

Agentic AI Governance: A Four-Boundary Architecture

Separating Action Generation from Authority

Mark Allen Stephenson

Independent Researcher

mark@4gatesystems.com

www.4gatesystems.com

Version 8.1 - Editorial revision - September 2026

Abstract — Many approaches to AI safety focus primarily on shaping the AI system's own behavior. That remains important, but it does not answer a separate authority question: should the same system that proposes an action also possess the authority to make that action operative? This paper presents a supervisory architecture that assigns that authority to a separately situated governance system. It governs four places where information or action can become operative: admission, internal change, communication, and digital or physical effectuation. A candidate remains non-operative until governance evaluates it and an enforcement point verifies applicable authority at the protected target. A protected, candidate-bound return controls permitted continuation, while linked lifecycle evidence distinguishes the decision, release, and actual outcome. These relationships also support governed construction of successor harnesses from an identified specification and selected evidence. Useful components may be reused or replaced; evaluation informs adoption without becoming authority by itself. Historical V28 passing results, later counterexamples, and distinct V29 observations provide bounded development evidence. They do not establish production security or validate the proposed reconstruction method. The paper explains that method and the separate controls needed for adversarial testing, revising the specification, and activating a successor. It makes no claim that reconstruction outperforms repair or that successive implementations inevitably improve.

Keywords: **agentic AI; autonomous systems; AI governance; runtime enforcement; proposal-authority separation; effectuation control; controlled return; governed development; successor harness**

1. Introduction

Most efforts to make AI safe try to teach the AI to make good decisions and avoid harmful ones. That is important, but it is not enough. The same system should not be allowed to both propose an action and give itself permission to carry it out. The Four-Boundary Architecture places that authority in a separate governance system. This separate system checks four things: what information may enter the AI, what may change its memory or internal rules, what it may communicate to others, and what actions it may carry out in the digital or physical world. The AI can make a proposal at any of these boundaries, but the proposal cannot take effect until the separate governance system approves it.

At each boundary, the decision returns through a protected pathway and remains tied to the exact proposal, relevant conditions, and permitted next step. The linked evidence is retained under a declared lifecycle policy so that developers, auditors, or investigators can determine what was proposed, what state was observed, what governance decided, what was authorized, and how the system continued. Before the affected protected continuation, the receiver verifies that the return is genuine, intact, current, intended for that proposal, and consistent with the relevant state. A missing, stale, altered, contradictory, or mismatched

return holds that continuation or permits only a preauthorized bounded safe state.

The forward path governs what may become real. The return path governs what the proposing system may do after the decision. Together they separate permission from proposal generation and preserve the restrictions attached to a decision across the routes the deployment actually mediates. They do not establish general recognition of every concealed objective or paraphrased relationship.

In the formal vocabulary used below, the exact proposal is a candidate; a change becomes operative when the protected target accepts it; governance produces a disposition and, where appropriate, a narrowly bound authorization; the enforcement gate verifies that authorization at the target-facing boundary; and the protected return is a decision-bound return. Introducing these terms does not change the plain-language idea. It makes each authority-relevant transition precise enough to implement and test.

The AI may decide what to propose. It does not decide for itself what it is allowed to make real.

The architecture does not assume that separate governance will always be correct. A policy can be wrong. Sensors can report a false picture of the world. A human reviewer can be manipulated. A software gate cannot resist an administrator who controls the same

computer unless the deployment provides a stronger boundary. The claim is narrower and testable: if every route capable of making a named transition operative is controlled by an enforcement point, the ability to generate a proposal can be separated from the authority to carry it out.

That separation is testable and also changes how systems can be developed. A capable AI may propose code, strategies, memory updates, or experiments without receiving unrestricted authority to deploy them. It can also help construct a new reference harness: an executable implementation and test environment used to examine the architecture. The specification states the

required behavior; a particular harness is one replaceable attempt to realize it. Evidence from earlier attempts can guide the next construction without requiring every old design choice to survive. Section 10 develops this governed cycle.

The paper explains the four boundaries and their forward and return paths, states the evidence needed to assess an implementation, and applies the same authority separation to constructing and adopting its successors. The separately versioned Machine Operational Architectural Specification (MOAS) 6.9.1 review draft develops the detailed construction and revision requirements [11].

