APRIL 2026

Delegation as *Structure*

Why agentic institutions must design delegation, not audit it.

Kendall Ferrell

ABSTRACT

The central problem of agentic AI is not that institutions lack accountability frameworks. It is that they are applying those frameworks in the wrong place and at the wrong time. Delegation — the transfer of authority from a human or institution to a system — is not a moment. It is a structure, and like any structure it has load-bearing components that must be deliberately designed. Authorization is one. Scope boundaries, suspension conditions, escalation paths, inheritance rules, and termination conditions are others. Institutions that treat delegation as a simple handoff and rely on post-hoc audit to reconstruct accountability have not solved the governance problem. They have deferred it until the cost of failure is highest. This paper argues that delegation architecture must be a first-class design artifact of any agentic workflow, specifies the structural components required, and defines operational requirements specific enough for a cross-functional team to build from.

1 The Real Problem

For several years the dominant public frame for artificial intelligence centered on capability. Systems generated text, summarized documents, answered questions, and assisted human users while remaining visibly downstream of human action. That phase produced familiar debates about quality, labor substitution, misinformation, and bias. Those debates still matter. They no longer describe the

structure of the current moment. The emerging phase is organized around action. Systems route tasks, trigger workflows, manage exceptions, interact across interfaces, and make decisions with operational consequences. Gartner has projected that 40% of enterprise applications will feature task-specific AI agents by the end of 2026, up from less than 5% in 2025. Nearly half of surveyed organizations already report having moved into agentic AI deployment.

The institutional response to this shift has been to treat it primarily as a governance question: who is accountable, how do we audit, what policies apply. That framing is not wrong, but it is downstream of the real problem. The real problem is that institutions are treating delegation as a capability feature when it is a structural engineering problem. Deploying an agentic system without explicit delegation architecture is not a governance gap. It is a design choice and a specific one. It chooses implicit maximum permissiveness at every unspecified boundary. The system acts until something breaks. Audit then tries to reconstruct a chain that was never built. That is not accountability. It is archaeology.

This paper argues that delegation must be designed, not audited. It specifies what that means structurally, why each component is necessary, and what operational requirements follow for institutions building agentic systems today.

2 What Delegation Actually Is

The word *delegation* is used loosely in most institutional AI discussion. It is treated as synonymous with deployment, or automation, or capability access. That looseness is part of the problem. Delegation is the transfer of authority to act on behalf of a principal. It is not the transfer of capability. A system may have the capability to send an email, approve a purchase order, or escalate a support ticket. Capability describes what it can do. Delegation describes under what authority it does it, within what bounds, and with what accountability attached.

This distinction matters enormously in practice. Most current agentic deployments authorize capability. They say: this system may act in this domain. They do not authorize delegation. They do not say: at this specific point in this specific workflow, authority to take this class of action is being transferred from this named person or body to this system, within these bounds, subject to these

conditions, with this named person remaining answerable if those bounds are exceeded. Capability authorization and delegation authorization are categorically different acts. Conflating them is the foundational institutional error of the current agentic moment.

Delegation is also not an event. It is a structure. And like any structure, it has components. Those components exist whether or not they are designed. Undesigned, they default to the worst case at every boundary: maximum scope, no suspension, no escalation, no inheritance limits, no defined termination. The institution that deploys without designing these components has not avoided them. It has chosen their worst-case values implicitly.

3 The Components of Delegation Structure

Delegation structure has six load-bearing components. Each must be explicitly designed. Each has a worst-case default if left implicit. The following defines each component, its function, and what its absence produces.

3.1 Authorization

Authorization is the entry point into the delegation structure. It is the formal act by which a named human or institutional body transfers authority to a system to act within a defined scope. Authorization is not approval of a system's deployment. It is approval of a specific transfer of authority at a specific point in a specific workflow.

Authorization must be synchronous with delegation — created at the moment authority transfers, not reconstructed afterward. A deployment decision made six months prior is not authorization for a specific action today. If the chain cannot be traced from the current action back to a contemporaneous authorization event, the delegation is unauthorized regardless of whether the system has technical access to perform it.

