idea, the paper introduces a generic operational model based on a network of interactions. Each interaction is defined through protocols between a controller and multiple services. Operational constraints are represented as scenarios consisting of an external task and the coordinated internal states of participating components. Participants may initiate additional interactions, thereby forming a dynamic interaction network. The model includes both proactive coordination mechanisms and reactive anomaly detection and response mechanisms.

Finally, the paper argues that this modular interaction-based structure improves the practical application of STPA by associating control states with specific interaction participants. According to the paper, this association increases traceability and helps achieve broader analytical coverage across all stages of the STPA process.

Introduction and Core Metamorphosis

The paper introduces a transition from the STPA approach to the system-theoretic process design (STPD) approach, where the hierarchy of constraints relies on a hierarchy of components within the system's operational model. In this approach, runtime surprises (Woods 2015; Dekker, 2011) are not perceived as external anomalies but rather as an inevitable part of complex systems. Within this modeling framework, systems are required to detect deviations from operational assumptions, adapt coordination and control mechanisms, manage degraded operational modes, and learn from unexpected events—all while maintaining safe constraint boundaries.

The article addresses the core dilemma of operational flexibility, which is necessary to cope with surprises, versus protection against operational threats and loss of control in anomalous situations. The extension to STPD significantly increases analysis complexity. Instead of dealing solely with predefined hazardous control actions, it must also contend with changing contexts, runtime state transitions, adaptive behavior, and resilience mechanisms (Hollnagel, Woods, and Leveson 2006; Woods 2015). Furthermore, the very addition of resilience manifests as complexity that induces coordination overhead, behavioral variance, and tighter coupling between components.

Additionally, the paper analyzes the theoretical boundaries of STPA in preventing unexpected failures (Safety I). These limits stem from the exponential growth in the number of possible system states and the practical difficulty of enumerating all unsafe control actions (UCAs). This stands in contrast to the original STAMP approach, which suggested prioritizing proactive constraint of the system toward proper operation according to an operational plan (Safety II (Hollnagel 2014)). The paper proposes to bridge STAMP and STPA through the proactive design of scenario-compatible system states and enforcing these states at runtime.

STPD extends STPA by integrating explicit mechanisms for runtime scenario coordination, rather than relying solely on the static design of the control structure. Within this framework, a canonical scenario representation (CSR) is defined, enabling a uniform, formal, and consistent description of operational modes and the transitions between them. Moreover, STPD focuses on enforcing scenario consistency across distributed controllers (Leveson 2011; Hutchins 1995) and components, ensuring that the entire system operates according to a shared operational understanding, even under conditions of uncertainty and dynamic changes.

The methodology also includes state transition control mechanisms that are aware of the scenario's coordination context, as well as capabilities for runtime verification of operational coherence among distributed controllers (Leveson 2011; Hutchins 1995). This is intended to mitigate deviations, miscoordination, and degradation into hazardous states. Additionally, the system is continuously required to verify that operations align with coordination definitions, detect deviations, and activate emergency, recovery, and troubleshooting mechanisms.

To implement this approach, a generic system operation model is proposed via a directed network representation, where nodes represent interactions between controllers and services, and the connecting lines represent scenarios. An interaction is defined through dedicated protocols, and scenarios are generated by combining external tasks received from the higher echelon, system states, and context variables. The operational model allows for both automatic discovery and reactive response to deviations, as well as the proactive initiation of diagnostic and recovery processes. Each system component can trigger additional interactions, thereby forming a dynamic network of controlled interactions. This generic model allows organizing the STPA control structure according to the models, thereby ensuring that the STPA safety constraints analysis addresses all conditions in normal operation.

Operational Background and System Challenges

The paper focuses on the central challenge of operational design in complex socio-technical systems—namely, how to guarantee both performance/throughput and safety/coordination among system components at runtime. The starting point is the shift from the 'blame the operator' paradigm to a systemic approach that views failure as a result of design flaws and structural mismatches between system components.

A prime historical example is the 2001 Kandahar incident (Mode confusion - Kandahar 2001 FFA), where a battery replacement caused a GPS reset. Consequently, its operational mode defaulted back to navigation mode, and the waypoint shifted from the Taliban target's location to the coordinates of the GPS unit itself, as illustrated in the following figure.

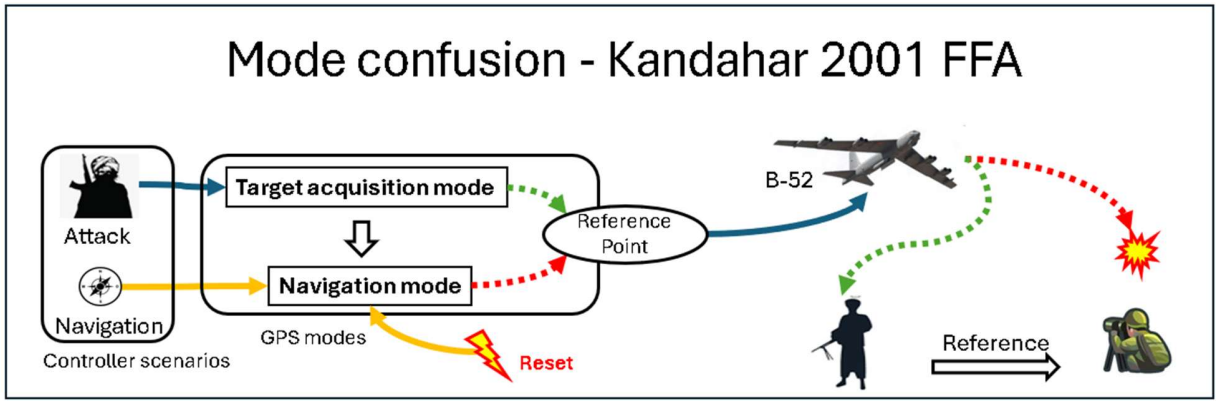

Figure 1. Mode confusion in the Kandahar accident

The failure stemmed from the design allowing a state where the GPS operational mode was mismatched with the active operational scenario. In such cases, the risk did not arise from a classic hardware or software malfunction, but rather from an inconsistent interpretation of the scenario across different control layers. The challenge lies in the fact that multiple independent units must collectively produce consistent behavior, even though their information regarding the state and intentions of neighboring units is partial and occasionally outdated. More examples are listed in the Appendix.

In alignment with the STAMP approach, safety is not achieved merely by reacting to failure triggers, but by constraining system operations to remain within proper, controlled boundaries. In this context, the STPA method is introduced, allowing the identification of unsafe control actions and the derivation of safety constraints early in the design phase. STPA provides support for real-time coordination, synchronization, and constraint enforcement in distributed, dynamic systems. The problem becomes particularly acute when different subsystems hold differing internal models of the system state—meaning an incompatibility exists between the 'process models (Leveson 2011)' of various controllers and the actual reality. This condition can lead to operational errors, coordination breakdown, and systemic failure even when each individual component operates 'correctly' from its local perspective. Although STAMP/STPA address process models, feedback, and loss of control, the engineering realization of runtime coordination among distributed controllers (Leveson 2011; Hutchins 1995) remains an open challenge.

