WHITE PAPER | VERSION 1.4

Outcome Coordination for AI Agents

Governing execution within systems and commitments across trust boundaries

DOI: 10.5281/zenodo.21792756

AUTHOR
Hammad Abbasi

RESEARCH AREA
AI agent security, enterprise architecture, distributed coordination

PUBLICATION DATE
August 2026

REFERENCE PROJECTS
Lattice and Covenant Layer

Research and implementation. Outcome Coordination is presented as an architectural model for delegated agent work. Lattice provides a capability-runtime design for governed internal execution. Covenant provides a protocol, onboarding framework, registry contracts, conformance tooling, and an end-to-end demonstration for commitments across trust boundaries. Both repositories are open for implementation, review, and comparative testing.

Abstract

AI agents are often placed inside the execution loop of enterprise and consumer software. The model selects tools, reconstructs parameters, carries intermediate state through context, responds to failures, and eventually causes an authenticated side effect. Each operation can be valid while the combined effect is outside the task the user intended.

This paper presents Outcome Coordination as an architectural response. The proposal is to organize agent systems around explicit outcomes, bounded authority, decision points, evidence, and responsibility for fulfillment. It separates the work performed inside one organizational trust boundary from coordination that crosses between organizations.

Lattice addresses the internal case. It exposes typed, outcome-shaped capabilities to the model and runs their steps in ordinary executable code. Sequencing, state, credential injection, permissions, failure policy, human tasks, and audit remain in the runtime. The model receives a projection designed for explanation and the next valid decision.

Covenant addresses the cross-boundary case. A user request becomes an objective. Eligible providers return offers with concrete terms. Valid acceptance activates one commitment. The provider fulfills through systems it controls, evidence is checked, and settlement records the resulting state. Covenant standardizes the public coordination objects and lifecycle while leaving provider internals open.

The paper defines an outcome coordinate as a research abstraction, explains how Lattice and Covenant represent different parts of it, and describes how the two layers can work together. It also sets out an evaluation program for measuring execution quality, authority preservation, auditability, recovery, and legitimate task completion.

Contents

1. Introduction
2. Working vocabulary
3. The execution problem
4. Outcome Coordination
5. Lattice: execution inside a trust boundary
6. Covenant: commitments across trust boundaries
7. Combined reference architecture
8. Security model and assurance boundaries
9. Evaluation program
10. Adoption path
11. Open research questions
12. Conclusion

References

Appendix: object and control summary

1. Introduction

Users usually describe the result they want. Agent infrastructure often converts that request into a sequence of software operations and asks the model to manage the sequence.

A request to arrange travel, onboard a supplier, investigate an insurance claim, or procure equipment contains judgment. Some conditions are absolute. Others express preference. Approval may be required at one point and unnecessary at another. The information needed to decide can change while the task is running.

Most enterprise APIs expose methods shaped around applications and backend services. An agent receives those methods as tools, then has to infer the business process that normally surrounds them. It may choose the right endpoint and still use it at the wrong time. It may carry a valid credential and still produce an effect that the user did not authorize. The backend sees a well-formed request. The unresolved question sits above the request: did the resulting effect still belong to the task?

The original Outcome Coordination essays proposed a change in the public interface between agents and software. The agent should remain close to intent, policy, comparison, approval, and monitoring. Exact fulfillment should remain with the system or provider that owns the domain and can be held responsible for the result.[1][2]

Two reference projects explore that idea at different scopes. Lattice focuses on execution inside one system. Covenant focuses on coordination across systems and organizations.[4][5] They can be used together, though neither requires the other.

> **Thesis**
>
> An agent system should expose enough structure for the model to understand the requested outcome and make bounded decisions, while deterministic execution, credentials, recovery, and accountability remain with a governed runtime or an accountable provider.

Scope

This paper covers delegated work that can produce real-world effects. Examples include booking, procurement, vendor operations, claims processing, logistics, account changes, and enterprise service delivery. It does not argue that every tool call requires a market, a blockchain, or a formal settlement process. Small, reversible actions inside a controlled environment may be served well by direct tools or code execution.

The proposal becomes useful when the task includes expensive side effects, several systems, external providers, approval boundaries, sensitive data, or a need to establish who accepted which terms.

2. Working vocabulary

The terms below separate concepts that are often collapsed into a single agent workflow.

Term	Meaning in this paper
Outcome	A real-world or system effect described with enough conditions to decide whether it was satisfied.
Procedure	A sequence of operations used to produce an effect.
Capability	An outcome-shaped contract backed by executable steps inside a system.
Projection	A task-scoped view that gives the model decision-relevant state, consequences, and valid next actions without exposing the full backend payload.
Objective	A requested outcome with constraints, preferences, approval conditions, and a freshness window.
Authority grant	A bounded delegation describing which consequential acts an agent may perform, under whose authority, for how long, and under which approval rules.
Offer	A provider proposal to fulfill an objective under stated economic, timing, evidence, and acceptance terms.
Acceptance	An authenticated statement that activates a specific offer under the accepting principal's authority.
Commitment	The obligation created by valid acceptance.
Evidence	A signed or integrity-verifiable claim about fulfillment, failure, refund, or another lifecycle event. Evidence still requires evaluation.
Settlement	The recorded state of an accepted commitment, such as fulfilled, failed, refunded, disputed, or expired.