Absence produces: actions taken under assumed authority. When something goes wrong, no authorization event exists to audit against. Reconstruction becomes guesswork.

3.2 Scope Boundaries

Scope boundaries define what the delegated system may and may not do within the authorized domain. Scope has two dimensions: action scope, which defines the classes of action the system may take, and consequence scope, which defines the maximum impact a single action may have

without human review.

Scope must be explicit and enumerated, not inferred from domain. "Authorized to handle procurement" is not a scope boundary. "Authorized to approve purchase orders below $10,000 with three or more competing quotes, excluding sole-source contracts and framework agreements" is a scope boundary. The difference is not pedantry. It is the difference between a boundary a system can respect and one it cannot even detect it is crossing.

Absence produces: scope creep by default. The system acts at the edge of its capability rather than the edge of its authorization. Operators cannot tell whether any given action was in scope without case-by-case review.

3.3 Suspension Conditions

Suspension conditions define the circumstances under which delegation pauses and authority returns to a human. They are the designed interrupt mechanism of the delegation structure. Without them, the system has no structural path to recognize that it has reached a boundary requiring human judgment.

Suspension conditions must be stateable in terms the system can evaluate at runtime. "Unusual circumstances" is not a suspension condition. "Action cost exceeds authorized consequence scope," "confidence below defined threshold," "action class not covered by current authorization," and "third sequential exception in current workflow instance" are suspension conditions. Each can be evaluated. Each creates a defined pause point where authority explicitly returns.

Absence produces: systems that continue acting through ambiguity because they have no structural mechanism to stop. Humans discover boundary violations after consequences have accumulated.

3.4 Escalation Paths

Escalation paths define how ambiguity, exception, and suspension surface back up the authorization chain. They are the communication architecture of the delegation structure. A system that can suspend but cannot escalate has paused without resolution. An escalation path specifies who receives the escalation, in what form, within what timeframe, and what happens if that person is unavailable.

Escalation paths must be tested under load. An escalation path that routes to a single named individual with no backup and no timeout is not a functional escalation path. It is a single point of failure that will produce either indefinite suspension or pressure to bypass the pause mechanism entirely.

Absence produces: suspended actions that accumulate without resolution, creating pressure on operators to override suspension conditions informally which destroys the integrity of the delegation structure entirely.

3.5 Inheritance Rules

Inheritance rules define whether and how a delegated system may sub-delegate authority further down to another system, agent, or workflow component and under what constraints. As agentic systems increasingly orchestrate other agentic systems, inheritance is no longer a theoretical concern. It is a live architectural question in most multi-agent deployments.

The default assumption must be that delegation does not inherit unless explicitly designed to do so. A system authorized to manage a procurement workflow is not thereby authorized to delegate approval authority to a sub-agent. Each delegation boundary must carry its own authorization event, scope boundaries, suspension conditions, and escalation path. The parent delegation does not transfer these automatically.

Absence produces: authority laundering. Each layer of sub-delegation dilutes the original authorization scope while the institution believes the original authorization still governs. By the time action occurs, no one can trace what was actually authorized.

3.6 Termination Conditions

Termination conditions define when delegation ends. Not when a workflow completes — when the authority itself expires. Delegation is not permanent by default. It should be time-bounded, scope-bounded, or event-bounded, with explicit re-authorization required to continue.

Most current agentic deployments have no termination conditions. Systems remain authorized indefinitely unless someone explicitly revokes access. This means delegation accumulates. Systems carry authority granted for one context into entirely different contexts months later. Re-authorization disciplines are as important as initial authorization disciplines.

Absence produces: authority accumulation. Systems carry broad, stale authorization into new contexts. The institution believes it is authorizing specific actions; the system is operating under permissions granted for purposes that no longer apply.

4 Why Audit Cannot Substitute for Design

The institutional instinct when governance lags is to audit more carefully. That instinct is not wrong for many problems. For delegation structure it is structurally insufficient, for three reasons.

