International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(21s) 2026 pp. 1967-1975
https://doi.org/10.70917/ijcisim-2026-5972
Article

Contract-Governed Orchestration: A Failure-Mode Analysis of AI-Driven Automation in Real-Time Enterprise Integration

Dillibabu Arumugam

Independent Researcher, USA

Abstract: A payment-reconciliation agent that closes an exception ninety seconds after the underlying account reopened has not made a bad prediction. It has acted on a picture of the world that no longer existed.

This article argues that failures of this kind - now appearing across enterprises that have inserted AI-driven decisioning into real-time orchestration layers - are architectural rather than statistical, and that the field's near-exclusive focus on model accuracy and explainability has left the actual failure surface under-examined. Four recurring classes are identified from converging literature on agentic middleware, schema evolution, and enterprise AI-agent security: stale-context decisioning, boundary overreach, cascading compensation failure, and unattributed autonomous action.

Against each, a governance mechanism is proposed: freshness-attributed schema contracts, tiered and earned autonomy budgets, simulation gating on consequential cross-process actions, and mandatory attributed rollback. Survey evidence from 445 IT and security professionals indicates that 53 per cent of organisations have observed AI agents exceeding intended permissions occasionally or sometimes, and 47 per cent experienced a security incident involving an AI agent in the preceding year [9].

The article's position is that contract governance, not model capability, is the binding constraint on how much orchestration-layer autonomy an estate can safely grant. Enterprises scaling automation without first building these guarantees are not moving faster; they are relocating their failures to where nobody is watching.

Keywords: enterprise integration; real-time orchestration; AI-driven automation; contract governance; failure-mode analysis; agent autonomy

1. Introduction

Ninety seconds separated two events in a payment-reconciliation workflow. A compensating transaction reopened an account, and an automated agent, reading from a cache that had not yet caught up, closed it as routine. No model produced a wrong prediction that day. The agent did precisely what it was built to do, with information that was, by the time it acted, already false.

This is not the failure the automation literature spends most of its attention preparing for. Bias, explainability, and prediction accuracy dominate the conversation about AI risk. Each is a legitimate concern. None describes what happened in the reconciliation workflow.

What happened was architectural. An orchestration layer built to coordinate deterministic, human-reviewed steps acquired an autonomous decision-maker at one of those steps, and the layer's existing staleness and boundary problems did not disappear. They were exercised faster, with more apparent confidence, and with far less visibility into what the agent did not know it did not know. Orchestration layers of this kind typically rest on durable, replayable event logs designed to give any consumer a consistent view of what has happened [2], carried inside standardised event envelopes and interface descriptions intended to make that view interpretable across producers and consumers [7], [8]. Neither mechanism guarantees that the view a consumer acts on is current. Both were built to make history



Copyright: © 2026 by the authors

replayable, not to make staleness visible at the moment of decision. *Figure 1 illustrates the reconciliation gap this scenario turns on.*

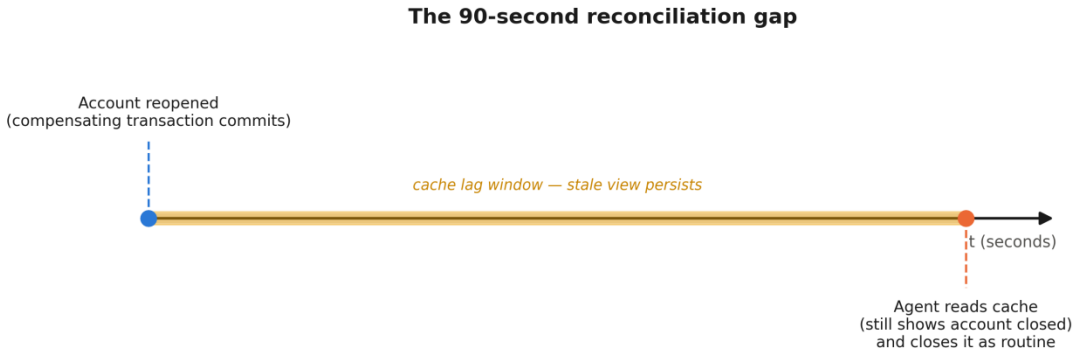


Figure 1: The 90-second reconciliation gap. The compensating transaction commits at t = 0; the agent acts on a stale cached view at t = 90, ninety seconds after the state it is reasoning about had already changed.

