Operation Design in Fifth Industrial Revolution Systems
A Coordination-Centered Runtime Architecture Complementary to System-Theoretic Safety

Avi Harel[1][a]
Ergolight, Haifa, Israel
Ergolight@gmail.com

Keywords: Operation design, System operation, Fifth Industrial Revolution, LLM, Safety, Usability, Productivity

Abstract: Fifth Industrial Revolution systems integrate automation, artificial intelligence, distributed services, and human operators within tightly coupled socio-technical environments. In such systems, operational failures frequently arise not from isolated component breakdown but from coordination mismatches that allow incompatible state configurations to coexist. System-theoretic safety frameworks such as STAMP provide rigorous methods for identifying inadequate control and deriving safety constraints (Leveson, 2011). However, they do not prescribe a canonical runtime execution architecture for enforcing structural state consistency. This paper introduces the Controller Multi-Service Interaction (CMSI) model, a coordination-centered runtime architecture that formalizes operational cycles, constrains admissible command sets by scenario, and enforces state-separation invariants that enforce coordination by construction. A minimal formal example demonstrates rule-based coordination enforcement. CMSI is positioned as complementary to system-theoretic safety: STAMP derives safety constraints, while CMSI provides an execution-level coordination substrate capable of enforcing them structurally. The architecture is differentiated from finite state machines, runtime monitoring, and contract-based design. Beyond safety, the model applies to promoting operational productivity and UX

1. INTRODUCTION

2. THE INDUSTRY NEEDS

Modern engineered systems operate within increasingly complex socio-technical environments characterized by tight coupling, automation, and interdependent subsystems. Across domains, operational obstacles continue to hamper safety, productivity, and user experience (UX). These obstacles often manifest not as immediate catastrophic failures, but as latent conditions that degrade system performance and create vulnerability to incidents.

The core problem is that operational obstacles: misalignments, hidden states, coordination gaps, accumulate within system operation. The challenge, therefore, is not merely to mitigate accidents after they occur, but to eliminate or prevent such obstacles from emerging and propagating during routine operation

2.1. Limits of Barrier-Based Safety

High-risk industries have historically relied on barrier-based safety models, most notably Reason's Swiss Cheese Model (Reason, 1990; 1997). In this view, accidents occur when latent conditions and active failures align across defensive layers.

While effective for hazard containment, barrier models are reactive. They assume hazards penetrate defences rather than addressing how incompatible operational states are allowed to coexist. Domain-specific standards (e.g., functional safety frameworks) improve compliance but do not eliminate coordination failures across interacting subsystems.

2.2. The Need for Feedback

Reason (1997) argues that most operational obstacles are latent rather than overt. Near misses, coordination

[a] https://orcid.org/0000-0002-8500-1533

mismatches, and scenario confusion frequently precede catastrophic events but remain uninvestigated. Dekker (2011) has described how organizational and developer biases discourage full transparency regarding system vulnerabilities. According to safety culture literature, developers may avoid providing feedback utilities or exception-tracking mechanisms, partly due to accountability concerns.

Furthermore, near misses are often underreported and insufficiently analysed, despite their recognized value in safety learning (Heinrich, 1931; Dekker, 2011). As a result, the industry lacks systematic tools and cross-domain standards for detecting, instrumenting, and investigating operational obstacles before they escalate.

The industry therefore requires both conceptual models and practical tools that enable systematic feedback, exception capture, and investigation support.

Research on tightly coupled systems has shown how unexpected interactions can generate cascading effects even when individual components function as designed (Perrow, 1984). System-theoretic approaches to safety further emphasize that accidents arise from inadequate control and flawed feedback within hierarchical structures (Leveson, 2011).

While these frameworks provide strong analytical tools, the increasing complexity of Fifth Industrial Revolution systems introduces challenges at the level of runtime coordination. The ability to identify unsafe control actions does not by itself guarantee that state configurations are structurally coordinated during execution.

This paper proposes the Controller Multi-Service Interaction (CMSI) model as a coordination-centered runtime architecture. This model comprises an execution-level structure within which operational coherence can be enforced through structural restriction of reachable state space and activity.

3. THEORY

3.1. Engineering Challenge

Barrier-based safety strategies face an inherent limitation: complex systems inevitably generate unforeseen interactions. Murphy's Law, "anything that can go wrong will go wrong", captures the practical implication of complexity under operational stress. In tightly coupled systems, small mismatches can propagate rapidly (Perrow, 1984).