First, audit is retrospective. It examines what happened after consequences have occurred. Delegation structure is designed to prevent certain failure modes from occurring at all — specifically, the failure mode of action taken outside the bounds of what was actually authorized. An audit can identify that a boundary was crossed. It cannot reconstruct what the boundary was supposed to be if no boundary was ever designed.

Second, machine-speed delegation makes retrospective reconstruction increasingly unreliable. A human process that handles hundreds of decisions per day can be audited meaningfully because the decision record is sparse enough to examine case by case. An agentic system handling thousands of actions per hour produces an audit surface that is practically unexaminable at the action level. Audit then necessarily operates at the aggregate or statistical level which means individual boundary violations are invisible unless they produce catastrophic outcomes. By the time an audit occurs, the authorization chain is already a reconstruction, not a record. Reconstruction under consequence pressure is not accountability. It is narrative.

Third, audit assumes there is a structure to audit against. If delegation was never designed — if there are no authorization events, no scope boundaries, no suspension conditions — then audit has no reference architecture. It can describe what happened. It cannot say whether what happened was within the bounds of what was authorized, because those bounds were never specified. The audit produces a chronicle, not an accountability finding.

This does not mean audit is useless. Audit against a well-designed delegation structure is genuinely valuable. It can surface boundary violations, identify scope drift, flag authorization gaps, and drive re-authorization decisions. The point is that audit is only meaningful when there is a designed structure to audit against. Without that structure, audit is an expensive way of producing uncertainty rather than accountability.

5 The Three Institutional Failures

Most institutions currently exhibit one or more of three specific failures in delegation architecture. Understanding which failure applies determines where to intervene.

5.1 Capability Authorization Without Delegation Authorization

The most common failure. The institution approves a system for deployment in a domain and treats that approval as sufficient authorization for any action the system takes within that domain. No authorization events are designed into the workflow. No scope boundaries are enumerated. The deployment approval is doing the work that only action-level delegation authorization can do.

This failure is most common in early-stage agentic adoption where the institution is focused on getting the system to work and treats governance as a second-phase concern. The problem is that by the time the second phase arrives, the system is in production, workflows are built around it, and retrofitting delegation structure requires re-engineering processes that teams have already optimized around the absence of that structure.

5.2 Delegation Design Without Inheritance Rules

A more sophisticated failure, common in institutions that have invested in governance. The institution has designed authorization events, scope boundaries, and escalation paths for its primary agentic systems. But as those systems begin orchestrating sub-agents — which happens quickly as agentic architectures mature — the delegation structure was never designed to propagate. Each sub-agent operates under implicit maximum permissiveness because no one designed inheritance rules for the multi-agent layer.

This failure is dangerous precisely because it is invisible to institutions that believe they have solved the delegation problem. Their primary systems have proper structure. The sub-agent layer does not, and that layer is where an increasing proportion of consequential action occurs.

5.3 Delegation Design Without Termination Conditions

The third failure is the most gradual and therefore the hardest to detect. The institution has built reasonable delegation structure for its initial deployment context. But delegation authority accumulates as systems persist, workflows expand, and re-authorization disciplines are not maintained. Systems carry authority granted for a specific context into entirely different contexts.

The institution believes each deployment is governed by current authorization. The systems are operating under permissions that have effectively become permanent. Authority accumulation is an institutional risk that grows over time. The longer agentic systems are in production without termination and re-authorization disciplines, the greater the gap between what institutions believe they have authorized and what systems are actually authorized to do.

6 Operational Requirements

The following requirements are specific enough for a cross-functional team — engineering, legal, compliance, and operations — to build from. They are stated as design requirements, not governance aspirations.

R1 · Authorization events must be synchronous and workflow-embedded.

- Every point in a workflow where authority transfers to a system must contain an explicit authorization event.
- The authorization event must record: the authorizing human or body, the scope of authority being transferred, the timestamp of transfer, and the workflow context in which transfer occurs.
- Authorization events must be created at transfer time, not reconstructed after the fact.
- A deployment approval is not an authorization event.
- Authorization events are action-class specific, not system-level.