Security researchers have begun putting numbers to the pattern. In a survey of 445 IT and security professionals, 53 per cent of organisations reported AI agents exceeding intended permissions occasionally or sometimes, and 47 per cent reported a security incident involving an AI agent within the previous year [9]. Read carefully, these are not model failures. They are boundary failures - precisely the seam between what an orchestration platform's contracts specify and what an autonomous agent is actually permitted to attempt.

That seam widens for a reason specific to orchestration platforms rather than to automation generally. A workflow engine already carries an implicit theory of who is allowed to do what, encoded across configuration files, access-control lists, and tribal knowledge about which team owns which process step. That theory was tolerable when the only actors executing steps were fixed rules or humans who could be trusted to notice when something looked wrong. An autonomous agent does not notice in the same way. It executes within whatever boundary the platform actually enforces, not the boundary its designers intended, and the gap between the two becomes the terrain on which the four failure modes in this article take shape.

Contributions. This article makes three contributions. First, it separates orchestration-layer automation failure from model-layer failure and argues the former has received disproportionately little structured attention. Second, it derives a four-class failure taxonomy organised by the architectural layer at which each failure originates, so that each class admits a categorically different remedy. Third, it maps each class to a specific governance mechanism and argues that the four mechanisms are interdependent rather than a menu, identifying which two an estate typically adopts and which two it typically defers.

Section 2 treats agentic middleware research, schema-evolution practice, and enterprise AI-agent risk data as three strands of one argument. Section 3 sets out the method. Section 4 presents the taxonomy and the governance framework. Section 5 tests where the framework holds. Section 6 concludes.

2. Literature Review

2.1 Coordination style: flexibility against traceability

Classical orchestration research separates centralised coordination, in which a workflow engine directs participants explicitly, from choreography, in which participants react to events with no central authority [1]. Agentic middleware complicates that separation by making the coordinator itself an autonomous actor - a system capable of interpreting goals, discovering services dynamically, and correcting its own course [5].

The consequence is not trivial. A workflow engine that fails does so along a traceable rule. An agentic orchestrator that composes its own flow at runtime fails in ways that depend on what it perceived in the moment,

which is a much harder object to reconstruct afterwards [5]. Neither strand of the literature resolves what should govern this trade between flexibility and traceability; it simply gets displaced from one design choice to the next.

The displacement matters more in heterogeneous estates than the literature's framing acknowledges. Agentic middleware work tends to evaluate autonomous composition against a relatively uniform service catalogue, where every participant exposes comparable interfaces and comparable guarantees. Carrier-scale integration environments rarely look like that. An autonomous orchestrator composing a flow across a decades-old billing mainframe, a cloud-native customer-data platform, and a third-party fraud-scoring service is not choosing among functionally interchangeable options. It is choosing among participants with wildly different latency profiles, consistency guarantees, and failure semantics - a distinction the flexibility-versus-traceability framing does not capture.

2.2 Schema compatibility was built for a different failure

Backward- and forward-compatible schema evolution, enforced at the point of publication, is what lets heterogeneous producers and consumers evolve independently without breaking each other [4]. That guarantee was designed for a downstream service parsing fields it expects. It offers no protection to an autonomous agent that infers meaning from a field's absence, or that treats a record as current because it is schema-valid rather than because it is fresh.

The distinction is easy to understate, because both failures look identical from the outside: a consumer receives a message and acts on it incorrectly. The remedies diverge sharply. Compatibility discipline is a versioning practice, enforced once, at schema-change time, by the team that owns the producer. Freshness discipline is a runtime practice, enforced continuously, by the consumer. No amount of rigour in the former substitutes for its absence in the latter. A schema can remain perfectly compatible for years while the freshness of any individual message drifts wildly with network conditions, cache invalidation timing, or upstream processing delay - none of which a compatibility contract was designed to observe.