The paper emphasizes that the complexity of STPA scales rapidly with system complexity. As the number of control actions, operational modes, and interactions increases, an immense number of potential unsafe control actions (UCAs) is generated. This situation complicates

generalization, coverage, and prioritization, often making the analysis overly expensive and cumbersome. To address this, a model-driven STPA approach is introduced, shifting the analysis focus from individual control actions to operational states and deviations from operational assumptions. Instead of manually analyzing every possible action, the system examines which states deviate from the normal operational model, thereby improving generalization capabilities and reducing the risk of missing hazardous conditions.

Furthermore, the paper proposes a 'lean' version of STPA for systems where risk is not catastrophic but stems primarily from interaction and coordination issues—such as software systems, smart home devices, or office services. This streamlined approach focuses primarily on state transitions, synchronization logic, and user-service interactions to rapidly identify coordination gaps, timing errors, and conflicting state assumptions.

The paper also draws an important distinction between prevention strategies and protection strategies. Prevention aims to avoid the creation of hazardous states altogether through safety constraints, state control, input validation, and resilient design. Conversely, protection assumes that coordination failures and unforeseen conditions will inevitably occur, and thus includes mechanisms for detection, alerting, transitioning to a safe state, controlled degradation, and recovery. Resilient systems must combine both approaches: minimizing failure probability while effectively handling coordination loss when it occurs in practice.

STPD as an Extension of STPA

This section presents STPD as a systemic extension of STAMP/STPA for distributed socio-technical systems, where safety and efficiency depend not just on component integrity, but primarily on maintaining consistent coordination among the state interpretations of controllers, services, and human operators. The core argument is that many systemic risks do not stem from a single component failure, but rather from a deviation from the operational coherence envelope—manifesting as a state where different components act under mismatched understandings of the operational scenario. In complex systems, partial information, delayed feedback, poor synchronization, or normal performance variance can cause the system to continue operating 'correctly' at a local level while its global behavior becomes highly dangerous. Therefore, the central challenge is not just preventing known failures but identifying emerging loss of control and the collapse of the assumptions supporting the original control scheme at runtime. The contribution of STPD is not in replacing STPA's control mechanisms, but in adding an operational coordination layer that manages scenario consistency, transition synchronization, interpretation alignment, and runtime constraint enforcement.

The text emphasizes that resilient systems must be capable of detecting even unpredicted situations, such as errors in specification, design, and component implementation. This requires an explicit representation of normal operations, including operational plans, constraints, process models (Leveson 2011), coordination rules, and assumptions regarding resources and expected behavior. The system cannot merely rely on detecting component faults; it must monitor the

integrity of the operational coherence itself, identifying deviations from expected task structures, coordination issues among distributed process models, breaches of coordination protocols, or gaps between expected and actual behavior. The proposed approach views anomalous situations as semantic rather than merely physical phenomena, since the earliest indication of a problem may be a contradictory interpretation or a loss of shared situational awareness. Therefore, the system must integrate structural monitoring, semantic consistency checks, uncertainty estimation, and the capability to transition to safe backup states when the active scenario can no longer be determined with confidence.

A central concept introduced is the unsafe operational state (UOS), where controllers or services no longer agree on the mission state or operational mode. In such a state, even commands normally deemed safe can become hazardous because inter-unit coordination is lost. From this arises the assertion that safety in distributed systems depends on consistency across the controllers' process models (Leveson 2011). The chapter cites several classic examples, such as the Therac-25 accidents, Three Mile Island (TMI), and friendly-fire incidents, to illustrate how critical failures are generated by a gap between the internal representation of the system state and physical reality. In these cases, it was not necessary to isolate a specific software, hardware, or operator fault; instead, the failure cause is attributed to different subsystems operating under conflicting assumptions regarding the operational state. Accordingly, the paper interprets many accidents as violations of systemic consistency constraints rather than isolated local failures.

Based on this, the chapter proposes a transition from an exhaustive failure-cause analysis model to a scenario-constrained situational control (SCSC) model. Instead of trying to analyze all possible system states—an intractable approach in complex systems due to 'state explosion'—systems engineers are required to define a limited number of operational scenario classes specified in operational documentation. Each scenario represents a group of states with shared operational meaning and constraints, along with permitted transitions and backup rules. At runtime, the active scenario is represented via the CSR, which serves as a shared reference base for interactions between the controller and services. This significantly reduces coordination complexity because the system is not required to analyze risk across all theoretical states, but only to operate within a well-defined scenario space. When the system encounters an uncertain situation that cannot be confidently mapped to one of the scenarios, it must transition to a safe mode scenario where only conservative, safe actions are permitted.

The chapter also defines 'scenario confusion', a condition where different components within the system operate according to different scenarios. Even though each component acts 'correctly' locally, the discrepancy between them creates a dangerous systemic failure.

Consider a system with two elements: a controller and a service, enabling them to perform two functions, depending on the service mode. The service provided should comply with the controller scenario. This defines normal situation. The operation is successful if the service complies with the operational scenario and is failure otherwise. Figure 2 illustrates this condition, in which the controller and service operate under inconsistent scenarios.