The risk inherent in the Swiss Cheese model lies in its reactive orientation. It assumes hazards penetrate defences rather than questioning how operational states become misaligned.

Aviation and industrial case studies demonstrate that many failures stem not from the absence of barriers, but from mismatches between controller expectations and system states. The engineering challenge is to identify what can go wrong, not only in terms of component failure, but in terms of coordination failure.

3.2. Principle: From Swiss Cheese to "Chinese Wall"

The proposed shift replaces the circumstantial barrier metaphor with "Chinese Wall", a structural separation metaphor in which incompatible operational states are prevented from coexisting. Rather than tolerating latent holes in barriers, the system enforces scenario consistency and state coordination.

Artificial intelligence (AI), particularly model-based agents, can support "Chinese Wall" solutions by monitoring system states and enforcing coordination constraints. By contrast, AI is less effective when merely used to add additional reactive barriers.

3.3. The Trouble with Standards

As Tanenbaum famously remarked, "The nice thing about standards is that you have so many to choose from." The proliferation of standards reflects fragmentation across domains. While standards such as ISO 26262, IEC 61508, or aviation safety frameworks provide structured guidance, they do not offer a universal model of operational coordination.

Thus, reliance on standards alone cannot resolve systemic coordination failures.

3.4. Principle of Self-Protection

A core principle is self-protection: systems should prevent operational obstacles autonomously, rather than relying exclusively on external supervision. This aligns with resilience engineering principles emphasizing anticipation and containment (Hollnagel, 2014).

The primary challenge is identifying potential obstacles in advance.

3.5. Principle of Self-Awareness

Complementing self-protection is self-awareness: systems must maintain awareness of their operational state and context. In terms of human factors, this parallels the concept of situation awareness (Endsley, 1995), though here extended to system entities.

3.6. 2. Relationship to System-Theoretic Safety

STAMP conceptualizes safety as a control problem embedded in socio-technical hierarchies (Leveson, 2011). It models accidents as the result of unsafe control actions, missing feedback, or flawed process models. Its associated method, STPA, systematically derives safety constraints.

STAMP therefore addresses control adequacy. It does not prescribe a canonical runtime execution model, nor does it define structural reachability constraints for operational state spaces.

3.7. Restating the Concept of Exceptions

Operational obstacles can be conceptualized as exceptions, deviations from defined operational rules or scenario constraints. Failures are not discrete events but the result of operating within exceptional situations. This reframing aligns with Perrow's (1984) observation that accidents in complex systems emerge from interactions rather than isolated component breakdowns.

3.7.1. Predictable exceptions

Those are exceptions due to predictable failures, such as malfunction of a critical unit. These exceptions may be detected and identified using sensors specific to the source of failure, for example, sensors of unit availability embedded in the unit.

3.7.2. Unpredictable exceptions

Those are instances of failures which are unpredictable at design time.

3.8. CMSI as addon to STAMP

CMSI defines execution semantics in which activity constraints derived from system-theoretical analysis can be operationalized as structural invariants in the form of operational rules. STAMP is a general framework, used to identify safety-oriented constraints. CMSI provides architectural framework for defining and enforcing constraints that prevent typical failure modes. Besides safety, CMSI constraints also apply to other operational attributes, comprising office productivity and usability of consumer products.

4. A MODEL OF OPERATION

4.1. Formal definition of operation

The system operation may be described as a collection of CMSI connected by a network of tasks required to activate the interactions. Formally, the system operation may be defined as:

SysOp = (Task, CMSI operatioin, Feedback)
by loops: Task ➔ CMSI operation ➔ Feedback

4.2. A basic model of CMSI

Basic CMSI operation is defined as:

$$\text{CMSI operation} = (C, M, X, A, T, R)$$

where
C is the controller,
M is the set of services,
X is the global state space,
A is the set of commands,
$T: X \times A \rightarrow X$ is the transition function, and
R is a set of rules defining normal system operation.

This structure is compatible with transition-system representations used in formal verification and model checking (Clarke, Grumberg, & Peled, 1999).

Analysis of documented accidents across domains reveals a recurring pattern: mismatch between controller situation and service situation. The critical challenge is pinned in the complexity of the global state space X. If X is described as a collection of final state machines, then cardinality of X is the multiplication of the cardinalities of the state machines, and therefore, it grows exponentially with the number of state machines.