2.3 Authority placement: prescription against evidence

Production guidance for agentic workflows converges on a reasonable-sounding consensus: verify between planning and execution, bound tool access, escalate when confidence drops [5]. Independent survey data suggests this consensus is not yet operating as designed. Fifty-three per cent of organisations report agents exceeding intended permissions, and 54 per cent report between one and one hundred unsanctioned AI agents in their environment, with ownership frequently unclear - only 15 per cent report that three-quarters or more of their agents have defined ownership [9]. The gap between what the prescriptive literature assumes and what the empirical literature finds is itself the strongest evidence for this article's position.

Multi-agent safety research adds a layer that single-agent guidance does not anticipate. Where one autonomous orchestrator raises questions about bounded scope, multiple cooperating agents raise questions about compounded scope. Security properties are not compositional across interacting agents: a boundary violation by one can propagate as a false input to another, and the resulting failure has no single point of origin a post-incident review can isolate cleanly [6]. Recent work formalises one version of the required discipline for multi-agent query resolution by inserting an explicit verification stage between planning and execution, so that no step proceeds without it [16] - a structure closely matching the simulation gating proposed in Section 4.2. That proposal's evaluation covers 25 curated queries, so it should be read as a design precedent rather than as validation at scale.

Regulatory guidance converges from a third direction. Article 14 of Regulation (EU) 2024/1689 requires that high-risk AI systems be designed so that natural persons can effectively oversee them, including the ability to disregard, override, or reverse the output [17]. Risk-management guidance treats continuous reassessment as a structural component rather than a post-deployment addition [18]. Both bear directly on the tiered autonomy budgets proposed in Section 4.2.

2.4 Grounding is a property of the pipeline

A quieter fourth thread connects the others. Retrieval-augmented generation, when the retrieved context is current and well-scoped, reduces hallucination substantially: one industry-track study found hallucinated steps as high as 21 per cent without a retriever, falling below 7.5 per cent for steps and below 4.5 per cent for tables once retrieval was introduced [10]. The parallel to orchestration is direct, even if the literature has not drawn it explicitly. A model grounded in context it trusts is only as reliable as that context is fresh, exactly as an orchestration agent is only as reliable as the state it reads before acting. Grounding is not a property of the model. It is a property of the pipeline - and pipelines, unlike models, are squarely within the integration architect's remit to design, instrument, and hold accountable [20].

3. Materials and Methods

No controlled experiment sits behind this article. What follows is a structured failure-mode analysis, a method suited to an architectural contribution rather than a laboratory study, and consistent with the vantage point of a practising integration architect synthesising recurring production patterns.

The analysis moved through three stages. First, candidate failure patterns were drawn from the converging literature in Section 2: agentic middleware and multi-agent safety research on how autonomous coordination departs from classical orchestration [5], [6]; schema-evolution practice on the compatibility guarantees available and where they stop [4]; and enterprise security survey data on where those guarantees break down in deployment [9]. Second, each candidate was checked against distributed-systems reliability literature - retry and circuit-breaker behaviour under failure [13], [14] and reliable event processing under idempotency constraints [4] - to confirm it reflects a documented architectural failure rather than a single anecdote. Third, each confirmed failure mode was matched to a governance mechanism addressing its specific root cause.

Scope decision. The analysis restricts itself to failure modes originating at the orchestration and integration layer, deliberately excluding those originating inside a model's training or inference process, such as adversarial input manipulation or training-data poisoning. This is not a claim that those are unimportant. It reflects a judgment that the orchestration-layer failure surface has received disproportionately less structured attention relative to its apparent frequency in the survey evidence reviewed [9], and that conflating the two surfaces in one taxonomy would blur exactly the distinction Section 2 argues is being lost.

The resulting taxonomy is organised along two axes: the architectural layer where a failure originates - perception, decision boundary, cross-system coordination, or accountability - and the mechanism through which it surfaces in production. This structure exists because failures originating at different layers need categorically different responses. The mechanism that fixes an attribution problem does not fix a staleness problem. Much of the current automation literature discusses AI risk as one undifferentiated category, which obscures precisely this distinction [22].

4. Results

4.1 A failure-mode taxonomy

Four failure classes recur across the literature, each aligned to one architectural layer.

Stale-context decisioning occurs when an autonomous agent acts on state that changed after it was last observed - a direct consequence of orchestration architectures never designed to signal staleness to an autonomous consumer [4], [5].

