Nick Clark · Published April 25, 2026 · PDF

How to Get End-to-End Audit Across a Multi-Vendor AI System

If your AI pipeline crosses vendor boundaries (one company's model, another's retrieval index, a third's transport layer, your own orchestration), the audit trail almost always breaks at each seam. This guide describes an architecture for making identity, policy, and attribution consistent from the first hop to the last, so the complete behavioral history is reconstructible from a single record. It describes an architecture disclosed in United States Patent Application 19/647,395, not a shipping library, and it centers on what that filing calls the Cross-Patent Architecture inventive step.

What You Are Building

You are building a system where you can answer, for any output, one question with confidence: what happened, in order, across every component that touched the work, no matter who supplied that component.

Most real AI systems are multi-vendor by the time they reach production. A request might hit a hosted foundation model, a separately owned vector or retrieval index, a message or transport layer between services, an identity provider, and a policy or governance layer you bolted on yourself. Each of those tiers logs something. None of them logs the same thing, keyed the same way, with the same notion of "who" and

"under what rule." When a regulator, a customer, or your own incident review asks "why did the system do that, and was it allowed to," you are left stitching together mismatched logs by timestamp and hope.

The goal here is an architecture where identity, policy, and attribution are not re-derived at each tier but travel with the unit of work, and where the full trajectory is reconstructible from one record rather than from the union of everyone's partial logs. This guide is for engineers designing that governance layer.

Why the Obvious Approaches Fall Short

The common approaches are reasonable and each solves a real piece. They just do not compose into end-to-end audit.

Centralized log aggregation (shipping every service's logs into one store) gives you a searchable pile, but the records were emitted by independent systems with independent schemas and independent identifiers. Correlating them is a reconstruction exercise, and the correlation is only as good as your best guess about which event caused which. Nothing guarantees a given tier logged the fields you now wish it had.

Distributed tracing (propagating a trace or correlation ID across service boundaries) is a genuine improvement and worth doing. It tells you that spans belong to the same request. But a trace ID is a correlation key, not a governance record: it does not carry the policy that was in force, the identity that authorized the step, or the attribution obligations attached to the content being handled. Vendors on the path are free to honor the header and log nothing else you care about.

Per-vendor compliance attestations push the burden onto each supplier to certify their own tier. That leaves the seams between tiers unaudited, which is exactly where authority, policy freshness, and attribution tend to get lost.

The structural gap is the same in every case: audit is treated as something each tier produces about itself, then reconciled afterward. Reconciliation is lossy, and it degrades further every time a vendor changes. What you want instead is for the audited state to be a property of the work itself, carried across tiers, so that no tier gets to decide unilaterally what was recorded.

The Architecture

The approach disclosed in the referenced patent application inverts the relationship. Instead of each tier owning a slice of the record, a single governed object carries its own state, its own governing policy, and its own history across every tier. Below are the mechanisms as disclosed. This is an architecture; you implement it.

A schema-conformant object as the unit of work. The disclosure defines a semantic agent as a persistent object built on a foundational schema of typed fields: intent, context, memory, a policy reference, a mutation descriptor, and a lineage field. Governance, memory, lineage, and execution eligibility are intrinsic fields of the object rather than external bookkeeping. Because the object is the thing that moves through the system, everything attached to it moves too.

Cognitive-domain fields that travel with the object. On top of the foundational schema, the disclosure adds independently tracked fields (for example affective state, integrity, confidence, and a capability field). What matters for audit is the transport property: the disclosure describes extending a state-preserving transport layer so that when the object migrates between execution substrates, these fields travel with it and are carried intact rather than reconstructed at the destination. The destination validates the arriving object's lineage continuity before proceeding. State is a property of the object, not of any host.

A policy reference embedded in the object. The object carries a policy reference field, and the disclosure describes a cryptographic policy framework that provides signed policy constraints applicable to the object's fields. Because the governing constraints ride with the object, the rule in force at a given step is part of the audited record, not a separate configuration you have to reconcile after the fact. The disclosure also addresses policy freshness across asynchronous gaps: when an object resumes after an interval and detects that the policy in force at suspension has been superseded, that staleness is itself treated as a signal that can hold execution pending the current policy.

