CRYPTOGRAPHICALLY ENFORCED GOVERNANCE FOR AUTONOMOUS AGENTS AND DISTRIBUTED EXECUTION ENVIRONMENTS

RELATED APPLICATION DATA

[0001] This application claims the benefit of priority of U.S. Provisional Patent Application Serial No. 63/800,515, filed on May 6, 2025, titled “Cognition-Native Semantic Execution Platform for Distributed, Stateful, and Ethically-Constrained Agent Systems”, which is incorporated by reference herein in its entirety.

FIELD

[0002] The present disclosure relates generally to governance and control of autonomous and semi-autonomous computational agents operating across distributed execution environments. In particular, the present disclosure is directed to cryptographically enforced governance for autonomous agents and distributed execution environments, and methods thereof.

BACKGROUND

[0003] Autonomous computational agents are increasingly deployed in environments that require independent decision-making, adaptive behavior, and execution across heterogeneous computing substrates, including cloud platforms, edge devices, federated systems, and intermittently connected environments. Such agents may possess persistent memory, adaptive planning or learning capabilities, and the ability to modify internal state, delegate tasks, or propagate derivatives of themselves. As agent autonomy increases, so too does the need for mechanisms that constrain execution and mutation behavior within defined safety, security, regulatory, and organizational boundaries.

[0004] Existing approaches to governance or policy enforcement in autonomous systems frequently rely on internal reasoning, intent modeling, alignment scoring, heuristic evaluation, or outcome prediction. Such approaches attempt to infer whether contemplated behavior is acceptable based on an internal cognitive state or predicted consequences. These methods are inherently probabilistic, difficult to audit, sensitive to model accuracy and interpretability limits, and do not provide deterministic guarantees that prohibited actions cannot occur.

[0005] Other systems attempt to enforce policy through centralized controllers, trusted runtimes, or substrate-specific access controls. These approaches couple governance enforcement to particular execution environments, infrastructure providers, or centralized services. When agents

migrate across substrates, operate offline, traverse intermittent connectivity, or interact with federated systems, such controls may be bypassed, degraded, or inconsistently applied. Centralized mechanisms further introduce single points of failure, increase attack surface, and constrain scalability across distributed deployments.

[0006] Conventional policy systems also commonly embed executable rules directly within agent software or application logic. Such embedding enables an agent, or an adversary acting through the agent, to alter, disable, reinterpret, or downgrade constraints through self-modification, update mechanisms, or replication. In distributed environments, governance updates may be inconsistently propagated, enabling downgrade attacks, replay of stale authority, or reliance on expired constraints. Moreover, many systems treat refusal to execute as an error condition rather than as an intentional, enforceable outcome, thereby incentivizing workarounds that undermine governance.

[0007] Existing audit and compliance mechanisms typically operate after execution has occurred. Logs, monitoring tools, and post-hoc analysis may detect violations but do not prevent prohibited execution from taking place. In systems capable of rapid autonomous action, post-execution enforcement is insufficient to provide meaningful operational guarantees, particularly where prohibited actions cause irreversible effects.

[0008] Additionally, prior systems lack robust mechanisms for ensuring continuity of governance across agent evolution. Unauthorized forking, cloning, reconstitution, or rehydration of agents may occur without preserving governance constraints, enabling restriction shedding through replication or mutation. Policy inheritance, override authority, and escalation control are often handled informally, manually, or by non-verifiable convention, limiting reliability and auditability across distributed environments.

[0009] Accordingly, there is a need for systems and methods that address these shortcomings.

SUMMARY OF THE DISCLOSURE

[0010] A computer-implemented method for cryptographically enforced governance of an autonomous agent includes receiving, at an execution substrate, a proposed action associated with an agent object, the proposed action being associated with one or more policy references including one or more canonical aliases provided by at least one of the agent object, the execution substrate, or a governing context, resolving the one or more policy references to obtain candidate external policy

objects, filtering the candidate external policy objects based on one or more freshness constraints including at least one of a validity window, a revocation state, or an anti-rollback monotonicity constraint, verifying authenticity of at least one external policy object remaining after the filtering using cryptographic verification, determining, prior to enabling performance of the proposed action, including prior to instantiating, activating, or admitting use of an execution context or capability context, whether the proposed action is authorized under the verified external policy object, and permitting the execution substrate to enable performance of the proposed action only if authorized, otherwise deterministically denying the proposed action as a valid non-execution outcome.