R2 · Scope boundaries must be enumerable and runtime-evaluable.

- Action scope must enumerate the specific classes of action the system may take, not describe a domain.
- Consequence scope must specify the maximum impact of a single action (financial threshold, data volume, number of affected parties, reversibility classification) before human review is required.
- Every scope boundary must be stateable in terms the system can evaluate at the moment of action — not terms that require post-hoc interpretation.
- Actions at the edge of scope must trigger confirmation, not proceed silently.

R3 · Suspension conditions must be explicit, runtime-evaluable, and tested.

- Suspension conditions must be defined for each authorization scope and stateable in terms the system can evaluate without human interpretation.
- Minimum required suspension triggers: action class not covered by current authorization; consequence scope exceeded; defined confidence threshold not met; sequential exception count exceeded within a workflow instance.
- Suspension must route to a defined escalation path immediately.
- Suspension without escalation routing is not a functional interrupt.
- Escalation paths must include a named primary, a named backup, a maximum response time, and a defined fallback action if response time is exceeded.
- Suspension conditions and escalation paths must be tested under realistic load conditions before production deployment.

R4 · Inheritance must be denied by default and explicitly granted.

- No delegated system may sub-delegate authority unless inheritance is explicitly designed into the parent delegation.
- Each sub-delegation must carry its own authorization event, scope boundaries, suspension conditions, and escalation path.
- These do not inherit from the parent delegation automatically.
- Multi-agent architectures must map all delegation boundaries before deployment.
- Every boundary where authority transfers between agents must be treated as a full delegation event.
- Inheritance grants must be narrower than the parent delegation scope.
- A sub-agent may not be granted authority exceeding the scope of the delegating system.

R5 · Delegation must have defined termination conditions and re-authorization disciplines.

- All delegation authority must be time-bounded, scope-bounded, or event-bounded.
- Open-ended authorization is not permitted.
- Re-authorization must be a scheduled, active decision — not a default continuation.
- The burden must be on the institution to reauthorize, not on someone to notice that reauthorization is needed.
- Systems must expose current authorization state visibly to operators: what authority is currently active, when it expires, what scope it covers, and what actions have been taken under it.
- Expired or terminated authorization must cause the system to suspend, not to fail silently or continue under stale authority.

R6 · Authorization chain must be traversable at any point during system operation.

- Operators must be able to trace, at any moment during system operation, the complete authorization chain for any action the system is currently authorized to take.
- The chain must show: who authorized, what was authorized, when authorization was granted, what scope was defined, what conditions apply, and what actions have been taken under this authorization.
- Traversability is a design requirement, not a logging requirement.
- A log that requires reconstruction to produce the chain is not sufficient.
- The chain must be navigable in real time.
- Authorization chain visibility must be accessible to engineering, compliance, legal, and operations without requiring technical extraction from system logs.

---

7 What Institutions Must Do Differently

The requirements above are not a compliance checklist to apply after agentic systems are built. They are architectural constraints that must shape how agentic systems are built. That distinction determines where institutions must intervene.

7.1 Treat delegation design as a pre-deployment gate

No agentic system should enter production without a completed delegation structure covering all six components: authorization, scope boundaries, suspension conditions, escalation paths, inheritance rules, and termination conditions. This is not a governance review that happens after engineering is complete. It is a design requirement that shapes the engineering from the start.

In practice this means that the team responsible for agentic workflow design must include delegation architecture as an explicit deliverable alongside functional requirements. The question *"what is this system authorized to do, under what conditions, and what happens at every boundary"* must be answered before build, not after launch.

7.2 Separate capability decisions from delegation decisions

Institutional decision-making about agentic systems currently conflates two decisions that must be separated. The first is a capability decision: should this system have access to this domain, this data, this interface? The second is a delegation decision: under what specific authorization, within what specific scope, subject to what specific conditions, with what specific accountability attached?