The outcome coordinate

An outcome coordinate is a research abstraction for the information that should remain stable enough to guide a run while still allowing controlled revision. It is not a claim that one object can fully capture human intent.

```
O = { G, H, P, A, C, E, T }
G = target effect
H = hard constraints
P = preferences and tradeoff policy
A = authority and approval conditions
C = commitment or execution state
E = evidence and completion conditions
T = freshness, expiry, and revision rules
```

The coordinate lets designers ask a question that permission checks alone cannot answer:

> Does the observed effect remain consistent with the requested outcome, the applicable constraints, the authority used, and the evidence required for completion?

That question is difficult. Semantic consistency between a broad objective and an observed effect cannot always be reduced to a deterministic rule. Outcome Coordination narrows the problem by placing more of the execution path inside reviewed code or provider commitments with explicit terms.

How the implementations represent the coordinate

Lattice does not currently store one monolithic outcome-coordinate object. Its representation is spread across the capability signature, projection schema, step graph, scopes, failure policies, runtime state, and audit record. Covenant distributes the same concerns across Objective, Authority Grant, Offer, Acceptance, Evidence Record, and Settlement Receipt objects.

Coordinate field	Lattice	Covenant
Target effect	Capability name, typed inputs, capability code	Objective target and accepted offer outcome
Hard constraints	Input validation, step conditions, runtime policy	Objective constraints and offer acceptance conditions
Preferences	Projection fields and model-side decision policy	Objective preferences and offer comparison
Authority	Runtime scopes, roles, credential store	Authority Grant, approval reference, accepting actor

Coordinate field	Lattice	Covenant
Execution or commitment state	Compiled plan, step state, run status	Offer, Acceptance, commitment lifecycle
Evidence and completion	Projection and audit record	Evidence Record, verification, Settlement Receipt
Freshness and revision	Capability version and run inputs	Expiry, timestamps, revocation, replay controls

3. The execution problem

3.1 The model becomes the workflow engine

In a direct-operation architecture, the model may choose a tool, read the response, extract an identifier, select another tool, construct new parameters, retry after an error, and decide when the task is complete. The conversation context becomes a temporary state store and transport channel.

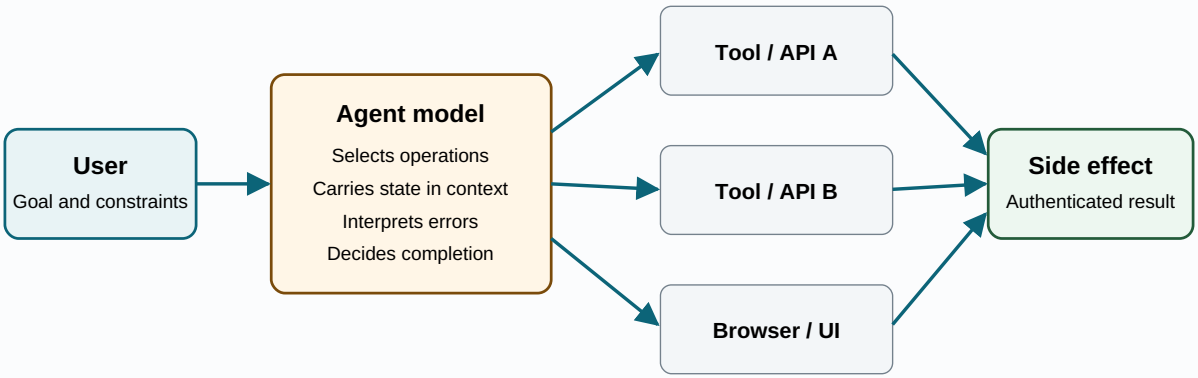

Figure 1. In a direct-operation design, the model participates in sequencing, state transfer, error recovery, and side-effect selection.

Long contexts do not ensure that the model will reliably retrieve the right detail at the right time. Research on long-context use found substantial position sensitivity, including degraded performance when relevant information appeared in the middle of the context.[9] The risk becomes operational when an old tool result, an injected instruction, or an inferred identifier is later treated as trusted state.

3.2 Individually permitted operations can form the wrong effect

Authentication answers who made the request. Authorization can answer whether that principal may call an endpoint. Schema validation can establish that the request is well formed. None of those checks independently proves that the combined effect matches the user's task.

Consider a procurement agent that may read inventory, inspect a budget, create a purchase order, and notify a supplier. Every call can be permitted. The result can still be wrong because the agent used the wrong budget period, placed the order before approval, or repeated a non-idempotent operation after an ambiguous timeout.

The distinction can be written as:

```
Operation validity = identity + permission + schema + current
resource state
```

```
Outcome legitimacy = operation validity + task consistency + correct
sequencing + applicable approval + sufficient evidence
```

Outcome legitimacy cannot be inferred from operation validity alone.

3.3 Tool surface and backend payloads compete with the task

Large tool sets introduce overlapping names, long schemas, and competing routes to the same effect. Microsoft Research describes cases where otherwise reasonable tools reduce end-to-end performance when they are present together, a phenomenon called tool-space interference.[10] The issue is architectural as well as statistical: the model has to reconstruct system boundaries that the application normally enforces.