[0011] A system for cryptographically enforced governance of cognition-native semantic agents includes a semantic agent object comprising semantic fields including at least an intent field, a memory field, a lineage field, and a policy field, wherein the policy field comprises one or more policy references including one or more canonical aliases, a policy resolution component configured to resolve, at runtime, the one or more policy references stored in the policy field to obtain one or more policy objects external to the semantic agent object, a verification component configured to verify authenticity and validity of the one or more policy objects using cryptographic verification, and a governance gate operatively coupled to an execution substrate and configured to, prior to enabling performance of a proposed action of the semantic agent object, including prior to instantiating, activating, or admitting use of an execution context or capability context for the proposed action, deterministically permit or deny the proposed action based on at least the verification and an authorization determination under the one or more policy objects. The proposed action comprises at least one of execution, mutation, delegation, or propagation, and wherein denial of the proposed action by the governance gate results in non-execution as a valid system outcome.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] For the purpose of illustrating the disclosure, the drawings show aspects of one or more embodiments of the disclosure. However, it should be understood that the present disclosure is not limited to the precise arrangements and instrumentalities shown in the drawings, wherein:

FIGS. 1A-1E illustrate a system-level architecture for cryptographically enforced governance in a distributed computing environment, showing one or more autonomous agents operating across heterogeneous execution substrates, each agent including externally referenced governance policy objects that are resolved and cryptographically verified as preconditions to execution and mutation;

FIG. 2 illustrates an internal structural representation of an autonomous agent relevant to governance enforcement, including a memory field, a policy reference field, a mutation descriptor field, an execution eligibility indicator, and a lineage field, wherein governance authorization is evaluated based on embedded state and externally verified policy objects rather than external orchestration;

FIG. 3 illustrates an example structure of a cryptographic policy object, including a canonical alias binding, a digitally signed policy body, scope metadata, enforcement metadata, validity constraints, and an enforcement class defining permitted and prohibited action classes;

FIG. 4 illustrates a runtime policy resolution and verification pipeline, in which a proposed execution, mutation, delegation, or propagation by an agent triggers canonical alias resolution, cryptographic verification of a referenced policy object, evaluation of freshness and applicability constraints, and a deterministic authorization or denial outcome prior to instantiation of an execution context;

FIG. 5 illustrates governance gating as a precondition to execution, showing execution eligibility being evaluated based on verified policy authorization, validity constraints, and accumulated memory state, wherein execution does not occur when governance authorization is absent;

FIG. 6 illustrates incorporation of execution feedback as enforcement signals, including latency, execution failure, congestion, or substrate refusal, and shows how such signals are recorded in agent memory and influence subsequent governance authorization decisions;

FIG. 7 illustrates lineage continuity and inheritance of governance constraints across agent mutations, showing cryptographically verifiable lineage records that constrain permissible descendant behavior and prevent unauthorized forks, cloning, or policy evasion;

FIG. 8 illustrates a quorum-based governance override mechanism, in which a plurality of authorized participants co-sign a replacement policy object, publish the override through a distributed alias system, and establish signature chain continuity for enforcement;

FIG. 9 illustrates fallback enforcement agents operating across a distributed execution substrate, monitoring governance-relevant events, validating policy integrity and override lineage, and issuing trust degradation or quarantine signals upon detection of violations or anomalous behavior;

FIG. 10 illustrates append-only governance audit and verification records, showing recording of policy resolutions, authorization denials, override approvals, trust degradation events, freshness failures, and non-execution outcomes for retrospective validation and compliance;

FIG. 11 illustrates cryptographically enforced governance across heterogeneous execution substrates, including cloud, edge, federated, and intermittently connected environments, demonstrating consistent precondition gating independent of physical infrastructure;

FIG. 12 illustrates an embodiment of governance enforcement without persistent cryptographic keypairs, integrating memory-resolved identity and trust-slope validation to authenticate policy compliance and execution eligibility based on continuity rather than static credentials; and

FIGS. 13A and 13B illustrate freshness, revocation, and anti-rollback controls for governance policy objects. FIG. 13A illustrates authorization-time evaluation of validity windows, revocation artifacts, cache revalidation, monotonicity constraints, and deterministic permit-or-deny outcomes. FIG. 13B illustrates append-only recording of freshness-related events and latest-known-good checkpoint records supporting anti-rollback enforcement.

DETAILED DESCRIPTION

1. System-Level Overview of Cryptographically Enforced Governance

[0013] The present disclosure provides systems and methods in which governance enforcement for autonomous and semi-autonomous computational objects is implemented as a deterministic cryptographic precondition to execution and other governed state transitions. Execution, mutation, delegation, or propagation is not permitted based on asserted intent, inferred alignment, predicted outcomes, or subjective reasoning quality. Such actions are permitted only when one or more externally governed policy authorities are resolved and cryptographically verified at runtime and the verified authority authorizes the proposed action under declared scope, validity, freshness, and enforcement constraints. Ethical, safety, regulatory, organizational, and operational constraints are examples of governance constraints expressible through such externally governed policy authorities.

[0014] Unlike training-based alignment, heuristic moderation, centralized orchestration, substrate-specific access controls, or retrospective monitoring—which cannot structurally prevent prohibited actions in autonomous or distributed systems—the disclosed architecture binds behavioral authority to externally maintained, cryptographically verifiable policy objects independent of the

agent object and immutable absent authorized override. Executable paths are not instantiated unless required cryptographic preconditions are satisfied. Non-execution is a valid and enforceable operational outcome.