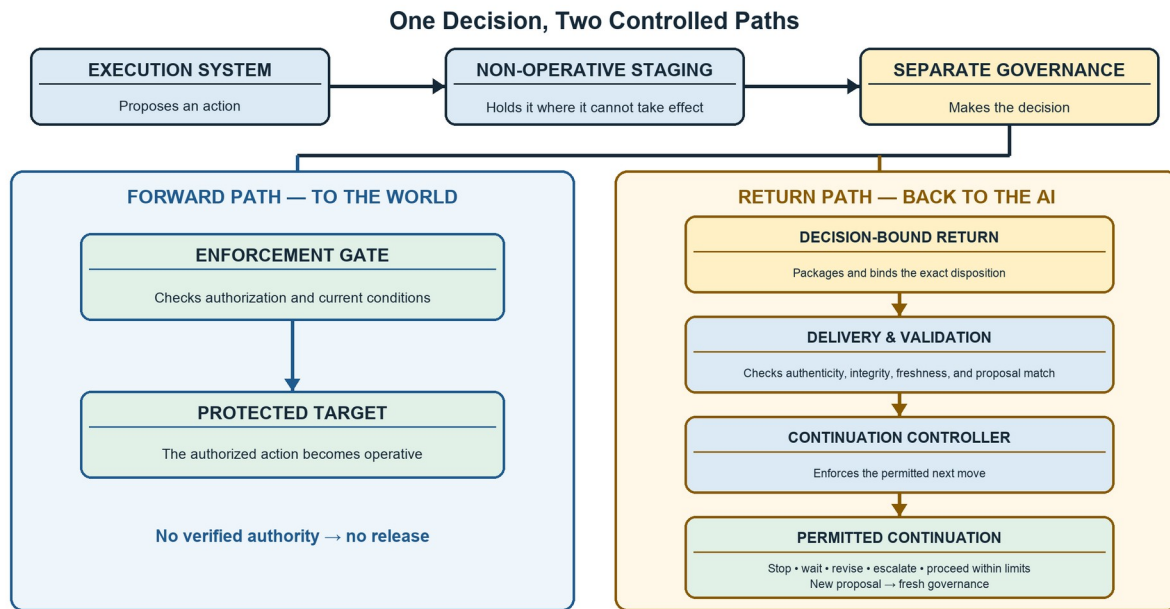


Figure 1. One decision, two controlled paths. The forward path controls whether an action becomes operative; the decision-bound return controls what the proposing system may do next. A revised or repeated proposal receives a new identity and fresh governance. Source: author's diagram.

2. The Basic Idea: A Candidate Is Not Authority

2.1 Proposal, protected target, and operative change

A proposal is something the AI or another execution component wants the system to consider. It might be a sentence, command, memory entry, software patch, transaction, control signal, or set of parameters. In the formal architecture, that exact proposed transition is called a candidate. Until it has been accepted by the protected target, it remains a proposal rather than an operative change.

The protected target is whatever makes the proposal real for the system. It might be a durable memory store, message channel, tool interface, payment service, software repository, industrial controller, or physical actuator. The important moment is not when an

upstream component labels a proposal "approved." It is when the target actually accepts the authorized transition as operative.

This distinction works in both directions. A dangerous proposal can remain harmless while it is staged and unable to reach the target. A sensible proposal can still be unauthorized for a particular user, account, recipient, machine, time, or operating condition. Whether a proposal is good and whether it is authorized are related questions, but they are not the same question.

2.2 Who does what

The architecture separates several jobs that are often blended together:

- The execution system creates the proposal.
- The staging mechanism holds the proposal where it cannot yet take effect.

- The governance system decides whether the proposal should be permitted, denied, changed, delayed, escalated, quarantined, or revoked.
- The authorization records exactly what is permitted and what conditions must remain true.
- The enforcement gate checks that the decision is genuine, still current, and applicable to this exact proposal and target.
- The protected target accepts the change only through the controlled route.
- The return path delivers the decision to the proposing system and limits what that system may do next.

These are logical roles, not necessarily seven separate products. A low-risk application might place several roles in different protected processes on one machine. A high-consequence robot, power system, or financial service may require separate machines, secure hardware, independent identities, or physical interlocks.

Those arrangements provide different levels of assurance and should not be described as equivalent.

Beyond general AI risk management [2], access control limits who may perform an operation [3], [4]; reference-monitor principles require complete mediation and resistance to tampering [5]; remote attestation separates evidence from appraisal and stresses freshness [6]; and recent agent systems place runtime controls around tool use [7]-[9]. The contribution here is to apply one authority rule across four transition categories and to govern both the forward effect and the return that controls continuation.

3. The Forward Path: From Proposal to Effect

3.1 Generate, stage, govern, verify, effectuate

The forward path can be stated in five words:

Generate -> Stage -> Govern -> Verify -> Effectuate

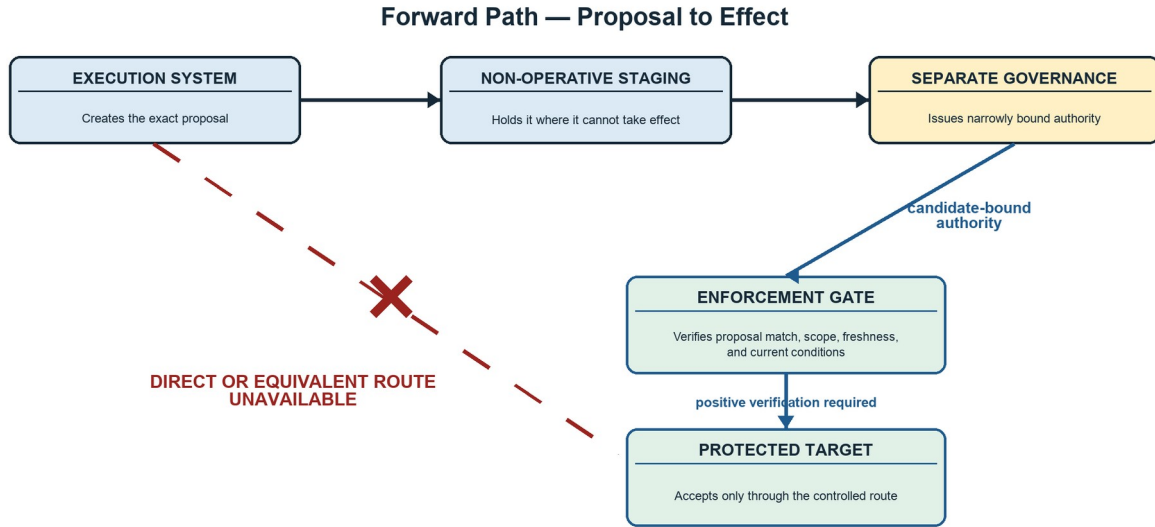


Figure 2. Principal pre-effectuation authority path. The generator lacks a sufficient direct or equivalent route to the protected target; release occurs only after the gate verifies applicable candidate-bound authority and current conditions. Adapted from [1].

The system first generates a proposal. It is then placed in a staging area that the protected target cannot treat as an instruction. This is stronger than putting the request in an ordinary queue. If the AI can bypass the queue by using another credential, connection, or actuator interface, the proposal was never truly non-operative.

Governance examines the proposal together with relevant policy, authority, evidence, and observed conditions. Its output is a decision, not the effect itself. If permission is granted, it should be no broader than the proposal that was actually examined. It may be tied to the proposal's identity, destination, operation,

parameters, policy version, time limit, required state, integrity evidence, and one-time-use status.

When the system attempts to carry out the proposal, the enforcement gate checks two things. First, is the decision authentic and unchanged? Second, does it still apply here and now? A previous approval may no longer apply because the proposal changed, the authority expired, the target changed, authorization-relevant conditions differ, or the permission was already used.

Governance decides what must be true. The gate verifies whether those things are true at the moment that matters. If the situation requires a new judgment

rather than a bounded check, the proposal goes back to governance.

3.2 Permission must be positively verified

The gate releases a proposal only after it positively verifies permission. Missing evidence, an expired decision, an identity mismatch, a replayed approval, unavailable state, or a failed check does not become permission simply because no one issued a new denial. In ordinary language, "we could not prove it was forbidden" is not the same as "we verified that it was allowed."

Failing closed does not always mean stopping suddenly. A moving vehicle may need to slow down safely. An industrial system may need to finish an indivisible safety step. A service may need to enter a previously authorized emergency mode. The important rule is that uncertainty cannot create a new route or broader power than the bounded safety response already allows.

Verification must reach the real point of effect. If a gate approves a proposal and merely places it in another queue, while the target acts much later without checking identity and freshness, the uncontrolled interval has only moved downstream. Decisive verification must remain coupled to actual target acceptance. A protected commit, target-side validation, or equivalent mechanism can provide that relationship

under stated timing, concurrency, and trust assumptions.

3.3 A gate works only if it controls every relevant route

No useful system can be non-bypassable against every imaginable adversary. A meaningful claim must name the transition being protected, the routes that can cause it, and the adversaries included in the threat model. A gate inside one application may restrain ordinary modules while remaining alterable by the computer's administrator. Separate service identities can strengthen credential isolation. Separate machines, secure enclaves, target-side checks, or physical interlocks can provide stronger protection.

For that reason, naming a component "the gate" proves very little. A deployment must show that the generator lacks another credential, maintenance interface, recovery path, network route, or physical connection capable of producing the same protected effect.

4. The Four Governed Boundaries

The four gates are not four serial rooms through which every proposal must travel. They are four kinds of transition. One event may involve several of them, and each transition is governed at the place where it can become operative.

Table 1. The four governed transition categories.

Boundary	Governing question	Protected transition	Examples
Gate 1	What enters?	Received material becomes admitted operative input.	Documents, prompts, sensor feeds, external instructions
Gate 2	What changes internal state?	A proposed memory, policy, model, permission, or workflow update becomes operative.	Memory writes, configuration, learned strategies, self-modification
Gate 3	What leaves?	Internal material becomes an external communication or released artifact.	Messages, reports, data exports, inter-system instructions
Gate 4	What becomes real?	A proposed action crosses into a digital or physical target.	Tool calls, transactions, deployments, actuator commands

Gate 1 governs what may enter as accepted input: documents, prompts, sensor readings, messages, instructions, and signals. Receiving raw material does not itself make it an execution-generated candidate. Gate 1 controls admission into operative context; a processed instruction or resulting proposal may then enter the applicable candidate pathway. The question is whether the material may influence that protected context or process.