Raw backend responses add internal identifiers, error codes, nested records, and sensitive data to the model context. Much of that material is useful to the runtime and unnecessary for the model's decision. The model needs a concise account of what happened, the policy consequence, and the next valid choices.

3.4 Direct operation still has a place

Outcome Coordination does not require formal commitments for every action. Direct operation remains reasonable when side effects are reversible, the environment is disposable, the tool set is small, credentials are narrow, and failures are cheap to detect. Coding inside a sandbox is a clear example. The boundary should tighten when work crosses into shared state, production systems, money movement, external commitments, or regulated records.

4. Outcome Coordination

Outcome Coordination is the design discipline of keeping the agent near the parts of work that require interpretation while moving exact execution into a governed runtime or an accountable provider.

The agent can clarify the request, distinguish hard conditions from preferences, compare valid options, apply policy, obtain approval, and monitor whether the expected result arrived. The execution owner handles credentials, sequencing, retries, internal state, compensating action, evidence production, and operational recovery.

The public security object depends on the trust boundary. Inside one organization, it can be a reviewed capability run tied to authenticated authority and a structured audit record. Across organizations, it becomes an accepted commitment with exact terms, evidence requirements, and a settlement lifecycle.

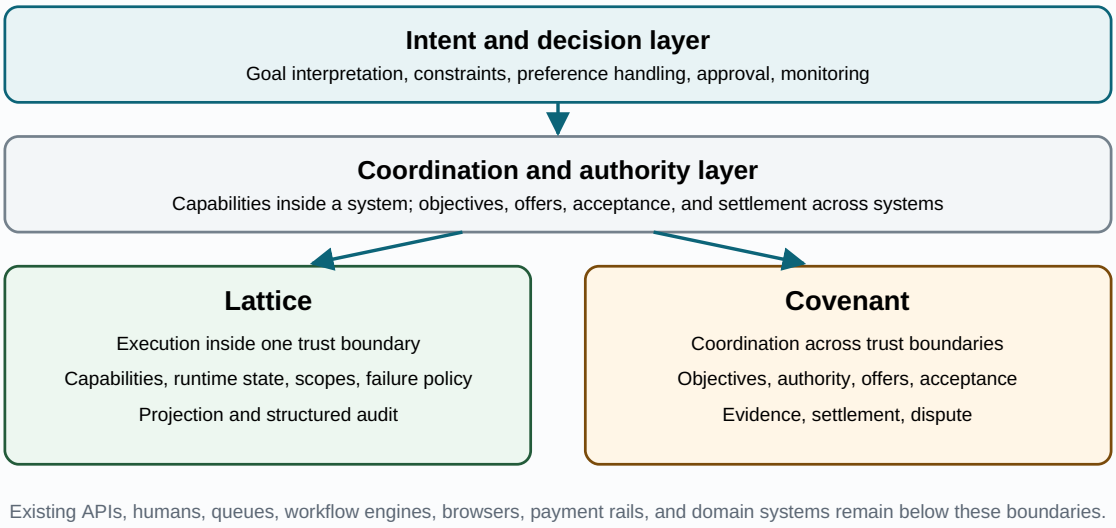

Figure 2. Outcome Coordination covers two scopes. Lattice governs execution inside a system. Covenant standardizes commitments between systems.

Design principles

- The model should propose an outcome or select among concrete terms. It should not be required to reproduce the full execution path.
- Authority should be bound to a principal, scope, exposure limit, approval rule, and validity period.
- Consequential state changes should occur at a named boundary that can be inspected and audited.
- Intermediate state and sensitive backend payloads should remain outside model context unless they are needed for a specific decision.
- The execution owner should declare failure behavior and recovery before production use.
- Completion should depend on evidence appropriate to the domain rather than a model's confidence that the task is finished.

What changes in the architecture

Tools do not disappear. The change concerns where they sit. Inside a provider or enterprise runtime, APIs and tools remain useful implementation mechanisms. At the agent-facing boundary, the interface becomes a capability or a commitment that describes the effect, conditions, and available decisions.

5. Lattice: execution inside a trust boundary

Lattice is a capability runtime for outcome-based execution inside one system. Its README defines the split directly: Lattice handles sequencing, state, permissions, failure policy, and audit, then returns a projection the model can reason over.[4]

5.1 Capability contracts

A capability is executable Python code with a model-facing contract. The contract declares a name, version, typed inputs, and projection schema. Nested steps declare dependencies, scopes, retries, and failure behavior. The runtime compiles and executes the graph.

```
@capability(
    name="VendorOnboarding",
    version="1.0",
    inputs={"vendor_name": str, "vendor_type": str, "region": str},
    projection={
        "vendor_id": str,
        "status": str,
        "compliance": str,
        "risk_score": int,
        "documents_pending": list,
    },
)
async def vendor_onboarding(ctx):
    @step(depends_on=[], scope="compliance.read")
    @retry(max=3, backoff="exponential")
    async def sanctions_check(): ...

    @step(depends_on=[sanctions_check], scope="vendor.write")
    async def create_vendor_record(): ...

    return projection(...)
```

The body remains ordinary code that can be reviewed, tested, debugged, and integrated with existing clients. Code generation can help create a starting point from OpenAPI or other metadata, but human review remains part of the deployment process.