4.2.1. Challenge

The root cause for the complexity involved in writing STAMP-oriented specifications is that

straightforward definition of the transition function is too complex to manage. The implied methodological challenge is to find ways to reduce this complexity.

4.2.2. Fine tuning the basic model

The methods proposed here to tackle this challenge are by scenario-driven rule definition and by LLM-assisted rule validation.

4.3. Canonical CMSI Cycles

Critical complexity is often pinned to the interactions. Operation proceeds through a structured cycle:

1. Controller receives a task from the owner
2. Controller situation perception
3. Controller evaluation of control options
4. Controller option selection
5. Controller scenario setting
6. Controller sends a message to the services
7. Services react to the controller message
8. Mutual reaction in case of unexpected diversion from a rule
9. Services inform the controller about the situation
10. Controller verifies that the services reaction complies with the rules
11. Controller informs the owner about exceptions

Unlike general transition systems, this cycle explicitly embeds rule enforcement and admissible command restriction as execution semantics.

4.4. Minimal Formal Example

An example described here is of the Kandahar friendly fire accident in 2001. The accident resulted from breaking the coordination rule, due to changing of the GPS mode from target acquisition to navigation mode, in the GPS reset following battery replacement (Harel, 2024b).

State variables attributing to this accident are:

$$\text{Mode} \in \{\text{Nav}, \text{Target}\}$$
$$\text{Weapon} \in \{\text{Safe}, \text{Armed}\}$$
$$\text{TargetClass} \in \{\text{Friendly}, \text{Hostile}\}$$

and the coordination rule is:

$$\text{Weapon} = \text{Armed} \text{ implies } (\text{Mode} = \text{Target} \text{ and } \text{TargetClass} = \text{Hostile})$$

In conventional state machines or hybrid automata (Alur et al., 1995), such constraints may be expressed as guards or invariants. However, in the CMSI framework, reachable states depend on transition definitions and on the situational rules: admissible command computation excludes the Arm command unless rule conditions are satisfied. Thus, accessibility of regions of state space are enforced by construction.

4.5. Scenario-driven coordination

Failure is often attributed to unexpected activity. We can reduce the complexity growth from exponential down to linear, by instrumenting the operation scenarios and activating the state space by the active scenario. A scenario-driven CMSI is defined as:

$$S = (C, M, X, F, T, R, \Sigma)$$

In which
C is the controller,
M the set of services,
X the global state space,
Σ a scenario classification function,
F the set of commands,
$R: \Sigma \rightarrow X$ the set of rules defining normal system behavior, and
$T: \Sigma \rightarrow \Sigma$ a scenario transition function, independent of F.

Example of definition of operation rule is:

$$\text{If rule } r(x_t) = \text{false}$$
$$\rightarrow \text{raise coordination exception } \varepsilon$$
$$\rightarrow \text{enter recovery mode, then}$$
$$\rightarrow \text{restrict } F_t \text{ to recovery function set } F^R$$

4.6. Embedded model-based LLM

New AI technologies enable learning the system behavior in cases of unit or component failure, or in case of predicable external hazards. This learning may be accelerated by simulation of system behavior in cases of failure. Embedded LLM may sense extreme conditions and assist in the troubleshooting by matching these data with those predicted by the simulation.

4.6.1. A simple example

A simple example of predictable failure is wrong sensory data. Three cases of this failure mode were

demonstrated by Harel (2024a), showing the missed opportunity for knowledge transfer about this kind of failure (Proton M), and about the role of scenario dependent risk evaluation (PL 603).

5. RULE-BASED DESIGN

5.1. Rules for exception prevention

Preventive rules evaluate potential controller decisions before execution. When combined with digital twins—virtual representations of system services—they enable previewing consequences under alternative scenarios. Digital twin concepts are widely recognized in cyber-physical systems research (Tao et al., 2019).

Preventive rules therefore support anticipatory control rather than post-event mitigation.

5.2. Reaction to exceptions

Systems must detect exceptional situations and respond via alerting, troubleshooting, or auto-recovery. This requires:

- Formal rule definition.
- Instrumentation (probes).
- State tracking.
- Recovery procedures.

Loss of Control (LOC) accidents illustrate how operators often lack the information needed to understand unexpected states.