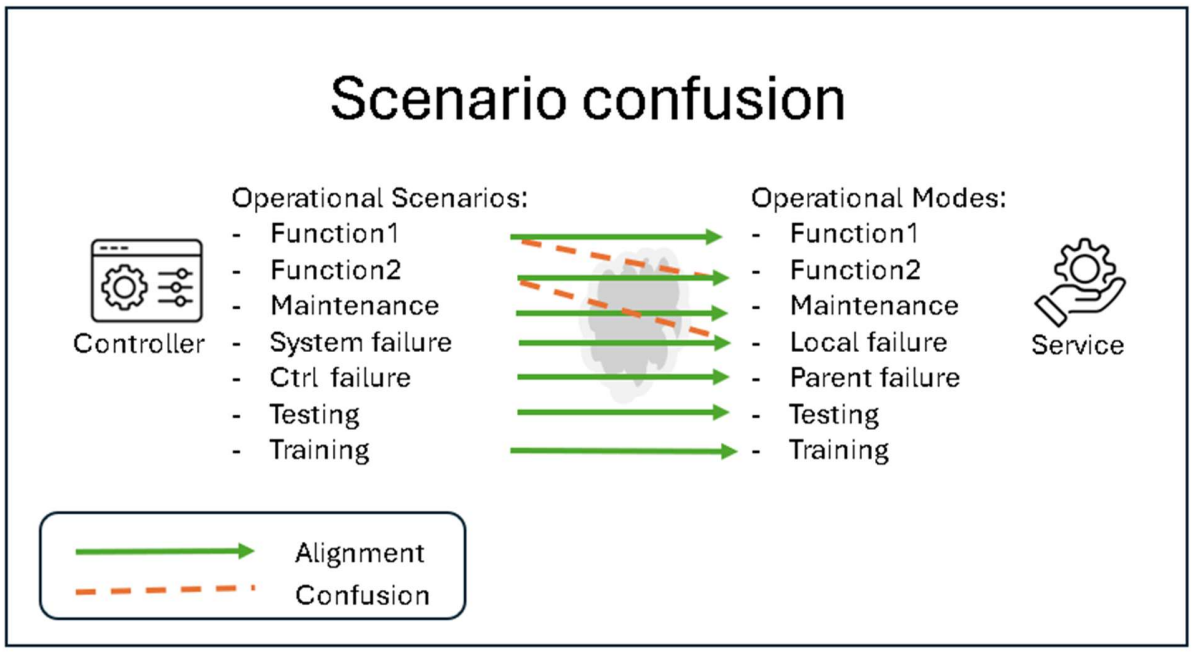

Figure 2. Scenario confusion

The figure illustrates a way to enforce scenario-mode compliance (green arrows) and examples of scenario confusion, when the modes divert from those induced by the scenarios (broken red lines).

The model shows enforcement mechanisms for scenario-mode alignment alongside examples of scenario confusion where operational modes deviate from the scenarios that should dictate them. In many historical instances, the scenario was never explicitly defined, leaving the control dilemma unresolved. In *Human Factors Engineering* (Reason 1990; Dekker 2014) (HFE), this is termed a 'mode error (Sarter and Woods 1995; Norman 1981)'. The underlying, flawed assumption was that the operator must ensure the operational mode matches the scenario. In reality, this assumption proved incorrect, and the system was incapable of self-protection. From a systems engineering perspective, this constitutes a coordination failure between the controller and the service. The argument is that most famous 'mode errors' in HFE are actually systemic coordination failures resulting from the absence of an explicit representation of the active scenario. Hence, the scenario must become an explicit system variable whose consistency can be verified at runtime. Figure 3 illustrates how scenarios are defined and how they impact the design.

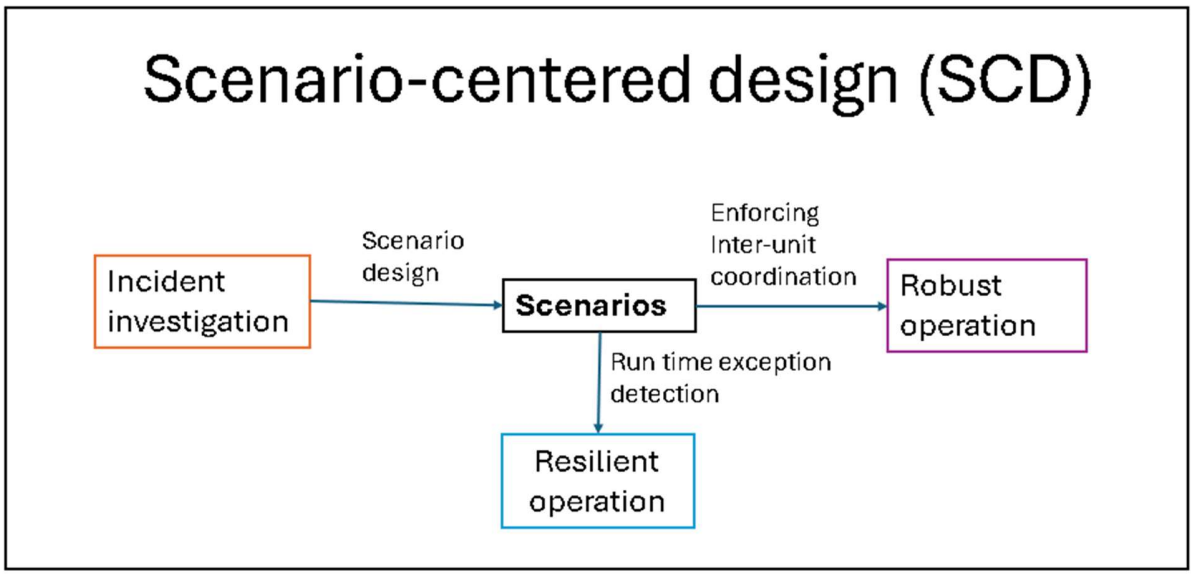

Figure 3. Scenario-centered design

The scenario serves as a shared reference point: it represents the mission as defined by the system owners and is used both for enforcing coordination and for identifying unexpected behavior when scenario consistency is compromised. Different components may hold a partial or delayed view of the state, but their behavior is constrained by the scenario available to them at execution time. Thus, coordination is achieved through consistency of interpretation rather than full system-state synchronization. Within this framework, the scenario functions as a control variable. Maintaining consistency of interpretation becomes an explicit control objective, and deviations from scenario boundaries become detectable states. In this manner, scenarios bridge system design and actual execution, supporting coordination, consistency, and cognitive load management in dynamic environments. The proposed architecture enforces scenario consistency as a system invariant, such that control actions are permitted only when valid within the active scenario and coordinated across all components. When a mismatch is discovered, normal operations are suspended until synchronization is restored.

Furthermore, the challenges of transitions between scenarios are addressed. Transitions are considered a critical vulnerability point because subsystems may temporarily operate under differing states or assumptions during the switch. The Therac-25 accidents are presented as an example where rapid state transitions without synchronization created a time window in which the software state, hardware state, and operator intent were severely misaligned. Therefore, safety depends not just on stability within a state, but on the ability to rapidly regain coordinated control during a transition. The paper underscores the need for risk mitigation during synchronization, utilizing monitoring mechanisms for risk indicators such as contradictory states, timeouts, or degrading sensor trust, alongside anomaly response mechanisms that encompass alerts, troubleshooting, resilience (Hollnagel, Woods, and Leveson 2006; Woods 2015), and recovery. Diagnostics and troubleshooting are viewed here as proactive systemic activities

focused on identifying coordination degradation before the system manifests hazardous behavior. Ultimately, while STPA focuses on identifying unsafe control actions within control hierarchies, STPD adds a new layer of distributed operational coordination and runtime coherence management, moving the needle from static control constraints to engineering operational resilience under uncertainty and surprise.

Operational Modeling Framework

This section details the framework for modeling operations in distributed socio-technical systems based on scenario coordination rather than exhaustive state analysis. The framework establishes a conceptual hierarchy separating the physical system state, local controller process models (Leveson 2011), operational states, scenarios, and coordinated runtime control. At the core of the architecture lies the canonical scenario representation (CSR), serving as a shared reference point for all controllers and services. The CSR does not describe the complete system state but only the information necessary for coordination, synchronization, and enforcing operational constraints.