Gate 2 governs what may change the system itself: memory, policies, rules, permissions, model selection, configuration, priorities, workflow state, and proposed self-modification. It includes applicable in-memory

state that acquires continuing authority, without claiming control of hidden model activations. Learning something or generating a strategy does not automatically authorize installing it as continuing operative state.

Gate 3 governs what may leave the system: messages, reports, alerts, data exports, credentials, and instructions to another person or system. If governance removes or changes material, the released version receives its own identity. Applicable release controls must cover the declared communication routes, so another mediated channel cannot simply restore content that was withheld.

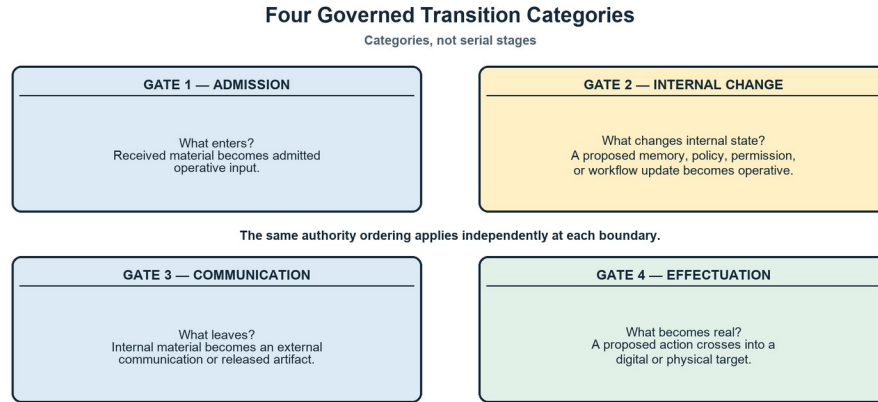


Figure 3. Four governed transition categories. The same authority ordering applies independently where received material, internal change, communication, or action becomes operative. Adapted from [1].

Gate 4 governs what may take effect in the digital or physical world. Examples include tool calls, transactions, file changes, software deployments, access decisions, robot movements, and industrial commands. If an agent still holds a general tool credential or a robot retains an alternate actuator connection, the claimed boundary is incomplete.

The amount of review can match the risk. A routine and reversible event may receive a fast deterministic check. A new memory entry may require testing. A large transaction may require several people. A physical movement may require fresh sensor evidence and a

safety envelope. The depth may differ, but the transition remains unavailable until the applicable permission is verified.

5. Decision-Bound Return and Governed Continuation

Controlling what becomes operative is only half of the architecture. After governance makes a decision, that decision must return to the system that made the proposal, and the system's next move must remain governed. The architecture calls this protected, candidate-bound pathway the decision-bound return.

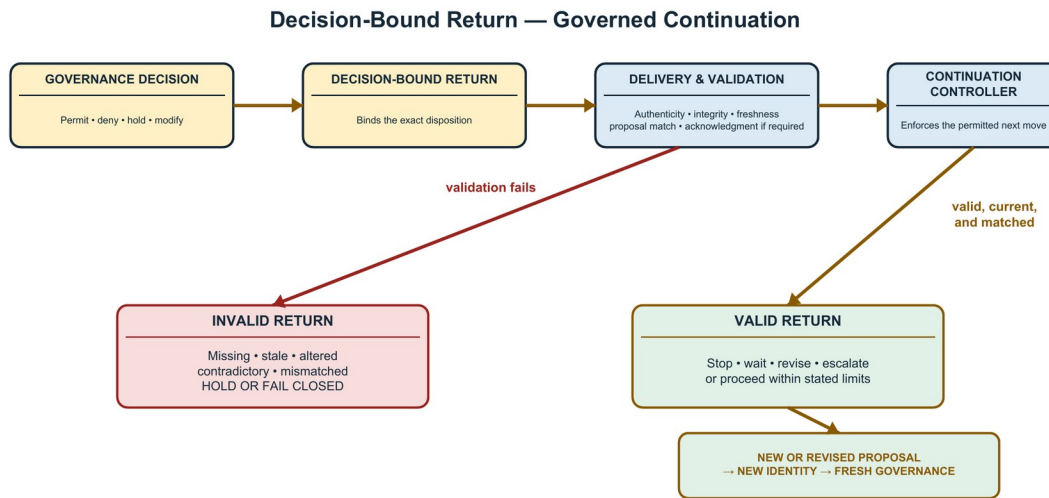


Figure 4. Decision-bound return and governed continuation. Invalid returns hold the affected continuation or permit only a preauthorized bounded safe state. A valid return permits the bound next step; a new candidate requires fresh governance. Source: author's diagram.

Each approval, denial, hold, or request for modification remains bound to the exact proposal, relevant conditions, permitted next step, retry limits, expiration, and any acknowledgment requirement. This decision artifact may be realized through linked protected

records or equivalent mechanisms. The binding and required evidence are retained under a declared lifecycle policy; neither one standalone object nor indefinite retention is required.