A single lineage chain as the audit spine. This is the load-bearing mechanism. The disclosure records each proposed mutation, each admissibility determination, and each field update into the one lineage field, with the explicit property that the complete behavioral trajectory of the object is deterministically reconstructible from the lineage field alone. Crucially, the extension fields write their updates into that same lineage chain rather than into a side log, so cognitive-domain changes are recorded in the same chain as all other state transitions. One chain, one order, one source of truth for "what happened."

Continuity-based identity with a common interface. Rather than static credential checks at each boundary, the disclosure establishes identity through trust-slope continuity: a temporally ordered sequence of non-invertible hashes, each evaluated for continuity with its predecessors and each carrying a graded confidence value rather than a binary match. It describes three identity substrates (agent, device, and human biological identity) that are structurally independent but interoperable through a common trust-slope interface, and that can be compositionally bound by policy (for example, requiring a biological identity above a confidence threshold, presenting through an attested device, interacting with a continuously validated agent). Notably, the cross-substrate interface operates on confidence and continuity values that are not invertible back to raw identity data, so identity can factor into audit across tiers without every tier sharing underlying identity material.

Attribution carried as governed state. The disclosure describes a rights-governance mechanism in which required creator attribution and content-exclusion obligations are policy-specified, evaluated as content is handled, and recorded, so that a step can be admitted with an attribution annotation or rejected per policy. Attribution obligations thus become part of the same governed trajectory rather than a separate, easily dropped concern.

The cross-application property tying this together (the home inventive step) is that these mechanisms are disclosed as co-pending applications operating simultaneously as one governed substrate. The disclosure gives concrete examples of the seams behaving as one system: the index becomes an execution substrate on which a traversing object records each step in its own lineage; identity revocation on a host mid-execution reclassifies that host and preserves the object's state because the fields are carried by the object, not the host. Same identity model, same policy framework, same lineage chain, across every tier.

How to Approach the Build

You are implementing this yourself. A workable order:

1. **Define the governed object schema first.** Before any tier, specify the typed fields the object carries: intent and context, memory, a policy reference, a mutation record, and a lineage field. Treat the lineage field as append-only and ordered. Everything downstream depends on this being the real unit of work, not a wrapper you strip at the first boundary.
2. **Make lineage the only place state changes are recorded.** The disclosed guarantee (reconstructible from lineage alone) only holds if every mutation writes there. Resist the temptation to let a tier keep its own private record. An illustrative interface sketch, faithful to the disclosed structure and not a shipping API:

```
record_transition(object, {
  mutation,            // what was proposed
  determination,       // permit | gate | suspend
  field_updates,       // every field that changed
  policy_ref,          // the signed policy in force
  identity_continuity  // trust-slope confidence, not raw identity
})  // appended to the single lineage chain, in order
```

3. **Attach policy to the object, not to the environment.** Give the object a policy reference and adopt a signed-policy scheme so the constraints in force are verifiable and are themselves part of the record. Decide how you detect superseded policy on resume, and treat staleness as a first-class condition that can gate execution.
4. **Adopt a continuity-based identity interface across tiers.** Standardize on a common representation (ordered non-invertible hashes with graded confidence) that every tier can produce and consume, rather than per-vendor credential formats. Keep it non-invertible so tiers share confidence, not secrets. Let policy decide when substrates must be compositionally bound.
5. **Preserve state across transport, do not rebuild it.** At each hop, carry the object's fields intact and validate lineage continuity on arrival before the destination acts. A break in continuity is an audit event, not a silent retry.
6. **Fold attribution and exclusion into policy evaluation.** Where the work touches third-party or restricted content, evaluate attribution and exclusion obligations as part of the same governed step so they land in the same lineage chain.
7. **Wrap vendor tiers you cannot modify.** In practice some vendors will not adopt your object. Put a thin adapter at that boundary that carries the object through, records the vendor call as a lineage transition on the way in and out, and validates continuity across the gap. The seam is then audited even if the vendor's internals are not.

What This Does Not Give You

This is an architecture disclosed in a patent filing, not a drop-in library, an SDK, or a downloadable package. There is nothing here to `npm install`. You design and build it, and the effort is real: a shared object schema, an append-only lineage discipline, a signed-policy scheme, and a continuity-based identity interface are all substantial pieces of work.

It has not been benchmarked or productized here, and this guide makes no performance, scalability, or security guarantees. The pseudocode is illustrative of the disclosed structure, not a working implementation, and it will not "just work" when pasted in.