5.3. Rules for reacting to exceptions

Reactive structures address both expected and unexpected hazards:

- **Alerting rules**: Triggered by unexpected or external hazards.
- **Rebounding rules**: Correct operator-induced errors.
- **Protective rules**: Activate safeguards following alerts.
- **Recovery rules**: Manage anticipated hazard patterns.

These categories reflect resilience engineering principles, which emphasize monitoring, responding, learning, and anticipating (Hollnagel, 2014).

5.4. Reacting to surprise

Unexpected hazards often arise from unmodeled interactions, a phenomenon consistent with Normal Accident Theory (Perrow, 1984).

The design challenge is to protect the system when operating in unknown conditions, with unknown risks. The design should deal with alerting, safe mode, stopping, reporting

Model-based large language models (LLMs) offer new capabilities for analyzing exception patterns and supporting root cause analysis (RCA). However, their effectiveness depends on structured operational models rather than free-form inference.

Special sensors must be installed to detect exceptional conditions, such as extreme ambient temperature. Special routines should be developed for troubleshooting the unpredictable.

5.5. Operation under risk

The operation under risk may be defined for both predictable and unpredictable exceptions. Typical options for service reaction are auto stop, pause, shut down or safe-mode operation. The selected option depends on the risk of the particular failure. Typical controller reaction is monitoring the services under risk and informing the owner about the problem.

6. ARCHITECTURAL DIFFERENTIATION

6.1. Distinction from Guarded State Machines

Finite state machines define states and guarded transitions. Guards validate selected actions. CMSI differs in computing admissible command envelopes prior to selection. The controller is structurally restricted to a bounded decision-space. This pre-decision restriction shifts safety enforcement from transition validation to decision-space bounding.

While equivalent behavior can be encoded in a sufficiently detailed state machine, CMSI mandates this architectural separation as part of execution semantics rather than as a modelling choice.

6.2. Scenario Abstraction Layer

Traditional transition systems operate directly over raw state variables. CMSI introduces an explicit

scenario abstraction Σ mapping operational states into semantic coordination contexts. This intermediate layer reduces complexity and supports bounded decision computation.

This architectural separation is not intrinsic to general transition systems.

6.3. Invariant Preservation as Structural Reachability

Formal verification techniques can prove invariants over transition systems (Clarke et al., 1999). Runtime verification frameworks monitor execution traces for property violations (Leucker & Schallhart, 2009). CMSI differs in embedding invariant preservation into execution semantics by constraining admissible commands such that violation states are unreachable.

This shifts invariants from verification artifacts to runtime architectural commitments.

6.4. Distinction from Runtime Monitoring

Runtime monitoring observes behavior and flags violations after they occur (Leucker & Schallhart, 2009). CMSI aims to prevent entry into forbidden state regions by bounding command sets before execution. Monitoring is reactive; CMSI is structurally preventative.

6.5. Distinction from Contract-Based Design

Contract-based design specifies assumptions and guarantees between components (Benveniste et al., 2018). Such contracts are typically interface local. CMSI invariants operate at global coordination scope across controllers and services, constraining reachable system-level configurations rather than local interaction pairs.

7. DISCUSSION

The framework shifts software engineering from performance-centric optimization toward learning-centric resilience. Methodologically, this requires:

- exception-aware requirements
- architecture grounded in operational utility
- AI-assisted trace analysis
- cultural commitment to learning

Organizations must replace tacit acceptance of rare failures with explicit negotiation of risk investment (Dekker, 2011). Learning from failure becomes a driver of both safety and innovation.

7.1. Contributions and Novelty

This work does not merely review safety engineering or AI applications. It proposes a unified architectural framework that connects previously separate research traditions. The main contributions are:

1. **Utility-centered system value model.** We formalize system value as operational utility combining performance, resilience, and development efficiency. Unlike classical performance optimization, this model explicitly incorporates latent costs of rare events.
2. **Exception-oriented design paradigm.** We elevate exceptions to first-class architectural objects. Failures are treated as structured inputs to design rather than post-hoc anomalies, enabling systematic learning loops.
3. **AI-assisted operational learning framework.** We position LLMs as infrastructure for extracting scenario families from historical incidents, transforming narrative failure data into reusable engineering knowledge.
4. **Integration of HSI, safety engineering, and AI.**
   Existing literature treats these domains separately. This paper synthesizes them into a coherent engineering methodology centered on continuous adaptation.