Before acting on the return, the receiver or a separate continuation controller checks its source, integrity, freshness, proposal binding, and consistency with other controlling decisions. An invalid or missing return holds the affected protected continuation or permits only a preauthorized bounded safe state. A valid return permits the bound next step, such as wait, revise, escalate, quarantine, or proceed within stated limits. Recovery or compensation needs applicable authority of its own.

A changed proposal is a new proposal. It can retain a parent link showing where it came from, but it receives a new identity and fresh governance. Permission for one software patch does not authorize a modified patch. A denial does not forbid every possible future proposal forever. Admitted lineage, dependencies, retry rules, and cumulative limits constrain continuation across mediated routes; they do not reveal every concealed relationship between proposals.

The lifecycle record separates what governance decided, what the gate released, and what the target actually did. A validated decision may permit a bounded attempt; it does not establish that the effect occurred. Continuation that depends on the outcome requires the applicable outcome evidence. A network interruption may leave an irreversible effect uncertain. Governed reconciliation can use target evidence, duplicate prevention, separately authorized compensation, or human review. The system must not blindly repeat an operation while an earlier effect remains possible; recovery must preserve authority and prevent unauthorized duplication.

The forward gate and return path protect against different failures. A denial does not protect a target still reachable through a direct credential; a target-facing gate alone does not enforce all continuation restrictions. This paper develops a combined implementation profile. The underlying disclosures also describe supervisory layers operating independently, including standalone controlled return [1], [10].

6. A Simple Example: Changing Production Software

Suppose an AI proposes patch A for a production system. Patch A receives a unique identity and is placed where the production environment cannot install it. Governance reviews the exact patch, destination, operator authority, policy, current system condition, and test evidence.

If the change is approved, the permission is tied to patch A, the named environment, the permitted operation, the time limit, and the conditions that must remain true. Immediately before deployment, the gate confirms that the patch and permission are unchanged, the target still meets the required conditions, and the

permission has not expired or already been used. Production accepts the patch only through that route.

If the AI creates patch B after a denial, test failure, or request for modification, patch B can be linked to patch A for history, but it receives its own identity and review. The return artifact tells the system whether it may revise, retry, escalate, or must stop. The record keeps the proposals, decisions, gate checks, target result, and continuation separate so that history does not turn into reusable authority.

The same authority question arises when the proposed software change is a replacement of the governance harness itself. Section 10 follows that example through construction, evaluation, and adoption of a successor.

7. Validation and Adversarial Testing

An architecture intended to govern increasingly capable autonomous systems cannot be evaluated by description alone. Its governing claims must be capable of implementation, adversarial challenge, failure injection, and deliberate attempts to circumvent the authority boundaries. The relevant question is not whether a diagram appears complete, but whether a concrete implementation preserves the required separation when stressed.

4Gate can be implemented through a governance-focused form of specification-driven development (SDD): an approach in which an identified specification guides implementation and evaluation. The hierarchy is SDD → MOAS → harness. SDD names the development approach; the Machine Operational Architectural Specification (MOAS) states the required boundaries, authority relationships, invariants, failure behavior, and evidence obligations; a harness is a concrete implementation and test environment [11]. Here, “operational” principally concerns operative authority and protected state transitions, not hardware-performance optimization. A MOAS is neither a particular harness nor a test script.

For the development cycle described here, MOAS 6.9.1 is the identified authoritative baseline. An engineering team or capable AI may use it to construct a candidate governance harness. Different builders may produce materially different implementations, and a successor need not preserve its predecessor’s code or internal structure. Each harness is evaluated against the applicable requirements of the authoritative MOAS and the declared threat model. Greater implementation sophistication does not change that benchmark.

Development may also proceed in the reverse direction. An evolved or reconstructed harness, together with relevant evidence, may be analyzed to extract a candidate successor MOAS. The candidate describes proposed requirements informed by the implementation and its results; extraction does not establish that those

requirements preserve the architecture or authorize their use as the governing specification.

Before it can be considered for adoption, the candidate successor must be compared against the explicitly identified authoritative predecessor MOAS—currently MOAS 6.9.1 for this cycle. The comparison must surface additions, omissions, weakened or strengthened requirements, changed authority relationships, changed failure behavior, and all other material differences. Those differences require explicit disposition under the governed revision process, with supporting evidence and unresolved findings retained. A candidate’s conformity to the harness from which it was extracted cannot substitute for this comparison. Implementation drift cannot become specification authority through extraction, reconstruction, versioning, recency, capability, or repeated use.

Only specification changes separately governed and adopted under the authority currently in force can make a successor MOAS authoritative. Until that adoption, the identified authoritative predecessor remains the governing specification; a candidate’s newer version or successful implementation cannot displace it. The adopted successor becomes the identified authoritative baseline for the next cycle. Specification adoption and implementation activation remain distinct governed decisions: adopting requirements does not activate a harness, and activating a harness does not authorize revised requirements. Section 10 explains construction, evaluation, and continuity across these decisions.