5.2 Progressive discovery

Lattice exposes two meta-tools to the model: `search_capabilities` and `execute_capability`. The registry holds metadata and lazy-loads the selected module. This keeps the model-facing surface stable as internal capabilities grow.[4]

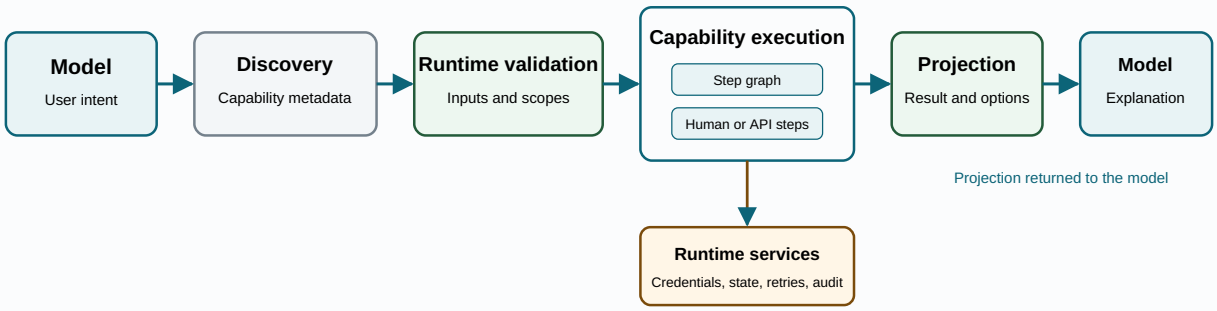

Figure 3. Lattice keeps execution state, credentials, recovery policy, and audit in the runtime. The model receives a task-scoped projection for explanation and the next decision.

5.3 Projections as decision surfaces

A projection is useful when the model can explain the result, present valid alternatives, and act on the user's choice without decoding backend-specific fields. A blocked travel request, for example, should identify the policy condition and available remedies rather than return a generic HTTP status and internal policy code.

```
{
  "status": "blocked",
  "reason": "policy_violation",
  "violations": [
    {"policy": "Travel limit", "threshold": 1500, "current": 1850}
  ],
  "options": [
    {"action": "request_approval", "approver": "Finance"},
    {"action": "reduce_cost", "target_total": 1500}
  ]
}
```

The projection reduces data exposure and cognitive load, though it creates a design obligation. A poorly designed projection can omit a condition the model needs or present a misleading summary. Projection review should therefore be treated as part of application and security design.

5.4 Permissions, credentials, failure policy, and audit

Each step can declare a scope. Lattice validates required scopes before execution begins and injects credentials at runtime. The model does not need to handle access tokens. Steps can specify retry behavior, soft-failure fallbacks, or hard failure that aborts downstream work. Every run emits structured records with execution status and per-step outcomes.[4]

Human tasks can participate in the same graph. A company can begin with a capability whose steps are performed by teams, then replace selected steps with APIs while keeping the model-facing signature stable.

5.5 What Lattice establishes

- The model requests a named, typed capability rather than improvising a long tool sequence.
- The runtime owns workflow state, dependency order, credential injection, and declared failure behavior.
- The model receives a bounded projection rather than every intermediate payload.
- Scopes can be checked before steps cause side effects.
- Execution produces a structured audit record.

5.6 Lattice's assurance boundary

Lattice concentrates execution control in typed capabilities, reviewed step graphs, preflight scope checks, runtime-managed credentials, and structured records. Surrounding application controls remain responsible for interpreting the user's request, approving capability design, setting appropriate scopes, and deciding which projection fields are required for a sound decision. The repository currently demonstrates this model through procurement and compliance workflows.[4]

6. Covenant: commitments across trust boundaries

Covenant is an open protocol and framework for outcome coordination between users, agents, brokers, providers, verifiers, and settlement services. It standardizes the public coordination model and leaves provider internals out of scope.[5][6]

6.1 Roles

Role	Responsibility
User or principal	States the objective, supplies authority, and approves consequential terms when required.
Agent	Interprets the request, applies policy, compares offers, obtains approval, accepts within authority, and monitors resolution.
Broker	Routes objectives to eligible providers, aggregates offers, and preserves provenance.
Provider	Makes an offer, fulfills accepted terms through systems it owns, and submits evidence.
Verifier	Checks evidence format, timing, and consistency with accepted terms and domain rules.
Settlement service	Records lifecycle state and links it to evidence and disputes.

One organization may perform several roles in an early deployment. The role definitions are still useful because they identify where authority, economic interest, verification, and record keeping are concentrated.

6.2 Protocol objects

The protocol defines Objective, Authority Grant, Offer, Acceptance, Evidence Record, and Settlement Receipt. The Objective carries the requested outcome, constraints, preferences, approval conditions, and freshness window. The Authority Grant bounds what the agent may do. The Offer states what a provider is willing to deliver and under which terms. Acceptance activates a specific offer. Evidence supports verification and settlement.[7]

6.3 Acceptance as the consequential boundary

An offer remains informational until an authorized actor accepts it. Valid acceptance creates the commitment and assigns fulfillment responsibility to the selected provider. This boundary prevents ranking, viewing, or tentative selection from silently becoming a purchase or other external effect.