Boundary overreach occurs when an agent's actual scope of action exceeds what was intended, either because the boundary was never formally encoded or because the agent composed tools and services in ways its designers did not anticipate [5], [6], [9].

Cascading compensation failure occurs when an autonomously triggered compensating action interacts with another in-flight process in a way no single component was positioned to detect. It is structurally close to the retry-

storm and cascading-failure patterns documented in distributed-systems resilience practice, where retry policy and circuit-breaker placement interact non-intuitively with observed error rates and response times [13], [14].

Unattributed autonomous action occurs when a decision cannot be traced afterwards to the state observed, the policy applied, and the outcome produced. Every incident review becomes forensic reconstruction. None becomes audit.

These four classes are not evenly distributed across an estate's automation portfolio, and the unevenness is diagnostic. Stale-context decisioning concentrates wherever an agent consumes state from a cache, materialised view, or replicated store rather than from the authoritative event stream, because replication introduces exactly the propagation delay the failure depends on. Boundary overreach concentrates wherever tool access was provisioned broadly for convenience during initial deployment and never subsequently narrowed - a pattern consistent with the unsanctioned-agent survey figures [9]. Cascading compensation failure concentrates in sagas spanning three or more services, where the combinatorics of overlapping in-flight compensations grow faster than any single team's mental model. Unattributed autonomous action concentrates wherever logging was designed for debugging rather than audit, capturing what an agent did without capturing what it believed to be true when it did it.

Table 1: Failure-mode taxonomy for AI-driven automation in real-time orchestration

Failure class	Architectural layer	Root cause	Evidence signal
Stale-context decisioning	Perception	No staleness contract between event backbone and consuming agent	Agentic self-correction is reactive, not preventive [5]
Boundary overreach	Decision boundary	Scope encoded implicitly or unenforced at runtime	53% of organisations report agents exceeding intended permissions [9]
Cascading compensation failure	Cross-system coordination	Compensating actions lack cross-process visibility	Retry and breaker configuration interact non-intuitively with error rate [13], [14]
Unattributed autonomous action	Accountability	No mandatory decision record tied to observed state and applied policy	54% report unsanctioned agents; only 15% report defined ownership for most agents [9]

4.2 A contract-governance framework

Each failure class maps to one governance mechanism, not to a general appeal toward responsible AI.

Schema contracts carrying mandatory freshness attributes address stale-context decisioning by making an event's age a machine-readable field an agent must check before acting, rather than an assumption baked silently into the pipeline [19].

Tiered autonomy budgets address boundary overreach by binding an agent's permitted actions to an explicit, versioned scope that widens only as its track record within a narrower one earns it. A precedent for the shape of such tiers exists in security-operations research, which defines five autonomy levels from manual operation through fully autonomous, and reports - in a cyber-range simulation rather than a production deployment - false positives falling from 70 to 35 per cent and automated ticketing rising from 10 to 75 per cent as autonomy was calibrated across those levels [12]. The architectural principle appears sound across domains; the magnitudes should not be assumed transferable.

Simulation gating addresses cascading compensation failure by requiring any compensating action touching more than one in-flight process to run against a replayed model of current process state before it executes. Action-

oriented process mining provides the technical basis for constructing such a replay from an organisation's own event data [3].

Attributed rollback addresses unattributed autonomous action by requiring every autonomous decision to carry a record of what it observed and what policy it applied, with automatic, bounded reversal the moment that record cannot be produced. Any reversal must itself preserve idempotent processing semantics, so that a governance-driven rollback does not duplicate or partially undo an already-settled effect [4].

The four mechanisms are deliberately not independent, and treating them as a menu misses the framework's central design intent. Freshness attributes are what make tiered autonomy budgets enforceable, because a budget granting an agent authority to act on account state is only as safe as the guarantee that the state is current; without freshness attributes, a widened tier simply gives stale-context failure more room to occur. Simulation gating depends on the same event history freshness attributes are drawn from, since a replay cannot reconstruct state it was never given a faithful, timestamped record of. And attributed rollback is what makes the other three auditable rather than aspirational: a freshness attribute checked but never logged, or an autonomy budget enforced but never recorded against, provides no evidence after an incident that the guardrail functioned.

The four guardrails form a single load-bearing structure. An estate that implements only the two cheapest to build - typically freshness attributes and rollback - without the two requiring more upfront design work will find its governance incomplete in exactly the dimension where cascading and boundary failures concentrate [23].