The operational plan is defined as a directed network of tasks, constraints, and dependencies that restricts the permitted behavior of the system. The system state represents the objective, full state, whereas the controllers' process models (Leveson 2011) are based on partial info and may thus be inconsistent. An operational state is a mission-relevant interpretation of the system state, and a scenario is a bounded operational context defining which actions and coordinations are allowed. The CSR acts as a systemic coordination anchor enabling controllers and services to maintain a shared interpretation of the operational state.

The operational model describes system activity as a network of interactions (controller multi-service interaction - CMSI) in each of which a single controller operates in partnership with several services, based on a shared CSR. Coordination is performed by propagating the CSR among components, synchronizing scenario transitions, and enforcing coordination constraints at runtime.

CSR control specifies how scenarios are established, updated, and propagated in the CMSI network, governing the way contextual information is maintained and shared. Its purpose is to ensure that transitions between scenarios remain coherent, that dependent interactions are provided with updated context, and that coordination is preserved across interaction boundaries. According to Boy and Narkevicius (2014) operational design should proceed from outside in. Figure 4 describes a model of operational networks, illustrating how an external actor triggers a sequence of interactions through passing canonical scenario representations.

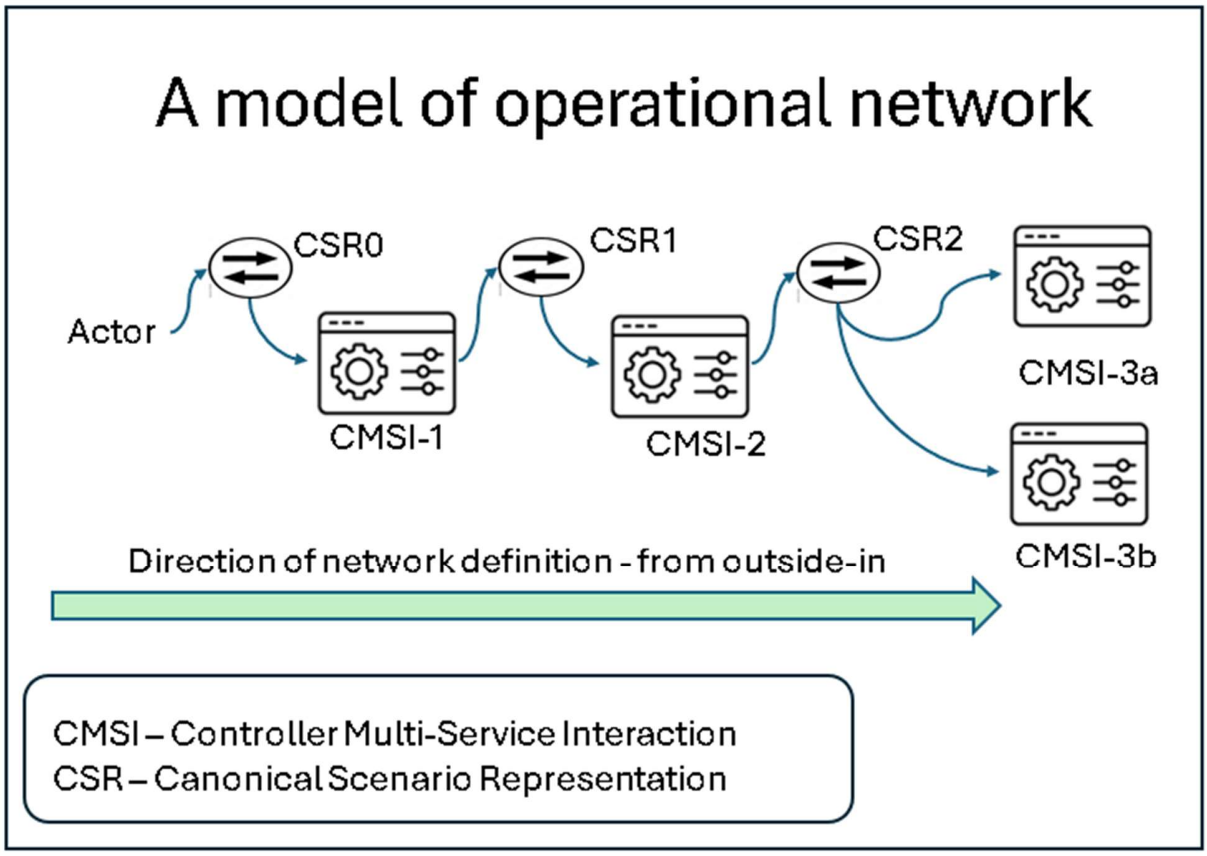

Figure 4. A model of operational network

This figure illustrates a simple network with four controller multi-service interactions, in which an external actor activates CMSI-1 by a canonical scenario representation CSR0, which activates CMSI-2 by CSR1, which activates CMSI-3a and 3b by CSR2. The CSR operates as a control layer over the CMSI network. Scenario control enables the CSR to function as a system-level reference rather than a local variable.

System operation may be modeled as a network of interactions, in form of a directed graph composed of CMSIs, where nodes correspond to interaction contexts implemented as CSR, and edges represent activation and dependency relationships between them. This structure provides a way to describe how coordination flows and propagates throughout the system. Figure 5 illustrates that scenarios should enable coordinating the supplier with the client task (on the left) and should enable coordinating the scenarios with the controller (on the right).

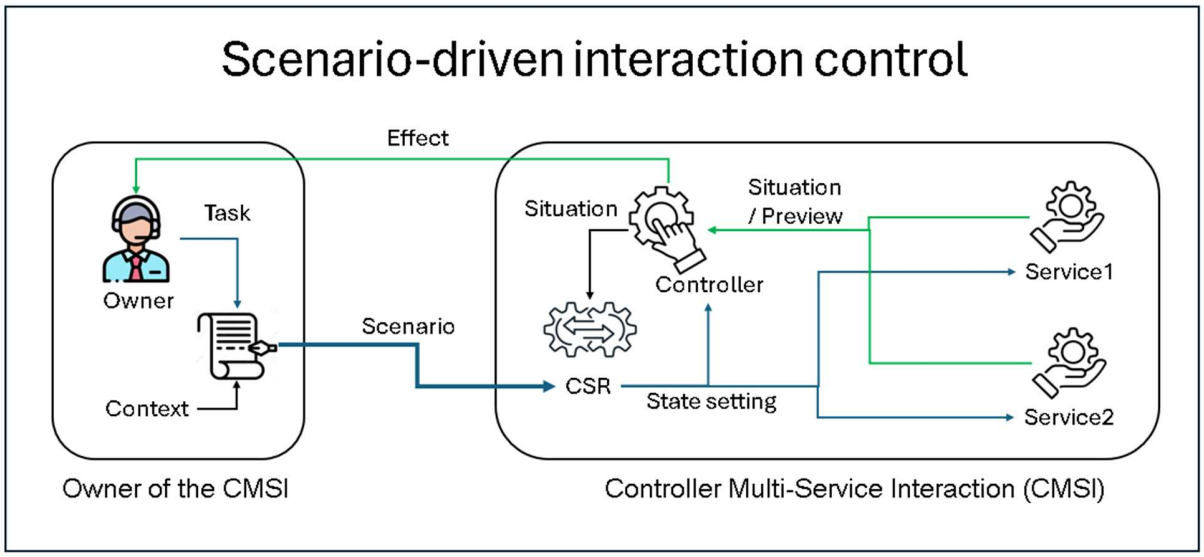

Figure 5. Scenario-driven operation

This figure shows a way for transition from scenarios to CSR, and the way it controls the interactions. The owner of the interaction defines the scenario based on the task and the context and delivers it to the controller. Based on the active scenario of the interaction (active, available, functional ...) the controller defines the CSR, a system variable used as an anchor for the coordination and sets the states of the services according to assignment rules.