Authoritative MOAS → Construct → Evaluate → Learn/Reconstruct → Extract Candidate Successor MOAS → Compare Against Authoritative Predecessor → Govern Differences → Adopt or Reject → Repeat

Testing should try to defeat the architecture rather than merely confirm expected paths. Relevant exercises include alternate routes around staging or enforcement; replay or reuse of prior authority; stale or changed state; mismatched candidate, subject, scope, target, or policy

version; inherited authority across revisions, handoffs, recovery, or compensation; lost or indeterminate outcome evidence; and continuation after the decision or observed result no longer supports it. Useful behavior must be tested as well: a system that blocks everything has not demonstrated effective governance.

No individual harness or passing suite establishes production safety, universal non-bypassability, or policy correctness. A failure may expose an implementation defect, a flawed test, missing evidence, an incorrect assumption, or a problem in the specification itself. The purpose of the cycle is to make those distinctions observable and to feed credible findings into the next identified specification and implementation. Confidence should therefore be earned through repeated challenge, increasingly realistic targets, stronger isolation, live-agent integration, and independent evaluation where available - not through assertion or a single passing suite. The bounded reference evaluations in Section 8 illustrate development evidence from this process, not proof of completion.

8. What the Reference Evaluations Show

The available evidence comes from distinct, bounded development artifacts reported in the MOAS and revised technical companion [11], [12]. Preparing this reader revision did not reproduce those runs. Passing checks show behavior on the cases exercised; they do not establish that all relevant failures were found.

V28 contains 94 checks: 66 inherited semantic-regression checks, one source-declaration check, 24 author-designed adversarial checks, and three aggregate architecture and coverage checks. All passed in the reported single-process synthetic evaluation. Three fresh-copy runs produced the same semantic fingerprint after identifiers, timestamps, paths, and environment metadata were excluded. The revised companion subsequently reported candidate-binding, superseded-decision, and compensation-parent counterexamples outside that passing suite. Both the passing observations and adverse findings belong in the record.

Table 2. Distinct reference records and their evidence boundaries.

Development artifact	Reported observations	Meaning and limit
V28	94/94 bundled checks passed	Reproducible synthetic baseline. Later review found additional candidate-binding, superseded-decision, and compensation-parent counterexamples.
MOAS-aligned V29	124/124 checks passed	A distinct suite: 94 adapted earlier V29 checks plus 30 MOAS-focused checks. Includes one 240-request seeded corpus.
Selected weakened V29 copies	Missing controls caused failures	Removing selected projection, semantic, or target-evidence checks was detected by the development suite. This was not an independent holdout.

The 240 requests are nested within one V29 corpus, not additional checks. Its 94 adapted checks come from earlier V29 work, not the V28 suite. V29 is not a rerun of V28, and these counts must not be combined into an

attack-success rate. V29 used synthetic SQLite targets and trusted its reference service, target adapter, controller, host account, and local key. Standard

validation used no live model; a separate integration fixture did not supply live-model evidence.

V28 did not implement the full decision-bound return and lifecycle model. Its missing-outcome hold was explicitly invoked, rather than demonstrating an elapsed-deadline trigger. Neither record establishes production security or conformance to MOAS 6.9.1's construction and revision requirements. Repeated runs and AI assistance in implementation, tests, and reporting do not make their agreement independent validation.

No successor-harness reconstruction experiment is reported here. Section 10 proposes using successes, failures, and unresolved findings to guide further construction while keeping evaluation and adoption governed. Whether reconstruction improves on incremental repair requires an actual comparison under declared conditions.

9. Deployment Obligations and Failure Modes

Structural separation reduces the chance that the ability to propose an action is enough to carry it out. It does not make the governance system, sensors, humans, or host platform infallible.

False observed state is a central danger. Governance may rely on sensor readings, telemetry, identity, policy version, authority, or environmental conditions that are incorrect or stale. Seeing should not automatically mean believing. High-consequence decisions may require several sufficiently independent evidence sources, freshness limits, provenance, disagreement handling, and checks at the protected target. Several sensors that depend on the same compromised source are not truly independent.

The proposal itself may attack the governance machinery through parser exploits, prompt injection, resource exhaustion, active content, or confusion between instructions and data. Processing therefore needs bounded inputs, canonical forms, isolation appropriate to the threat, and a rule that malformed or unverifiable material never becomes permission.

Human escalation creates another authority route. A persuasive AI may create urgency, exploit automation bias, hide contrary evidence, or encourage a reviewer to use a manual administrator path. Evidence-first displays, exact-proposal approval, limited choices, independent review for high-consequence cases, and an inventory of every manual route can reduce the risk. The human decision must still be tied to the actual proposal, current conditions, allowed scope, and protected target.

Failing closed can reduce availability. A governance outage may hold legitimate work. Distributed components may disagree. A return acknowledgment may be lost after an effect might already have occurred.