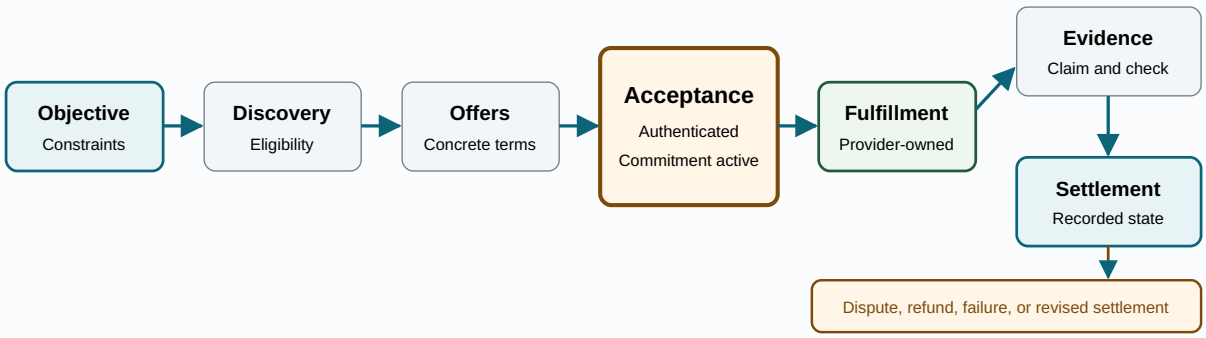

Figure 4. Covenant separates exploratory coordination from the acceptance event that activates provider responsibility.

6.4 Integrity, replay, and lifecycle state

Offer, Acceptance, Evidence Record, and Settlement Receipt objects are intended to be integrity-verifiable. The specification calls for identifiers, timestamps, validity windows, deterministic hashes, signatures, replay detection, and state-transition checks. Minimum settlement states include fulfilled, failed, refunded, disputed, and expired.[7]

6.5 Onboarding and shared registry state

The reference framework uses signed participant manifests, versioned conformance profiles, onboarding states, stake, and an EVM L2 registry. Participants can move through requested, identity-verified, conformance-passed, probation, active, restricted, and revoked states. Operational data such as manifests, evidence payloads, policies, and analytics remains off-chain; the chain anchors participant status, stake, and settlement references.[5]

The chain is an implementation choice for a network whose participants do not share a natural central operator. A single-company deployment could keep equivalent state in a conventional registry. The security requirement is consistent, auditable state and clear custody, not the use of a specific ledger.

6.6 What Covenant establishes

- The public request begins with an objective rather than a provider-internal operation.
- Authority can be represented separately from model reasoning.
- Offers are concrete and remain non-binding until valid acceptance.
- Acceptance assigns provider responsibility under stated terms.
- Evidence and settlement become protocol objects instead of informal messages.

- Dispute is represented as a first-class lifecycle state.

6.7 Covenant's assurance boundary

Covenant makes authority, offer terms, acceptance, evidence, settlement, and dispute state explicit. Network policy and domain governance supply the surrounding assurance: authority validation, provider eligibility, broker disclosure, evidence profiles, verifier independence, and dispute handling. This division gives each decision a visible owner and a record that can be inspected across organizations.[8]

The current repository describes the protocol specification, onboarding framework, and chain-backed registry as functional while retaining active-draft status.[5]

7. Combined reference architecture

Lattice and Covenant occupy adjacent layers and can be deployed independently or together. Covenant defines the cross-boundary commitment, while Lattice can govern the selected provider's internal fulfillment.

Lattice can run a provider's internal fulfillment after a Covenant offer is accepted. It can also run internal work that never leaves the organization. Covenant does not require a provider to use Lattice. A provider may fulfill with a workflow engine, human operations, legacy software, or another runtime, provided it can honor the accepted terms and produce the required evidence.