Table 2: Contract-governance framework mapped to failure modes

Guardrail	**Target failure mode**	**Enforcement mechanism**	**Autonomy tier applicability**
Schema contracts with freshness attributes	Stale-context decisioning	Mandatory age field validated at consumption	All tiers
Tiered autonomy budgets	Boundary overreach	Versioned, explicit scope earned incrementally	Supervised through bounded-autonomous
Simulation gating	Cascading compensation failure	Replay-based verification before cross-process actions execute	Bounded-autonomous and above
Attributed rollback	Unattributed autonomous action	Mandatory decision record with automatic reversal on gap	All tiers

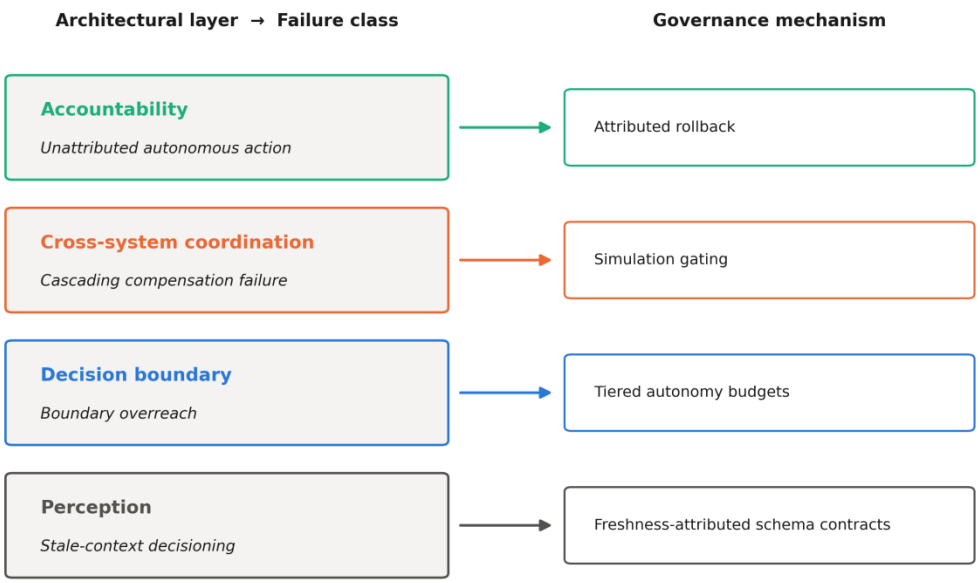


Figure 2: The failure-mode taxonomy mapped to its governance mechanism. Each architectural layer's failure class (left) is addressed by a specific guardrail (right); the four guardrails form a single load-bearing structure rather than an independent menu.

5. Discussion

The taxonomy reads differently depending on which estate it is held against, and that difference is informative. Inside a heterogeneous, carrier-scale environment - legacy operational-support systems, cloud-native services, and third-party platforms under continuous real-time load - all four failure classes appear with meaningful frequency, because the number of producers and consumers multiplies every surface where staleness, boundary ambiguity, and attribution gaps take hold. Inside a smaller, homogeneous stack owned end-to-end by one team, boundary overreach and unattributed action are less likely by construction. Contract governance is not a universal prescription. It is a response calibrated to estate complexity, and the calibration is where architectural judgment does the real work [19].

That calibration has organisational consequences worth naming. Freshness attributes and attributed rollback are comparatively cheap to introduce into an existing schema-registry discipline and can be added incrementally, service by service, without a coordinated cutover. Tiered autonomy budgets and simulation gating are not incremental in the same way [20]. They require a governing body - a platform team, an architecture review board, or some hybrid - with standing authority to grant and revoke scope, and a replay infrastructure faithful enough to production state that a simulated result is trustworthy enough to gate a live action on. Estates that adopt the cheap half and defer the expensive half are not partially governed. They are governed against two failure classes and exposed against the other two, and knowing which two matters more than knowing that governance has been implemented in the abstract [21].