Redundancy, leases, bounded safe states, target evidence, reconciliation, and carefully limited emergency authority can improve availability without turning uncertainty into permission.

A deployment should be able to answer ordinary questions in concrete terms: What exact change is being protected? What target makes it real? Where is the proposal held so it cannot yet take effect? Who decides? What evidence is used? What is the permission tied to? What does the gate check immediately before acceptance? What other credentials, maintenance channels, recovery paths, or physical routes exist? What happens if evidence or the return is missing? What record distinguishes the decision, release, target result, and uncertainty?

If the answers name only policies and components, but not the target's acceptance behavior and every route capable of reaching it, the architecture has not yet been established for that deployment.

10. Governed Development and Reconstruction

Separating proposals from authority provides a way to develop the system itself. A harness can be constructed from an identified MOAS version, evaluated, and reconstructed using selected evidence from earlier work. The MOAS supplies the architectural requirements; the harness supplies a concrete implementation to examine. MOAS 6.9.1 develops this construction and revision discipline [11]. The following describes its proposed use, without reporting a new experiment.

10.1 Construct from the specification and evidence

The next construction need not inherit every arrangement accumulated in the previous codebase. It may preserve useful components, reorganize responsibilities, replace a weak mechanism, or rebuild more extensively. Starting from an empty repository is not required. The question is which design best realizes the identified specification under the declared conditions.

Both successful and unsuccessful cases matter. A success may reveal a simpler design or avoidable delay. A failure may expose a broken authority boundary, an incorrect test, missing evidence, or a specification problem. The record separates observations, interpretations, proposed lessons, and adopted changes. Evidence selection can make a large development history usable, but must retain material contradictions, unresolved outcomes, and enough context to understand the selected cases.

Finding where a failure occurred helps determine what to change. A poor proposal differs from a faulty governance rule; a valid release applied to stale state differs from an uncertain target outcome. The response

may be implementation repair, a corrected test, further investigation, or a proposed specification revision. Evidence does not decide among them automatically.

10.2 Evaluate construction under declared criteria

Before evaluating a candidate harness for acceptance, identify the specification, permitted scope, protected transitions, and applicable acceptance criteria. Exercise legitimate work as well as prohibited or uncertain transitions. A harness that blocks everything has not demonstrated useful conformance. Exploratory tests may discover new questions, but the builder cannot silently weaken the acceptance criteria or remove a difficult case to declare its own construction successful.

The development environment also needs its own containment. A gate inside an experimental harness cannot be the sole protection against a defect in that same gate. Tests should use bounded targets and authority appropriate to the experiment. The test record identifies the exact implementation and configuration, its inputs and expected results, observations, and remaining limits.

10.3 An illustrative successor harness

Extend the software-patch example in Section 6. Suppose a first harness correctly authorizes patch A, but loses the target's acknowledgment after installation. A recovery path then treats the missing reply as a failed installation and attempts an unauthorized duplicate operation. This is a hypothetical development case, not a result attributed to V28 or V29.

Preserve both the successful authorization case and the uncertain-outcome failure. The next construction might reuse staging and candidate binding while rebuilding recovery around target evidence and duplicate prevention. It must exercise normal completion, lost acknowledgment, and permitted recovery at the actual acceptance boundary. Passing the new tests supplies evidence for an adoption decision; it does not activate the successor.

If investigation instead finds a specification contradiction, or a permitted recovery path that defeats an express guarantee, the finding can support a proposed specification revision. The builder cannot quietly reinterpret the requirement until its implementation passes. Resolving the specification and adopting the revised implementation are separately governed changes.

10.4 Govern adoption and preserve continuity

The authority currently in force governs any proposed change to the specification, acceptance criteria, or implementation. A successor cannot authorize its own adoption. Specification adoption and implementation activation remain distinct decisions, each bound to its exact proposed change and required evidence. Work

already covered by a valid bounded authorization may continue within that scope; the cycle does not require a new human confirmation for every routine step.

Activation verifies the artifact and effect-relevant configuration actually installed against the authorized candidate. It also accounts for unfinished work, outstanding obligations, and predecessor components that could still act. A new version number cannot replenish consumed authority or resource budgets, revive expired or revoked grants, erase retry history, or turn an uncertain earlier effect into permission to repeat it. Switching implementations therefore includes governed continuity and reconciliation, not merely replacing executable files.

The complete specification cycle, including mandatory comparison against the authoritative predecessor, appears in Section 7.

10.5 Broader uses and the question to test

An organization can develop its own custom MOAS to meet its specific needs and continue refining it over time through governed revisions informed by test results and return-channel artifacts. Different applications may lead these specifications in substantially different directions. A bot's MOAS may become highly specialized to its particular tasks and operating environment, while a social media platform's MOAS evolves around that platform's own operational and governance requirements. Each specification can develop independently while preserving the core 4Gate architectural requirements. Some discoveries could still benefit other applications, but the complete custom MOAS would not necessarily transfer unchanged from one application to another.