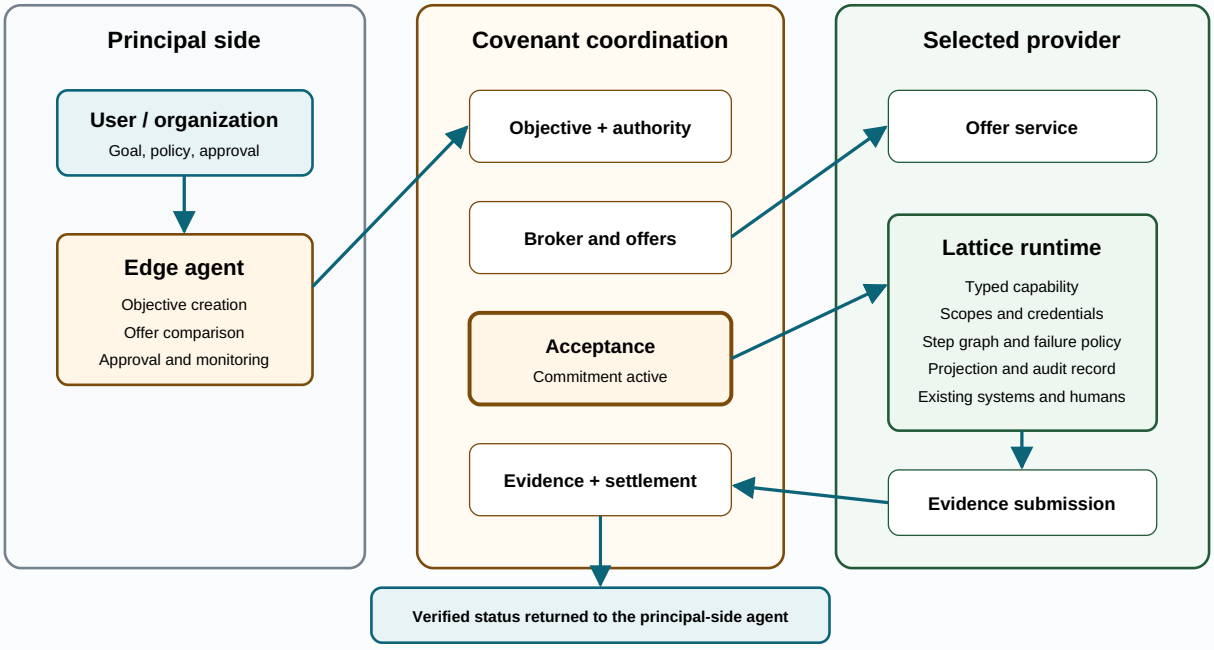

Figure 5. A Covenant commitment can be fulfilled by a provider that uses Lattice internally. The protocol remains independent of the provider's implementation.

Boundary cases

Scenario	Appropriate architecture
Internal supplier onboarding across compliance, ERP, documents, and finance teams	Lattice can coordinate the internal capability. Covenant is unnecessary unless an external party must make and honor a separate commitment.

Scenario	Appropriate architecture
Corporate travel purchased from competing external providers	Covenant can coordinate offers and acceptance. The selected provider may use Lattice or another internal runtime.
Internal capability needs a specialist external service midway through execution	A Lattice step can publish a Covenant objective, wait for settlement, then continue using the verified result.
Disposable coding task in an isolated workspace	Direct code execution may be sufficient. Lattice becomes relevant when the task crosses into shared or production state.

Where the outcome coordinate lives

In the combined architecture, the outcome coordinate has two views. The principal-side view carries the user's objective, authority, preferences, and approval conditions. The provider-side view carries the accepted terms and the internal capability needed to fulfill them. Evidence links the observed result back to the accepted commitment.

No component should silently strengthen its authority. An offer does not become authority. A model recommendation does not become acceptance. A typed capability request does not by itself prove that the principal authorized the resulting effect.

8. Security model and assurance boundaries

Outcome Coordination moves low-level execution into governed runtimes and accountable provider relationships. Prompt-injection defenses, credential isolation, transaction controls, application security, and human review remain part of the surrounding assurance model.

Assurance area	Controls in the architecture	Supporting practice
Intent integrity	Trusted instruction boundaries, source labeling, objective comparison, approval before acceptance	Preserve provenance for external content and request fresh approval when proposed terms materially change.
Delegated authority	Separate model proposals from authenticated actions, bind approval to exact terms, record the accepting principal	Grants should state purpose, limits, validity, and the class of commitments the agent may accept.
Capability and provider execution	Typed capability selection, reviewed dependency graphs, conformance, provider eligibility, idempotency, declared recovery policy	Business owners review capability versions, while provider history and evidence quality inform routing policy.
Evidence and settlement	Domain evidence profiles, verifier checks, lifecycle state, replay protection, dispute references	Each domain defines what evidence is sufficient for fulfillment, refund, failure, or dispute.
Privacy and infrastructure	Minimum disclosure, role-specific views, encrypted payloads, key management, append-only audit, reconciliation	Retention limits, selective attestations, isolation, and independent reconciliation reduce unnecessary trust and exposure.
Dispute and governance	Explicit dispute states, evidence retention, appeals, upgrade policy, operator accountability	The operating model defines when a case moves to contractual, legal, or domain-specific authority.

8.1 Prompt-injection controls at the coordination boundary

Moving execution out of the edge model reduces the number of low-level side effects the model directly controls. Source provenance, trusted instruction boundaries, objective diffs, and fresh approval for materially changed terms help protect the earlier stages where the model interprets the request and compares options.

8.2 Acceptance binds authority to exact terms

The acceptance boundary is strongest when the accepting actor is authenticated, the authority grant is current, the exact offer is bound to the signature, and any required approval is attached. A material change in price, date, refund policy, provider, or evidence terms produces a fresh decision rather than inheriting an earlier approval.

8.3 Evidence is defined by the domain

A booking confirmation, delivery photo, signed invoice, and ledger entry support different claims. Covenant can standardize references, integrity, timing, and lifecycle linkage, while a domain profile defines which evidence supports a particular settlement state.

8.4 Shared ledgers support custody and audit

An append-only registry gives multiple participants a durable record of status and settlement references. Off-chain evidence remains subject to domain verification. Chain anchoring is useful where agents, providers, brokers, and settlement services need a common state without assigning custody to one participant.

8.5 Outcome equivalence needs domain rules

Two execution paths may produce effects that are economically or operationally equivalent while differing in implementation. A small difference can also violate a hard condition. Domain profiles can define acceptable equivalence for a specific class of work, while Lattice narrows internal execution through capabilities and Covenant binds the provider to accepted terms.