7.1.1. Novelty claim

Safety engineering emphasizes resilience (Leveson, 2011), human factors research critiques rationality assumptions (Hollnagel, 2019), and DevOps promotes feedback-driven improvement (Humble & Farley, 2010). However, no prior framework explicitly combines:

operational utility
+ exception-first architecture
+ AI-driven cross-system learning

as a unified design paradigm.

This paper positions learning from failure not as a safety add-on, but as the organizing principle of 5IR software architecture.

7.2. Redefining System Value: Operational Utility

Traditional metrics describe performance under nominal conditions. Yet systems optimized for nominal efficiency accumulate hidden fragilities that surface under stress (Hollnagel, 2019).

Operational utility
= performance + resilience to adversity

Resilience includes detection, adaptation, and recovery capacity. Rare events generate disproportionate economic and societal costs (Reason, 1990). Development organizations often underestimate these costs due to visibility bias (Dekker, 2011).

Explicit modelling of latent operational costs reframes engineering tradeoff. AI-assisted analysis can expose historical risk patterns, challenging implicit assumptions about acceptable fragility.

7.3. Human–System Interaction Limits

Human operators are adaptive components of control loops. Classical models assume rational behavior under stress, but empirical evidence shows that decision-making is shaped by subjective perception and habit (Hollnagel, 2019).

Under emergency conditions, operators revert to routines optimized for normal operation (Reason, 1990). Bainbridge's ironies of automation demonstrate that increased automation reduces operator readiness for exceptions (Bainbridge, 1983). HSI optimization therefore requires:

- design for degraded modes
- interpretable exception signals
- cognitive support under stress
- explicit modeling of human limitations

System utility is inseparable from human–machine collaboration quality.

7.4. Human–System Interaction Limits

Human operators are adaptive components of control loops. Classical models assume rational behavior under stress, but empirical evidence shows that decision-making is shaped by subjective perception and habit (Hollnagel, 2019).

Under emergency conditions, operators revert to routines optimized for normal operation (Reason, 1990). Bainbridge's ironies of automation demonstrate that increased automation reduces operator readiness for exceptions (Bainbridge, 1983).

HSI optimization therefore requires:

- design for degraded modes
- interpretable exception signals
- cognitive support under stress
- explicit modeling of human limitations

System utility is inseparable from human–machine collaboration quality.

7.5. Utility Optimization and Risk Surfaces

System behavior can be conceptualized as a performance–error tradeoff curve (Leveson, 2011). Maximizing throughput often increases error likelihood. Engineering decisions define an operational point on this curve.

Real systems involve multidimensional risk surfaces including safety, availability, security, and human workload. Utility optimization becomes navigation across interdependent risk dimensions rather than single-metric optimization.

AI enables empirical refinement of these tradeoffs based on cross-system evidence.

7.6. AI Integration in the 5IR

In 5IR, AI acts as a context-aware collaborator. LLMs can synthesize large corpora of incident narratives to identify scenario families, weak signals, and recurring failure patterns. Applications include:

- preventive troubleshooting
- anomaly clustering
- scenario-dependent behavior analysis
- inter-unit coordination insights

Historical failures such as Therac-25 show how rare interactions escape traditional hazard analysis (Leveson & Turner, 1993). AI-assisted synthesis

converts isolated failures into structured design knowledge.

Although rare-event probabilities remain difficult to estimate, qualitative pattern recognition improves preparedness and response.

7.7. Exception-Oriented Design

Exception-oriented design treats deviations from defined norms as primary architectural entities. Systems explicitly model:

- normal operation rules
- exception classes
- reaction strategies
- recovery trajectories

Exceptions become probes for learning.

Continuous tracing and feedback loops support systematic capture of anomalies (Humble & Farley, 2010; Allspaw & Robbins, 2010). This aligns with DevOps principles of continuous improvement and organizational learning. Effective exception management requires:

- detectability through tracing
- hierarchical alerting
- graded responses
- affordable modelling
- rule-based scenario abstraction

LLMs assist by extracting exception classes, clustering narratives, and proposing recovery strategies.

7.8. Mapping Safety Constraints into CMSI

Safety constraints derived through system-theoretic analysis can be translated into CMSI invariants and admissible command restrictions. For example, a prohibition on weapon arming in navigation mode becomes a rule restricting reachable states. System-theoretical analysis identifies the requirement. CMSI provides the structural mechanism to enforce it at runtime.