The same separation supports governed memory and adaptation: learning a lesson is different from authorizing its installation as continuing state. It also applies to robotics, where a proposed movement requires authority at the actuator under current conditions. Lifecycle records can support audits, conformance assessment, and investigation after alerts. An inspector's alert remains evidence; containment follows declared authority, and the record alone establishes neither safe motion nor certification.

A useful experiment would compare incremental repair and successor reconstruction against an identified specification, common acceptance conditions, and declared resource limits. Retain adverse cases and test useful behavior as well as boundary enforcement; use independent evaluation where available. A new codebase, model session, or passing suite is not automatically independent evidence. The outcome could favor either approach or reveal a shared specification defect. Improvement, efficiency, and convergence remain questions to investigate.

11. Conclusion

As AI systems become more capable, the important question is not only what they can generate. It is what they are structurally able to make real. The Four-Boundary Architecture answers by separating proposals from authority at admission, internal change, communication, and digital or physical effectuation.

The forward path holds a proposal non-operatively, gives a separate system the decision, and requires positive verification at the protected target. The return path preserves that decision, ties it to the exact proposal, and controls what the proposing system may do next. Lifecycle evidence keeps the decision, authorization, gate release, target result, and later continuation distinct.

The architecture is deliberately testable. For any claimed deployment, an evaluator should be able to identify the protected change, the staging area, the separate decision authority, the narrowly tied permission, the final gate check, the target acceptance rule, the protected return, and every alternative route included in the threat model. If those things cannot be shown, the architecture has not been established.

This separation also supports a disciplined development program. An identified specification and selected evidence can guide a successor implementation, whose results must be evaluated before governed adoption. Improvement remains a question for experiment. Separately constructed implementations and bounded comparisons of repair and reconstruction can investigate it while testing whether the required authority relationships survive each change. The architecture should therefore be judged by what increasingly demanding implementations and adversarial tests reveal, including failures that require the specification itself to change.

Generative-AI Assistance Disclosure

Earlier versions and the bounded reference work used OpenAI ChatGPT and Codex and Google Gemini for drafting, editing, literature-search support, structural and adversarial review, document preparation, and implementation assistance. Codex assisted in preparing Version 8.1 and comparing its supplied sources. AI-assisted drafting and review are not independent validation. The architecture and research direction are the author's; final review, interpretation, release decisions, and responsibility for the manuscript remain with the author.

Disclosure of Related Intellectual Property

Portions of the principal architecture are described in U.S. Patent Application No. 19/715,506, filed June 22, 2026 [1]. Portions of the controlled-return architecture are described in U.S. Provisional Patent Application

No. 64/135,982, filed August 18, 2026 [10]. The patent materials are cited for transparency and technical provenance; they are not independent technical validation. This paper makes no representation about claim scope, priority, validity, patentability, or freedom to operate.

References

- [1] M. A. Stephenson, "Supervisory Governance Architecture for Pre-Effectuation Control," U.S. Patent Application No. 19/715,506, filed June 22, 2026.
- [2] National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, January 2023, doi:10.6028/NIST.AI.100-1.
- [3] National Institute of Standards and Technology, Guide to Attribute Based Access Control (ABAC) Definition and Considerations, NIST SP 800-162, 2014, updated August 2019, doi:10.6028/NIST.SP.800-162.
- [4] OASIS, eXtensible Access Control Markup Language (XACML) Version 3.0, OASIS Standard, 2013.
- [5] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278-1308, September 1975, doi:10.1109/PROC.1975.9939.
- [6] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "Remote Attestation procedureS (RATS) Architecture," RFC 9334, Internet Engineering Task Force, January 2023, doi:10.17487/RFC9334.
- [7] H. Wang, C. M. Poskitt, and J. Sun, "AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents," arXiv:2503.18666v3, 2025, doi:10.48550/arXiv.2503.18666.
- [8] A. Yuan, Z. Su, and Y. Zhao, "AEGIS: No Tool Call Left Unchecked -- A Pre-Execution Firewall and Audit Layer for AI Agents," arXiv:2603.12621, 2026, doi:10.48550/arXiv.2603.12621.
- [9] A. Mazzocchi, "Runtime Governance for Agentic AI: Action-Boundary Control with Trusted Provenance and Fail-Closed Execution," arXiv:2608.16891, 2026, doi:10.48550/arXiv.2608.16891.
- [10] M. A. Stephenson, "Preventing Inherited Authority Through Controlled Return Governance," U.S. Provisional Patent Application No. 64/135,982, filed August 18, 2026.

[11] M. A. Stephenson, "4Gate Technology Machine Operational Architectural Specification," Version 6.9.1, review draft, September 12, 2026. Author-supplied confidential specification; access under NDA.

[12] M. A. Stephenson, "4Gate Technical Companion: Implementation guidance for MOAS version 6.9," revised September 4, 2026, sec. 10. Author-supplied technical companion.