> **Assurance boundary**
>
> Outcome Coordination makes authority, execution ownership, acceptance, evidence, and settlement easier to inspect. Its strongest use is alongside domain policy, secure implementation, qualified providers, evidence review, and human approval for consequential decisions.

9. Evaluation program

The repositories demonstrate that Outcome Coordination can be implemented across internal execution and cross-boundary coordination. Comparative experiments can now measure how the approach affects execution accuracy, recovery, context exposure, audit quality, and legitimate task completion.

9.1 Research hypotheses

ID	Hypothesis	Measurement
H1	Outcome-shaped capability requests reduce tool-selection and sequencing errors for multi-step internal workflows.	Task completion, wrong-operation rate, ordering violations, duplicate side effects
H2	Runtime-managed state and projections reduce sensitive and irrelevant data entering model context.	Tokens, sensitive-field exposure, context contamination, explanation accuracy
H3	Explicit acceptance reduces premature external side effects when offers or conditions change during a task.	Unauthorized acceptance rate, stale-offer acceptance, approval mismatch
H4	Provider-owned fulfillment improves recovery from operational failures compared with edge-agent orchestration.	Recovery success, partial completion, compensation quality, time to stable state
H5	Evidence and settlement objects improve post-incident reconstruction and dispute handling.	Audit completeness, attribution accuracy, dispute latency, unresolved cases
H6	The added controls preserve legitimate task completion while keeping correction burden and latency within acceptable limits.	Task completion, false-block rate, reauthorization frequency, user correction burden, completion latency

9.2 Lattice benchmark design

A Lattice evaluation should compare at least three implementations of the same workflow:

1. A direct tool-calling agent with endpoint-shaped tools.
2. An agent that writes code against the same endpoints inside a controlled runtime.
3. An agent using Lattice capability discovery and execution.

Scenarios should include changing API responses, ambiguous timeouts, partial service failure, missing approvals, stale identifiers, prompt injection in external content, and a maliciously plausible next step. Runs should use the same models, task instructions, credentials, and backend fixtures.

Useful measurements include task success, duplicate side effects, policy-order violations, scope failures caught before execution, raw payload tokens exposed to the model, projection omissions, audit completeness, execution latency, and human intervention.

9.3 Covenant benchmark design

A Covenant evaluation should simulate a market with honest, unreliable, and adversarial participants. The test environment should vary offer quality, provider failure rates, evidence strength, broker ranking bias, authority limits, and settlement policy.

Measurements should include objective fidelity, offer comparability, acceptance correctness, replay rejection, fulfillment variance, evidence sufficiency, settlement accuracy, broker concentration, provider reliability, privacy leakage, and dispute resolution time.

9.4 End-to-end evaluation

The combined test should begin with a user request and end with a verified effect. A Covenant provider can use Lattice internally, while another provider uses a conventional workflow engine. This isolates protocol behavior from provider implementation and tests whether the handoff between accepted terms and internal execution remains intact.

9.5 Reporting discipline

Results should distinguish prevention from detection. A blocked unsafe action, a corrected action after approval, and a harmful action discovered by audit are different outcomes. Reports should also separate implementation defects from model errors and protocol ambiguity.

10. Adoption path

10.1 Begin inside one trust boundary

Choose a workflow whose current execution path is known and whose failures are measurable. Vendor onboarding, procurement approval, account provisioning, or claims intake are suitable because they have existing owners and policy checkpoints.

1. Describe the outcome in business terms and identify the conditions that require approval.
2. Map the existing process, including human steps, system dependencies, failure paths, and compensation.
3. Define one capability signature that remains stable even if the internal implementation changes.
4. Design the projection from the decisions the model must make after execution. Remove fields that only support internal processing.
5. Declare scopes and validate them before the first step runs.
6. Exercise the capability against failure fixtures and inspect the audit record before production use.

A capability can begin with human tasks. Automation can replace selected steps without changing the model-facing contract.

10.2 Add cross-boundary commitments where responsibility changes

Covenant becomes relevant when another organization or independent service is expected to stand behind the result.

1. Define an Objective schema for the domain, including hard constraints, preferences, approval conditions, and expiry.
2. Define a provider Offer schema with concrete terms and evidence requirements.
3. Make acceptance an authenticated action bound to the exact offer hash and current authority grant.
4. Create an evidence profile that describes what supports fulfilled, failed, refunded, and disputed states.
5. Onboard participants through conformance and probation before routing meaningful traffic.
6. Record settlement in a registry whose custody model fits the participating organizations.

10.3 Preserve existing systems

Outcome Coordination is an additional interface above existing APIs and workflows. It does not require providers to replace their booking systems, ERP platforms, queues, human operations, or payment rails. The provider gains freedom to change internal implementation while keeping the public offer and evidence contract stable.

10.4 Use a narrow first domain

Cross-domain protocols tend to become vague. A first production profile should stay within one domain, define evidence precisely, and limit the classes of authority an agent can exercise. Portability can follow after the lifecycle and disputes are understood in practice.

11. Open research questions

The architecture creates a clearer research surface. Several questions remain open.