Scenario control enables the CSR to function as a system-level reference point rather than just a local variable. The article highlights that safety failures in distributed systems frequently originate from inconsistent operational interpretations (scenario confusion), leading locally correct actions to produce dangerous global behavior. Consequently, the safety objective is defined as the continuous preservation of scenario consistency and inter-component coordination.

The CSR is also described as a structural invariant. Instead of relying on reactive arbitration between components, transitioning to a specific scenario pre-allocates permitted and validated states to all participating services. This eliminates dangerous intermediate states and establishes a 'structural normalcy' where many hazards are logically removed from the possible state space. The operational framework is driven by scenario-based interactions where the owner defines the scenario based on the task and context, passing it to the controller, which sets the CSR and assigns service modes accordingly.

The model focuses particularly on CSR transitions and synchronization. During a scenario transition, the system is expected to temporarily enter a safe mode, propagate transition commands to all components, await synchronization confirmations, and correct timing anomalies. Normal operation during transitions is prohibited because asynchronous state transitions can generate unpredictable behavior.

The paper introduces an extension of traditional STPA control loops through the CMSI structure. Instead of a single controller acting against a single process, the CMSI coordinates multiple services under a shared scenario. Additionally, the model introduces a feedforward layer where the controller evaluates future operational alternatives using previews, workload assessments, and service simulations. This allows for the early identification of actions that could create inconsistencies or violate scenario constraints.

Figure 6 illustrates how a CMSI module can provide feed-forward information for optimizing the controller decision.

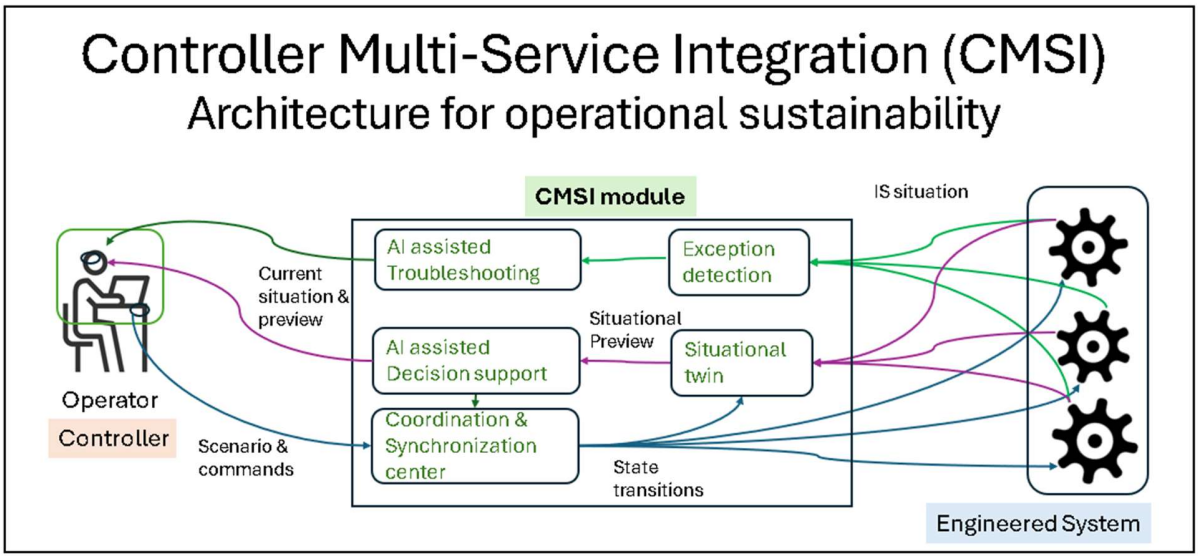

Figure 6. CMSI operation.

The figure illustrates a hypothetical CMSI module that may be used to coordinate and synchronize the controller and the services. Employing agents (implemented by situational twins) this module can provide the controller with information required for proactive troubleshooting and reacting. The framework does not prescribe specific prediction methods. Its effectiveness depends on model accuracy and computational constraints.

A key element involves handling unpredicted exceptions. The core claim is that it is impossible to map all potential UCAs in complex systems in advance. Therefore, instead of focusing on preventing each hazardous action in isolation, the system must ensure it remains within a normal, coordinated operational envelope. Exceptions are identified by comparing actual behavior against the behavior expected under the active scenario. Deviations may appear as synchronization faults, illegal task transitions, resource anomalies, timing inconsistencies, or loss of coherence between subsystems.

Many failures occur because the system detects anomalous conditions too late, provides insufficient recovery support, or continues operating after coordination has begun to degrade. A

generic model of exception management outlines how an exception evolves through sequential operational states (Normal operation -> Exceptional situation -> Unexpected operation -> Failure), identifying the critical points where coordinated handling is required to prevent escalation.

Even with structural enforcement mechanisms, not all inconsistencies can be prevented. Systems operating under uncertainty, incomplete information, or degraded conditions may still deviate from expected behavior.

Exception management concerns the way deviations from expected behavior are handled. Within this context, two complementary modes of operation can be distinguished. The first is situation control, which is continuous in nature. Here, operation progresses through ongoing regulation relative to the defined scenario, with adjustments made as needed to maintain consistency. The second is exception management in the narrower sense, which is event-based. In this mode, attention is focused on deviations that exceed predefined thresholds and therefore require explicit corrective action.

Many failures occur because the system detects abnormal conditions too late, provides insufficient recovery support, or continues operating after coordination has already degraded. Figure 6 illustrates a conceptual model of exception management.