The approach also has boundaries. Its strongest form assumes tiers can carry your governed object; for vendors that cannot, you fall back to boundary adapters, and the audit is only as complete as those adapters make it. It presumes you control enough of the pipeline to impose the object and the single lineage discipline. And it is aimed at behavioral and governance auditability (what happened, under what policy, authorized by what continuity), not at proving properties of a third party's model internals, which remain outside anything the object can observe.

Disclosure Scope

The architecture described in this guide is disclosed in United States Patent Application 19/647,395. This guide is educational: it explains an architectural approach so that a skilled engineer can understand and build it themselves. It is not a warranty, a specification, a benchmark, or an offer of software, and it does not describe a shipping product. Where third-party technologies are mentioned for context, they are described neutrally and only to situate the approach. Every claim above about how the disclosed approach works is grounded in the referenced filing.

Cross-Patent Architecture (/cross-patent-architecture)

All 40 steps → (/inventive-steps)

Cross-cutting architectural principles that compose every primitive into a coherent platform.

Chapter 1 (/patents/19-647395/chapters/foundation)

PRIMARY TECHNICAL DISCLOSURE

- Cross-Patent Architecture, Articles (/articles/cross-patent-architecture)

SECONDARY TECHNICAL

- Transit Cognitive State (/articles/cross-patent-architecture/transit-cognitive-state)
- Substrate Identity Revocation During Active Cognition (/articles/cross-patent-architecture/substrate-identity-revocation)
- Policy Freshness Across Asynchronous Execution (/articles/cross-patent-architecture/policy-freshness-asynchronous-execution)
- Governance Authority Evaluation via Integrity Trajectory (/articles/cross-patent-architecture/governance-authority-integrity-trajectory)
- Discovery Agent as Schema-Conformant Index Traverser (/articles/cross-patent-architecture/discovery-agent-schema-index-traverser)
- Unified Substrate for Governed Information Acquisition (/articles/cross-patent-architecture/cross-tier-navigation-world-as-model)

APPLICATIONS · GENERAL

- One Governed Platform, Not Four Integrated Systems: A Unified Architecture Spine for Agent Execution, Cognition, Content, and Spatial Tiers (/articles/cross-patent-architecture/unified-governed-platform)
- World-as-Model Systems: Navigating the Physical World, Cognition, and Discovery as One Governed Model (/articles/cross-patent-architecture/world-as-model-systems)
- End-to-End Lineage and Audit: Reconstructing Any Agent Action Across Every Tier of the Stack (/articles/cross-patent-architecture/end-to-end-lineage-and-audit)
- Moving Governed AI Agents Across Clouds and Vendors Without Losing Identity: Substrate Portability via the Cross-Patent Architecture (/articles/cross-patent-architecture/portability-across-substrates)
- Cross-Patent Architecture: Why a Coherent AI Platform Needs a Shared Governance Authority at the Foundation, Not as a Feature (/articles/cross-patent-architecture/ai-platform-foundation)

- Regulated Cross-Domain Deployment: One Governance Authority and Policy-Freshness Model Across Every Tier of an End-to-End System (/articles/cross-patent-architecture/regulated-cross-domain-deployment)

APPLICATIONS · SPECIFIC

- Palantir Foundry and AIP (the ontology-based data/operations platform plus its AI orchestration layer) vs a cross-tier governed architecture: where does end-to-end action attribution live? (/articles/cross-patent-architecture/palantir-foundry-aip)
- Microsoft's integrated AI stack (Azure AI Foundry, Microsoft Fabric, Entra, and Copilot) vs a single cross-domain governance architecture: how do coherence and one governance chain differ from an integrated product suite? (/articles/cross-patent-architecture/microsoft-ai-stack)
- Amazon Web Services' integrated AI/data stack (Bedrock, SageMaker, and surrounding data/identity services) vs a unified cross-tier governed agent architecture (/articles/cross-patent-architecture/aws-ai-stack)
- NVIDIA's full-stack AI platform (NVIDIA AI Enterprise, NIM microservices, and the CUDA/hardware-to-software stack) vs a substrate-independent governance architecture (/articles/cross-patent-architecture/nvidia-ai-enterprise)
- Databricks Data Intelligence Platform (lakehouse plus Mosaic AI, Unity Catalog governance, and agent tooling) vs an agent-resident cross-patent architecture: where governance lives (/articles/cross-patent-architecture/databricks-data-intelligence)

Cross-Patent Architecture overview → (/cross-patent-architecture)