Deriving the coordinate

How should a system separate hard constraints from preferences when the user's language is ambiguous? Which uncertainty should be resolved before offers are requested, and which can remain as a ranking preference?

Revision and reauthorization

Which changes can the agent make during negotiation under an existing grant? When price, time, provider, risk, or evidence terms change, what threshold requires a fresh user decision?

Outcome equivalence

How can a verifier determine that an observed result is equivalent to the accepted outcome when the domain allows substitution? Can equivalence rules be represented in a portable profile without recreating the full provider workflow?

Projection completeness

How can projection designers measure whether the model has enough information to explain the result without exposing sensitive or distracting data? Can omitted-risk testing become part of capability review?

Authority provenance

How should authority flow through user, organization, agent, broker, and provider roles without becoming broader at each hop? Which revocation and delegation models remain practical in long-running tasks?

Evidence quality

Which evidence combinations support a settlement state in travel, logistics, procurement, claims, or identity operations? How should systems handle evidence that is authentic but incomplete or misleading?

Broker incentives

How can routing and ranking remain inspectable when brokers earn fees, operate related providers, or control access to demand? Which market structures reduce suppression and self-preferencing?

Provider-side agent use

A provider may use its own models internally. How should Covenant conformance evaluate the provider's operational controls without prescribing its implementation? Can a Lattice audit record be used as supporting evidence without exposing confidential internal state?

Privacy and selective disclosure

Objectives can reveal travel plans, medical needs, spending limits, and organizational intent. How can providers evaluate feasibility with the minimum data, and when can evidence be verified through attestations rather than full payload disclosure?

Human responsibility

Which decisions should remain human even when technical acceptance is valid? How should the system represent institutional responsibility when a model recommends, a person approves, and a provider fulfills?

12. Conclusion

Agent systems currently place a large share of execution work inside a probabilistic model loop. The model selects operations, carries state, interprets errors, and creates authenticated effects. Standard controls can establish identity, permission, and request validity while leaving the relationship between the effect and the user's task unresolved.

Outcome Coordination reorganizes that boundary. Inside one organization, Lattice represents work as typed capabilities executed by a runtime that owns state, scopes, credentials, failure policy, human steps, projections, and audit. Across organizations, Covenant represents work as objectives, offers, exact acceptance, provider commitments, evidence, settlement, and dispute.

The two projects address different scopes. Lattice is a provider-side or enterprise execution architecture. Covenant is a public coordination protocol. They meet when a cross-boundary commitment is fulfilled through a governed internal capability.

The proposal should be judged through measured results rather than broad safety claims. Its strongest contribution is a cleaner place to ask the question that tool authorization leaves open: did the effect remain within the outcome, authority, terms, and evidence that governed the task?

References

1. Abbasi, H. "The Future of Agents Is Outcome Coordination." Level Up Coding, 11 March 2026. Publication.
2. Abbasi, H. "The Future of Agents Is Outcome Coordination, Part II." 13 March 2026. Publication.
3. Abbasi, H. "The Missing Runtime Between AI Agents and Enterprise Backends, Part 2 of 2." 20 May 2026. Publication.
4. Abbasi, H. *Lattice: The capability runtime for outcome-based execution.* GitHub repository, open design and implementation. Repository.
5. Abbasi, H. *Covenant Layer: Open protocol and framework for outcome-based coordination.* GitHub repository, active draft and implementation. Repository.
6. Covenant Protocol. "Introduction." Defines the protocol scope, public coordination model, and provider-internal exclusions. Specification.
7. Covenant Protocol. "Core Protocol." Defines roles, objects, lifecycle, integrity baseline, and protocol rules. Specification.
8. Covenant Protocol. "Security." Defines threat-model changes, acceptance semantics, evidence limits, broker power, replay, privacy, and dispute. Specification.
9. Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. "Lost in the Middle: How Language Models Use Long Contexts." 2023. arXiv.
10. Fourney, A., Payne, T., Murad, M., and Amershi, S. "Tool-space interference in the MCP era: Designing for agent compatibility at scale." Microsoft Research, 11 September 2025. Research article.

Suggested citation

Abbasi, H. (2026). *Outcome Coordination for AI Agents: Governing Execution Within Systems and Commitments Across Trust Boundaries.* White Paper, Version 1.4, August 2026. https://doi.org/10.5281/zenodo.21792756.

Appendix: object and control summary

Layer	Primary object	Consequential boundary	Evidence produced
Model interaction	User request, task-scoped projection, recommendation	User clarification or approval	Conversation record and decision rationale
Lattice	Capability signature and compiled execution plan	Runtime execution after input and scope validation	Projection and structured audit record
Covenant	Objective, Authority Grant, Offer	Valid Acceptance of exact offer terms	Evidence Record and Settlement Receipt
Provider internals	Domain workflow, application transaction, human task	Provider-defined production action	Domain proof, receipt, status, or failure record

Minimum review questions

- What exact effect can occur, and which component causes it?
- Which principal's authority is used at the consequential boundary?
- What conditions require a fresh approval?
- Which state remains outside model context?
- How are retries, duplicates, partial failure, and compensation handled?
- What evidence supports completion, and who evaluates it?
- How can a user or operator dispute the recorded result?
- Which claims are implemented today, and which remain research hypotheses?