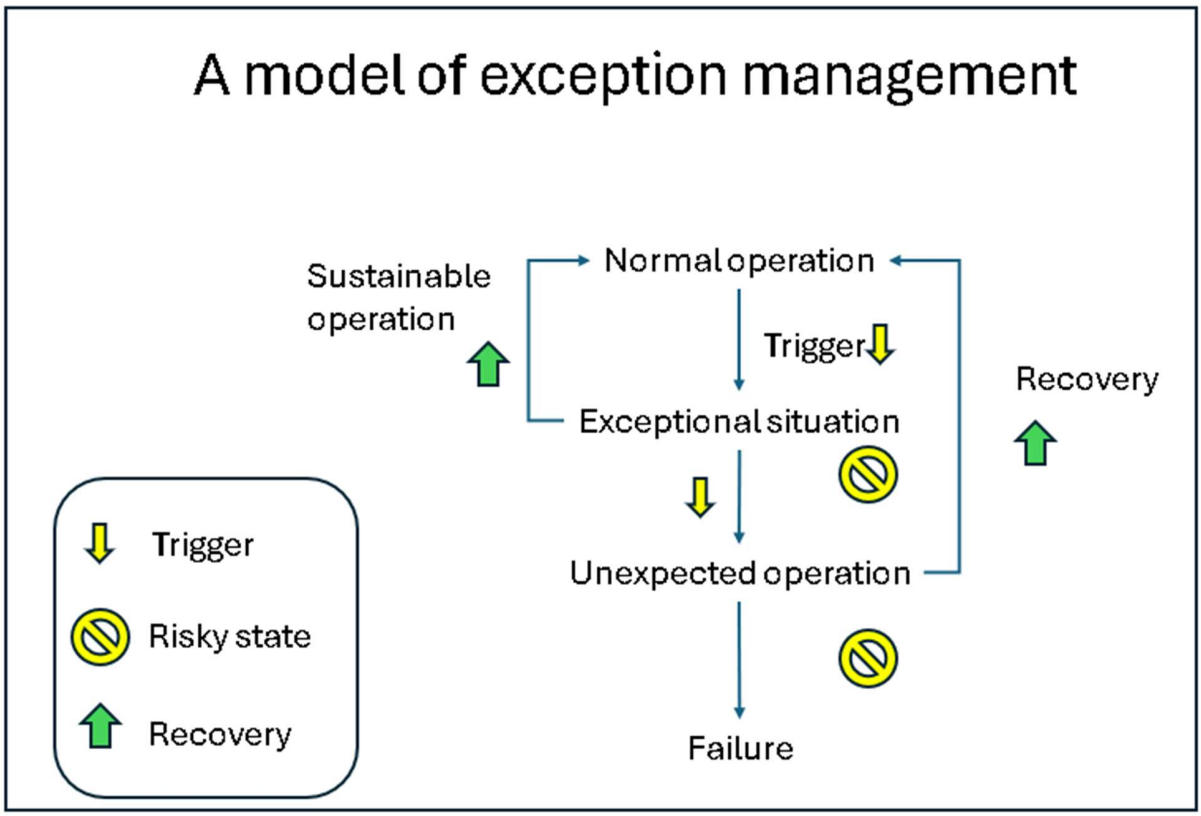

Figure 7. Exception handling

Figure 7 highlights how an exception evolves through successive operational conditions and identifies the points at which coordinated handling is required to prevent escalation. Following an initiating event, the system may transition from normal operation to an exceptional state. If the exception is not properly handled, the system may further degrade into unexpected operation and ultimately failure.

The model of normal operation allows for the detection of even unforeseen exceptions by defining what constraints must be maintained; any violation is interpreted as a deviation, shifting the safety focus to runtime validation of operational consistency.

Operational Engineering and Design Methodology

The engineering framework focuses on enforcing operational constraints through scenarios, distributed coordination, and transition control. Instead of looking at isolated components and isolated failures, the framework centers on preserving total operational consistency and preventing unsafe conditions before they emerge. The proposed framework may be implemented through a progressive top-down design process that begins with externally observable operational behavior and gradually refines the internal control architecture. The sequence reflects the principle that operational meaning should drive system structure, rather than emerging as a byproduct of component integration. Figure 8 shows a plan with stages of the operational design.

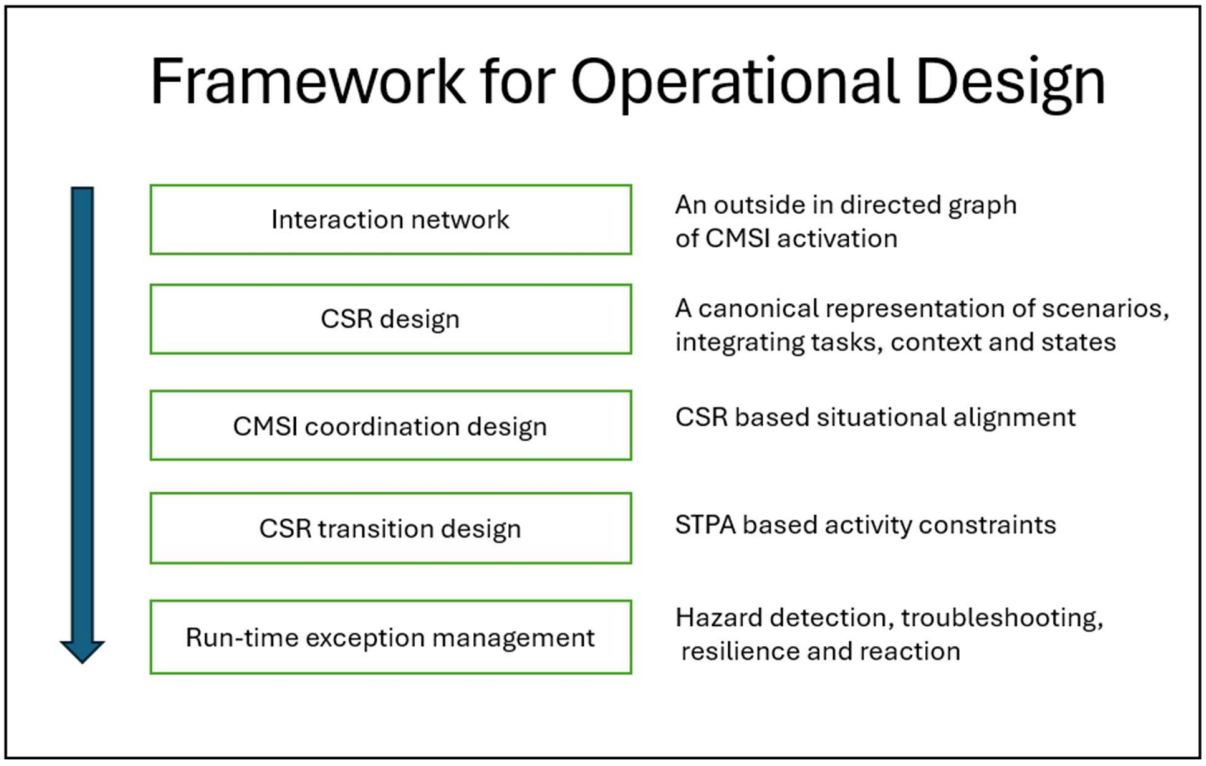

Figure 8.Framework for operational design

We start with a definition of the system functions and features used to get these functions, then define the scenarios that should constrain the system states, then design the situational alignment in the CMSI, going down to control actions by STPA, and finally design the run-time reaction to exceptions, comprising the detection, alarms, troubleshooting, resilience (Hollnagel, Woods, and Leveson 2006; Woods 2015) and recovery.