There is a relationship worth stating plainly between this framework and the model-centric risk literature it does not try to replace. Explainability techniques that clarify why a model produced a given output remain valuable, particularly for the reasoning step inside an orchestration agent [11]. But explainability answers a different question than governance does. It explains a decision after it has executed; contract governance constrains which decisions are permitted to execute at all. An estate that invests only in the former can produce failures that are perfectly explainable and entirely avoidable [22].

Limitations. The taxonomy comes from published literature and architectural pattern-matching, not from a controlled deployment. The four guardrails, while each grounded in documented precedent individually, have not been tested together as one system in any source reviewed here. Two evidential caveats deserve explicit statement. The survey figures underpinning the boundary-overreach class come from a vendor-sponsored industry survey of 445 respondents, not from an academic instrument, and should be read as indicative of prevalence rather than as a precise population estimate [9]. The autonomy-tier precedent comes from a security-operations cyber-range simulation, not an integration context, and its reported magnitudes almost certainly do not transfer [12].

Future work. Closing the evidential gap needs a longitudinal case study this analysis does not attempt: tracking a single estate's incident rate across each of the four failure classes before and after the framework's introduction. A narrower and more immediately achievable test follows from Section 2.4. If grounding a model in current, sourced context cuts hallucination from above 21 per cent to below 7.5 per cent [10], the orchestration equivalent - grounding an agent's decision in current, contract-verified state - should reduce the structurally similar failure of stale-context decisioning [23]. No source reviewed here measures that equivalence. Instrumenting an orchestration layer with freshness attributes and measuring the stale-context failure rate before and after is the most direct empirical test this framework's central claim could be given [24].

6. Conclusion

Most of what fails when AI-driven automation runs inside real-time orchestration is not model judgment. It is architectural boundary, and it keeps failing because governance is still treated as a policy layer bolted on after deployment rather than a structural property designed in from the start.

This article separates four distinct failure classes, each traceable to a specific architectural layer, and assigns each a governance mechanism rather than a general instruction to proceed carefully. Underneath both is a single claim: autonomy inside orchestration should expand only as fast as an estate can prove it deserves to - through demonstrated freshness guarantees, scope that is bounded and earned rather than assumed, simulated verification before consequential actions execute, and an unbroken record of what every autonomous decision saw and did. Build those guarantees before scaling automation, and autonomy compounds safely. Scale first and govern afterwards, and the survey evidence already suggests what happens next. The failures do not disappear. They move somewhere harder to see.

Declarations

Funding. This research received no external funding.

Author contribution. D. Arumugam is the sole author and is responsible for all aspects of this work, including conceptualisation, literature synthesis, analysis, and writing.

Conflicts of interest. The author declares no conflict of interest.

Data availability. No proprietary datasets were used. All quantitative figures cited are drawn from the publicly available sources listed in the References.

Use of AI tools. Generative AI tools were used to assist with literature synthesis, drafting, and editing during the preparation of this manuscript. All analysis, interpretation, and conclusions are the author's own, and the final content was reviewed and approved by the author.

REFERENCES

1. G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley, 2003.
2. J. Kreps, N. Narkhede, and J. Rao, "Kafka: a distributed messaging system for log processing," in *Proc. 6th Int. Workshop on Networking Meets Databases (NetDB)*, Athens, Greece, Jun. 2011.
3. G. Park and W. M. P. van der Aalst, "Realizing a digital twin of an organization using action-oriented process mining," in *Proc. 3rd Int. Conf. on Process Mining (ICPM)*, 2021. doi: 10.1109/ICPM53251.2021.9576846