Capability decisions are appropriately made at the system or deployment level. Delegation decisions must be made at the workflow and action-class level. An institution that makes only capability decisions and treats them as sufficient has not made delegation decisions. It has avoided them.

7.3 Build re-authorization as a standing discipline

Authorization decay is as serious a risk as authorization absence. Institutions must build re-authorization as a standing operational discipline, not a one-time deployment activity. This means scheduled re-authorization reviews tied to delegation termination conditions, ownership of re-authorization assigned to named roles, and systems that surface expiring authorization proactively rather than waiting for someone to notice.

Re-authorization reviews should not be perfunctory. They are the mechanism by which institutions confirm that current delegations still reflect current intent, current risk tolerance, and current operational context. As agentic systems expand into new workflows and new contexts, re-authorization is the primary check on authority accumulation.

8 Conclusion

The governance debate about agentic AI has been framed primarily as a question of accountability: who is responsible when systems act in ways that cause harm. That framing is not wrong, but it is downstream of the real problem. Accountability is only meaningful when it can be traced. Tracing requires a chain. A chain requires design.

Institutions that deploy agentic systems without designing delegation structure are not deferring accountability. They are making it structurally unachievable — because when something goes wrong, there is no authorization record to audit against, no scope boundary to determine whether the action was in bounds, no escalation record to show whether the right people were notified, no termination condition to establish whether the authority was still valid.

The question is not whether institutions will be held accountable for agentic action. They will. The question is whether they will have built the structures that make accountability meaningful rather than merely formal.

Delegation is not a moment. It is a structure. It has six load-bearing components, each of which must be explicitly designed. Authorization is the entry point, but scope boundaries determine what the system may actually do, suspension conditions determine when it must stop, escalation paths determine how ambiguity surfaces, inheritance rules determine what happens in multi-agent architectures, and termination conditions determine when authority expires.

Institutions that design all six components before deployment, treat re-authorization as a standing discipline, and build delegation chain traversability into their operational architecture are not just managing risk better. They are building the only foundation on which accountable agentic action is possible.

Institutions that do not are not neutral. They have chosen the worst-case value at every undesigned boundary. That is a design decision. It should be recognized and treated as one.

References

[1] Informatica. "AI Adoption Trends 2026: Trust, Data Quality & Governance Challenges." *informatica.com*, 2026. informatica.com/blogs/cdo-insights-2026-ai-adoption-accelerates-but-trust-and-governance-lag-behind.html

[2] Gartner / Lowtouch AI. "AI Reality Check: 2025 Adoption Today vs. 2026 Transformations Ahead." *lowtouch.ai*, 2025. lowtouch.ai/ai-adoption-2025-vs-2026/

[3] Deloitte. "AI trends: Adoption barriers and updated predictions." *deloitte.com*, 2025. deloitte.com/us/en/services/consulting/blogs/ai-adoption-challenges-ai-trends.html

[4] Council on Foreign Relations. "How 2026 Could Decide the Future of Artificial Intelligence." *cfr.org*, 2026. cfr.org/article/how-2026-could-decide-future-artificial-intelligence

[5] Joget. "AI Agent Adoption 2026: What the Data Shows." *joget.com*, 2026. joget.com/ai-agent-adoption-in-2026-what-the-analysts-data-shows/

[6] ALM Corp. "AI Adoption for Digital Agencies: Best Practices That Deliver Results (2026)." *almcorp.com*, 2026. almcorp.com/blog/ai-adoption-digital-agencies-best-practices/

Cite as: Ferrell, K. (2026). *Delegation as Structure: Why Agentic Institutions Must Design Delegation, Not Audit It.* DelegationOS Doctrine Paper v1.0. delegationos.dev/doctrine/delegation-as-structure

Canonical URL: https://delegationos.dev/doctrine/delegation-as-structure
Contact: founders@delegationos.dev
Formats: HTML and PDF both published at the canonical URL; PDF also distributed via SSRN.

SSRN ID and DOI will be added to v1.0.1 once assigned post-publication.