Enforcement is based on two complementary mechanisms: preventing inconsistent scenarios, and detecting/interpreting deviations when they appear. The approach restricts the set of possible behaviors relative to the active operational scenario. Within each CMSI, only actions matching the scenario are permitted, while inconsistent commands are blocked upfront, reducing the need for procedural protections and human corrective intervention.

In the proposed methodology, the design process advances 'outside-in'. First, the operational boundary between the system and its environment is defined, and only then are internal mechanisms examined. The analysis focus is on external interactions, services provided, connections with operators, collaborative systems, and the operational environment. From these external interactions, the operational network is derived, representing the distributed coordination structure between controllers, services, and operational assumptions. The AF296 accident demonstrates how inconsistent assumptions between pilot intent and thrust service states created a hazardous coordination structure, even though each component independently operated reasonably.

Following the definition of external interactions, the operational space is structured using scenarios and operational state representations. The design includes defining the ownership mission, characterizing the operational context, and deriving relevant controller states. Scenarios minimize ambiguity by creating coherent operational narratives, while the CSR layer translates these narratives into operational control states. The Air France 447 accident illustrates how differing interpretations of the same state between the autopilot and human pilots created coordination gaps that led to inappropriate responses during the stall.

The next phase develops the CMSI coordination architecture, responsible for maintaining interpretation consistency across system components. The architecture includes scenario-to-state mapping, operational state representations, component simulation via digital twins (Grieves and Vickers 2017), and mechanisms to preserve consistency between distributed process models (Leveson 2011). A CMSI mismatch arises when operational interpretations become mutually inconsistent despite active coordination dependencies. The Therac-25 accidents show how valid local operational assumptions produced a hazardous global state due to a mismatch between the radiation controller and the physical tray position. The proposed approach counters this complexity by bounding permitted transitions and commands per scenario instead of enumerating all possible system states.

Subsequently, the transition control architecture is developed to manage interaction dynamics and time-dependent behavior. This phase includes preview mechanisms for early prediction, transition constraint analysis, verification of timing relationships and action sequences, and operational decision support. The emphasis shifts from static state evaluation to understanding potential future system evolution. The Air China Nagoya accident demonstrates a situation where the automation transitioned to TO/GA mode while the pilots continued acting under normal landing assumptions. The risk arose not from a single fault but from a lack of synchronization in the transition between human and automated control states, highlighting the need to pre-identify hazardous interaction sequences, feedback delays, and inconsistent timing conditions.

The theory also addresses troubleshooting and operational resilience (Hollnagel, Woods, and Leveson 2006; Woods 2015). Troubleshooting is designed to return the system to an acceptable functioning level under time pressure and partial information, rather than providing an exhaustive explanation of the failure. While traditional systems relied on heuristics and pattern recognition, modern systems include backup mechanisms, including the reconstruction of state sequences, decisions, and interactions that led to the anomaly. The final stage involves exception and resilience planning, assuming that all conditions cannot be foreseen. This stage includes alerting mechanisms, resilience strategies, STPA integration, and converting unsafe control conditions into operational defense structures. The core idea is that the system must not only prevent failures but continue operating in a controlled manner after coordination loss and recover from it. In summary, the chapter presents a hierarchical design path where operational intent constrains scenarios, scenarios constrain control states, and control states constrain system behavior and resilience mechanisms.

Discussion and Future Horizons

The text presents an operational safety engineering framework that expands STPA from static design analysis into runtime coordination. The primary argument is that in complex systems, safety does not arise solely from individual component integrity, but from the capacity of controllers, services, and operators to maintain a coordinated operational interpretation of the system state. Thus, many systemic failures are not the result of a single faulty component, but of a divergence between local process models (Leveson 2011), operational assumptions, and differing state perceptions. The proposed framework adds a runtime scenario coordination mechanism to STPA using the CSR, which acts as a shared operational reference to synchronize distributed components. Rather than treating safety merely as the prevention of unsafe control actions, the system continuously monitors whether all participants still share the same understanding of the operational situation, mission context, and operating conditions. This maintains STAMP's core assumptions while expanding the operational aspect into dynamic coordination.

The article distinguishes between the physical state of the system and its operational interpretation. Controllers operate processes based on local models derived from partial observations and delayed feedback, whereas scenarios define bounded coordination classes containing rules for transition, synchronization, and permitted behavior. This approach addresses the combinatorial growth of the state space in modern systems. Instead of mapping every possible combination of states, the system operates within a limited number of operational scenarios with defined constraints, executing an engineering shift from exhaustive state enumeration to managing architectural constraints at a higher abstraction level.

The framework also alters the conception of runtime coordination. Instead of relying primarily on procedures and human coordination, coordination constraints are embedded directly into the architecture. The CSR functions as an operational invariant against which states and control actions are evaluated. Unsafe behavior is prevented not just via alerts, but through the structural exclusion of inconsistent operational configurations. This aligns with broad trends in resilient systems design, where safety is achieved via pre-bounded and pre-controlled operational structures. In the human factors domain, the framework views operator errors primarily as coordination phenomena rather than personal cognitive failures. In many accidents, operators acted reasonably relative to their local information, but the overall system lost operational consistency. Therefore, engineering responsibility shifts from an operator-centric view to designing systems that actively maintain consistency relative to the scenario and reduce cognitive load. Scenarios serve as operational abstractions that limit the number of interpretations an operator must hold simultaneously. However, a central role remains for human judgment under conditions of uncertainty or degraded functioning.

The paper stresses that complex systems will inevitably encounter unpredicted situations. Therefore, the framework does not attempt to guarantee complete correctness under every possible condition but rather bounds behavior during uncertainty. When a scenario-guided consistency breach occurs, the system transitions to more restricted, safe operational states to prevent uncontrolled escalation. Alongside the benefits, substantial limitations are noted. Successful implementation depends heavily on the quality of scenario definitions and transition logic precision. Furthermore, maintaining CSR consistency across distributed systems can generate synchronization overhead and delays. The framework also does not resolve fundamental failures such as software bugs, defective sensors, or cyber-attacks. Moreover, it remains at an architectural level and requires formalization of synchronization processes, consistency metrics, validation mechanisms, and runtime enforcement protocols.

Looking forward, the connection to MBSE (Madni and Sievers 2018; Friedenthal et al. 2014) and digital twins (Grieves and Vickers 2017) is presented. The framework integrates naturally with SysML models (Friedenthal et al. 2014), control structures, runtime monitoring, and resilient operational design. The CSR could eventually enable a bridge between design models and real-time operational supervision via scenario-oriented digital twins. Digital twins are presented as physical simulations for controller decision-making, but also as active support

mechanisms for preserving operational coherence, identifying inconsistencies, and validating scenario constraints.