7.9. Implications for Usability and Productivity

By bounding admissible command sets, CMSI reduces exposure to contextually invalid actions. Established human factors principles suggest that reducing irrelevant alternatives may reduce error likelihood and cognitive workload (Norman, 2013; Wickens & Hollands, 2000). These implications remain hypotheses requiring empirical validation.

7.10. Implications for Autonomous Systems

For AI agents, embedding decision-making within invariant-constrained state spaces restricts exploration of unsafe regions. Such containment may be relevant in high-risk systems integrating autonomous components, although empirical validation remains future work.

7.11. Boundary Conditions

CMSI assumes discrete state representation and definable transition functions. Continuous systems may require hybrid extensions similar to hybrid automata frameworks (Alur et al., 1995). CMSI does not replace safety analysis or formal verification; rather, it provides execution-level semantics within which those analyses can be operationalized.

8. CONCLUSION

The CMSI model introduces a coordination-centered runtime architecture characterized by a canonical interaction cycle, scenario abstraction, pre-decision command-envelope restriction, structural invariant preservation, and coordination exception semantics. Positioned as complementary to system-theoretic safety, CMSI provides an execution-level substrate capable of enforcing safety constraints while bounding decision spaces in complex socio-technical systems.

Future work includes formal verification through model checking, simulation-based evaluation of reachable-state restriction, and empirical assessment of human and AI performance under bounded decision architectures.

REFERENCES

Allspaw, J., & Robbins, J. (2010). *Web operations: Keeping the data on time*. O'Reilly Media.

Alur, R., Courcoubetis, C., Henzinger, T. A., & Ho, P. H. (1995). Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. *Theoretical Computer Science*, 138(1), 3–34.

Bainbridge, L. (1983). Ironies of automation. *Automatica*, 19(6), 775 to 779. https://doi.org/10.1016/0005-1098(83)90046-8

Benveniste, A., Caillaud, B., & Passerone, R. (2018). Contracts for systems design. *Foundations and Trends in Electronic Design Automation*, 12(2–3), 124–400.

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. MIT Press.

Dekker, S. (2011). *Drift into Failure*. Ashgate.

Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors*.

Heinrich, H. W. (1931). *Industrial Accident Prevention*. McGraw-Hill.

Hollnagel, E. (2014). *Safety-I and Safety-II: The Past and Future of Safety Management*.

Hollnagel, E. (2019). *The ETTO Principle: Efficiency-Thoroughness Trade-Off*. CRC Press.

Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley.

Leucker, M., & Schallhart, C. (2009). A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5), 293–303.

Leveson, N. (2011). *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press.

Leveson, N. G., & Turner, C. S. (1993). *An investigation of the Therac-25 accidents*. IEEE Computer, 26(7), 18–41. https://doi.org/10.1109/MC.1993.274940

Norman, D. A. (2013). *The Design of Everyday Things* (Revised and Expanded Edition). Basic Books.

Perrow, C. (1984). *Normal Accidents: Living with High-Risk Technologies*. Basic Books.

Reason, J. (1990). *Human Error*. Cambridge University Press.

Reason, J. (1997). *Managing the Risks of Organizational Accidents*. Ashgate.

Tanenbaum, A. S. (1989). *Computer Networks*. Prentice Hall.

Tao, F., Zhang, H., Liu, A., & Nee, A. Y. C. (2019). Digital twin in industry: State-of-the-art. *IEEE Transactions on Industrial Informatics*.

Wickens, C. D., & Hollands, J. G. (2000). *Engineering Psychology and Human Performance* (3rd ed.). Prentice Hall.

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.

	Controller	**Situation**	**Service**	**Situation**
Transportation				
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				
Bhopal (1983)	Operator	Normal	Container	Maintenance
TMI (1979)	Main pump	Disabled	Backup pump	In maintenance
Medical				
Therac 25 (1985-87)	Treatment	X-ray	Tray position	Out
Missile Launch				
Brahmos (2022)	Inspection	Inner state	Connectors	Junction box
Cheongung (2019)	Maintenance	Test cable	Connection	Functional
Friendly Fire Accidents				
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				
Air conditioning	Control unit	Activated	Activation	Delay
Home TV	Control unit	Scrolling	Mode	Tuning

Source of this table: https://avi.har-el.com/eng/Articles/index.htm

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