4. M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol, CA, USA: O'Reilly Media, 2017. (See Ch. 4, Encoding and Evolution.)
5. E. Bandara, R. Gore, P. Foytik, S. Shetty, R. Mukkamala, et al., "A practical guide for designing, developing, and deploying production-grade agentic AI workflows," arXiv:2512.08769, 2025. (Preprint; not peer reviewed.)
6. C. Schroeder de Witt, K. Krawiecka, I. Krawczuk, B. Hagag, W. L. Anderson, et al., "Open challenges in multi-agent security: Towards secure systems of interacting AI agents," arXiv:2505.02077, 2025. (Preprint; not peer reviewed.)
7. AsyncAPI Initiative, *AsyncAPI Specification*, version 3.1.0, Jan. 2026. [Online]. Available: https://www.asyncapi.com/docs/reference/specification/latest
8. Cloud Native Computing Foundation, *CloudEvents: A Specification for Describing Event Data in a Common Way*, version 1.0.2, 2022. (CNCF graduated project, Jan. 2024.) [Online]. Available: https://cloudevents.io
9. Cloud Security Alliance, "Enterprise AI security starts with AI agents," 2026. (Survey of 445 IT and security professionals, conducted Sep. and Nov. 2025; sponsored research.) [Online]. Available: https://cloudsecurityalliance.org/artifacts/enterprise-ai-security-starts-with-ai-agents
10. P. Béchard and O. Marquez Ayala, "Reducing hallucination in structured outputs via Retrieval-Augmented Generation," in *Proc. 2024 Conf. North American Chapter of the ACL: Human Language Technologies (NAACL), Industry Track*, 2024, pp. 228–238. doi: 10.18653/v1/2024.naacl-industry.19
11. A. Kovari, "AI for decision support: Balancing accuracy, transparency, and trust across sectors," *Information*, vol. 15, no. 11, art. 725, 2024. doi: 10.3390/info15110725
12. A. Mohsin, H. Janicke, A. Ibrahim, I. H. Sarker, and S. Camtepe, "A unified framework for human-AI collaboration in security operations centers with trusted autonomy," *ACM Transactions on Internet Technology*, 2026. doi: 10.1145/3837073
13. M. T. Nygard, *Release It! Design and Deploy Production-Ready Software*, 2nd ed. Raleigh, NC, USA: Pragmatic Bookshelf, 2018.
14. M. R. Saleh Sedghpour, C. Klein, and J. Tordsson, "An empirical study of service mesh traffic management policies for microservices," in *Proc. 2022 ACM/SPEC Int. Conf. on Performance Engineering (ICPE '22)*, 2022, pp. 17–27. doi: 10.1145/3489525.3511686
15. M. Mohammad, "Resilient microservices: A systematic review of recovery patterns, strategies, and evaluation frameworks," arXiv:2512.16959, 2025. (Preprint; not peer reviewed.)
16. X. Zhang, Y. Cui, G. Wang, Q. W. Qiu, Z. Li, F. Han, Y. Huang, H. Qiu, B. Zhu, and P. He, "Verified multi-agent orchestration: A plan-execute-verify-replan framework for complex query resolution," arXiv:2603.11445, 2026. (Preprint; not peer reviewed.)
17. European Parliament and Council of the European Union, Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act), Article 14: Human oversight, Official Journal of the European Union, 2024.
18. National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1. Gaithersburg, MD, USA: NIST, Jan. 2023. doi: 10.6028/NIST.AI.100-1
19. K. Rekulapalli, H. Nilagiri, and M. Vakkanthula, "A comparative evaluation of national FHIR Implementation Guides: Strategic methodologies for enhancing interoperability, regulatory alignment, and clinical comprehensiveness in global health IT," Health Systems, pp. 1–13, 2026.
20. R. K. Shaw and A. Kulagin, "Zero-Trust Network Access on IBM z/OS: Applying BeyondCorpPrinciples to Mainframe Perimeter Architecture," International Journal of Computer Information Systems and Industrial Management Applications, vol. 18, no. 6s, pp. 342–351, 2026.
21. H. Gaggar, "Architecting scalable enterprise AI agent infrastructure for distributed intelligent systems," International Journal of Advances in Signal and Image Sciences, pp. 193–208, 2025. doi: 10.29284/da9w0117.
22. R. K. Shaw, "Privileged Access Management for z/OS System Programmers: A Zero-Trust Control Framework for RACF, TSO, and SDSF Administrative Sessions," Journal of Intelligent Decision Making and Information Science, vol. 3, no. 4s, pp. 1559–1582, 2026.
23. R. K. Shaw, "SNMPv2-to-SNMPv3 Migration in Financial Mainframes: Risk Reduction and Repeatable Playbook," Journal of Engineering Research and Sciences, vol. 5, no. 6, pp. 1–14, 2026
24. H. Gaggar, "MemGov: Policy-Governed Memory Lifecycle Management for Enterprise LLM Agents," Journal of Intelligent Decision Making and Information Science, vol. 3, no. 11s, pp. 2199–2214, 2026.