A significant portion of the discussion is dedicated to the contribution of large language models (Bommasani et al. 2021; Weidinger et al. 2022) (LLMs (Bommasani et al. 2021)). AI models (Russell and Norvig 2021) are presented as supporting tools rather than operational authorities. In design, they can assist in organizing knowledge, identifying latent assumptions, generating alternative scenarios, and analyzing consistency. In STPA, they can assist in identifying unsafe control actions and managing the combinatorial complexity of hazardous interactions. At runtime, LLMs could serve as an adaptive interpretation layer that explains anomalies, merges information from disparate sources, and supports situational awareness. However, because their output is probabilistic and prone to semantic hallucinations, they cannot replace formal models and validated engineering constraints. In troubleshooting, LLMs are described as semantic correlation engines capable of unifying logs, reports, telemetry, and procedures into a single interpretative space, supporting hypothesis generation and operational reconstruction even with partial data. Yet, because they relate to semantic plausibility rather than causal truth, they can produce convincing but incorrect narratives. Thus, it is proposed to embed them within constraint-based diagnostic frameworks, where formal models and verification mechanisms remain the authoritative components.

The conclusion emphasizes a conceptual shift from safety engineering focused on individual components toward managing the coherence of distributed operational interpretations. The primary engineering goal becomes maintaining scenario-guided consistency and runtime coordination under conditions of uncertainty, partial observation, and operational changes. The proposed theory still requires formalization and empirical validation but is presented as a scalable architectural direction for complex socio-technical systems.

References

Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. “Concrete Problems in AI Safety.” arXiv preprint arXiv:1606.06565.

Bommasani, Rishi, Drew A. Hudson, Ehsan Adeli, et al. 2021. “On the Opportunities and Risks of Foundation Models.” arXiv preprint arXiv:2108.07258.

Boy, Guy A. 2013. *Orchestrating Human-Centered Design*. Berlin: Springer.

Boy, Guy A., and Jurgis Narkevicius. 2014. “Context-Based Design of Socio-Technical Systems.” *Proceedings of the European Conference on Cognitive Ergonomics*.

Dekker, Sidney. 2011. *Drift into Failure: From Hunting Broken Components to Understanding Complex Systems.* Farnham: Ashgate.

Dekker, Sidney. 2014. *The Field Guide to Understanding Human Error*. 3rd ed. Farnham: Ashgate.

Endsley, Mica R. 1995. “Toward a Theory of Situation Awareness in Dynamic Systems.” *Human Factors* 37 (1): 32-64.

Friedenthal, Sanford, Alan Moore, and Rick Steiner. 2014. *A Practical Guide to SysML*. 3rd ed. Waltham, US-MA: Morgan Kaufmann.

Grieves, Michael, and John Vickers. 2017. “Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems.” In *Transdisciplinary Perspectives on Complex Systems*.

Hollnagel, Erik. 2014. *Safety-I and Safety-II: The Past and Future of Safety Management*. Farnham: Ashgate.

Hollnagel, Erik, David D. Woods, and Nancy Leveson. 2006. *Resilience Engineering: Concepts and Precepts*. Farnham: Ashgate.

Hutchins, Edwin. 1995. *Cognition in the Wild*. Cambridge, US-MA: MIT Press.

Lee, Edward A. 2008. “Cyber Physical Systems: Design Challenges.” *Proceedings of the 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*.

Leveson, Nancy G. 2004. “A New Accident Model for Engineering Safer Systems.” *Safety Science* 42 (4): 237-270.

Leveson, Nancy G. 2011. *Engineering a Safer World: Systems Thinking Applied to Safe*ty. Cambridge, US-MA: MIT Press.

Madni, Azad M., and Michael Sievers. 2018. “Model-Based Systems Engineering: Motivation, Current Status, and Research Opportunities.” *Systems Engineering* 21 (3): 172-190.

Norman, Donald A. 1981. “Categorization of Action Slips.” *Psychological Review* 88 (1): 1-15.

Rasmussen, Jens. 1997. “Risk Management in a Dynamic Society: A Modelling Problem.” *Safety Science* 27 (2-3): 183-213.

Reason, James. 1990. *Human Error*. Cambridge: Cambridge University Press.

Russell, Stuart, and Peter Norvig. 2021. *Artificial Intelligence: A Modern Approach*. 4th ed. Hoboken, US-NJ: Pearson.

Sarter, Nadine B., and David D. Woods. 1995. “How in the World Did We Ever Get into That Mode? Mode Error and Awareness in Supervisory Control.” *Human Factors* 37 (1): 5-19.

Weidinger, Laura, John Mellor, Maribeth Rauh, et al. 2022. “Ethical and Social Risks of Harm from Language Models.” arXiv preprint arXiv:2112.04359.

Woods, David D. 2015. “Four Concepts for Resilience and the Implications for the Future of Resilience Engineering.” *Reliability Engineering & System Safety* 141: 5-9.

Acknowledgments

The author thanks Avigdor Zonnenshain, Valerie Sitterle, and Thomas Allen McDermott Jr. for their insightful comments. The author used AI-based tools (OpenAI ChatGPT and Google Gemini) to assist in drafting and editing the manuscript. The author is solely responsible for the content.

Appendix - Case studies

Analysis of well documented accidents indicates that many of them involve mismatch between the situation of the controller and at least one of the services.

Source of these cases: https://avi.har-el.com/eng/Articles/index.htm

These studies demonstrate the need to enforce coordinated situations between the controller and the services.

Transportation

	Controller	Situation	Service	Situation
AF 296 (1988)	Pilot	Pullup	Thrust	disable
AF 447 (2009)	Autopilot	On	Pivot tube	iced
➔	Pilot	Pullup	Airplane	stall
Nagoya (1994)	Pilot	Final Approach	TO/GA	activated
PL 603 altimeter (1996)	Pilot	Cruise flight	Altimeter	maintenance
Torrey Canyon (1967)	Helm	Manual control	wheel	disconnected

Industry

	Controller	Situation	Service	Situation
Bhopal (1983)	Operator	Normal	Container	Maintenance
TMI (1979)	Main pump	Disabled	Backup pump	In maintenance

Medical

	Controller	Situation	Service	Situation
Therac 25 (1985-87)	Treatment	X-ray	Tray position	Out

Missile Launch

	Controller	Situation	Service	Situation
Brahmos (2022)	Inspection	Inner state	Connectors	Junction box
Cheongung (2019)	Maintenance	Test cable	Connection	Functional

Friendly Fire Accidents

	Controller	Situation	Service	Situation
Tseelim A (1988)	Supported u.	Stage 2	Supporting u.	Stage 1
Tseelim B (1990)	Supported u.	Dry run	Supporting u.	Live ammunition
Kandahar (2001)	Supported u.	Tgt acquisition	GPS	Navigation

Home appliances

	Controller	Situation	Service	Situation
Air conditioning	Control unit	Activated	Activation	Delay
Home TV	Control unit	Scrolling	Mode	Tuning