[0015] Referring to FIGS. 1A-1E, a system-level architecture 100 for cryptographically enforced governance in a distributed computing environment is illustrated. The architecture 100 includes one or more agent objects 110 configured to operate across one or more execution substrates 190. An agent object 110 may comprise a structured data object, task object, job descriptor, mobile code object, workflow object, containerized artifact, or other machine-readable representation capable of proposing governed actions. In an embodiment, an agent object 110 is a cognition-native semantic agent comprising a plurality of independently addressable semantic fields.

[0016] In the illustrated embodiment, the agent object 110 includes an intent field 112, a context block 114, a memory field 116, a policy reference field 118, a mutation descriptor field 119, and a lineage field 120. These respectively encode semantic purpose, contextual parameters, accumulated governance-relevant history, externally referenced governance authorities, declared mutation classes, and continuity of descent. Governance enforcement operates through evaluation of the policy reference field 118 in conjunction with the memory field 116 and lineage field 120. The intent field 112 and context block 114 may inform semantics or scope selection but do not determine authorization. In embodiments, the policy reference field 118 is also referred to herein as a policy field, and the fields 112, 114, 116, 118, 119, and 120 are examples of semantic fields.

[0017] The policy reference field 118 contains one or more canonical policy aliases 130. Each canonical policy alias 130 identifies an external policy object defining enforceable constraints. Canonical policy aliases 130 are stable references and do not embed authority; they are dereferenced at runtime to obtain authoritative policy content. Multiple policy objects may apply concurrently, with precedence defined by embodiment.

[0018] When the agent object 110 proposes an action 122 resulting in execution or another governed behavior, the proposed action 122 is submitted to a governance gate 180. The governance gate 180 is a deterministic enforcement checkpoint that conditions instantiation of an execution context on cryptographic verification of applicable policy authority and on an authorization determination under the verified policy authority. The governance gate 180 may be implemented within an execution substrate 190, as middleware, as a distributed validation service, or as a logically

composed function across nodes. Authorization occurs prior to execution and prior to instantiation of an execution context.

[0019] To evaluate the proposed action 122, a policy resolution request 140 is issued to a Dynamic Alias System 150. The Dynamic Alias System 150 resolves canonical policy aliases 130 into one or more resolved policy objects 160 according to defined resolution rules, including scope-aware routing, trust-zone constraints, revocation awareness, caching policies, monotonicity or anti-rollback requirements, and resolution auditing. The output comprises resolved policy objects 160 corresponding to the referenced canonical policy aliases 130.

[0020] In embodiments, policy objects are externally maintained, immutable-by-default authority artifacts and may be expressed as structured semantic objects. Accordingly, a policy object 160 may also be referred to as a policy agent in certain embodiments, and references herein to policy objects include policy agents unless otherwise stated.

[0021] Each resolved policy object 160 includes binding to a canonical policy alias 130, a policy body defining enforceable constraints, cryptographic material attesting to authority, and metadata specifying scope, validity windows, freshness requirements, enforcement class, and related qualifiers. Each resolved policy object 160 is submitted to a verification module 170, which determines authenticity and validity under an applicable trust model and freshness constraints. Verification may use public-key cryptography or continuity-based identity mechanisms, including memory-resolved identity or trust-slope validation. The verification module 170 produces a verification result 172 indicating acceptability for enforcement.

[0022] The governance gate 180 conditions the proposed action 122 on the verification result 172 and on whether the resolved policy object 160 authorizes the proposed action 122 under declared constraints. If authorization is satisfied, the governance gate 180 permits the proposed action 122 to proceed along an execution path 184 into an execution substrate 190. If authorization is not satisfied, the governance gate 180 denies the proposed action 122 and prevents instantiation of an execution context, producing a non-execution outcome 186. Denial is a structurally valid system outcome 186 resulting from failure of resolution. Denial is a structurally valid system outcome resulting from failure of resolution, verification, freshness, validity, scope applicability, lineage continuity requirements, or authorization for the relevant action class.

[0023] The execution substrate 190 represents heterogeneous computing environments including cloud-based environments 190a, edge environments 190b, and federated or decentralized environments 190c. The execution substrate 190 does not independently grant governance authority; authority derives from verified policy objects referenced by the agent object 110, and the substrate functions as validator and enforcer of precondition gating.

[0024] The architecture 100 further includes one or more fallback enforcement agents 196 distributed across the execution substrate 190. Fallback enforcement agents 196 monitor governance-relevant events, validate policy and override continuity, detect lineage discontinuities or policy evasion, and emit enforcement signals. Upon detection of a violation or invalid state, a fallback enforcement agent 196 may issue a trust degradation signal 197 and initiate a quarantine action 198 restricting or isolating an agent object 110, execution context, or propagation pathway. Enforcement actions may be propagated across nodes to maintain distributed consistency.

[0025] An append-only audit ledger 199 records governance-relevant events including policy resolutions, verification outcomes, authorization decisions, denials, override approvals, violations, trust degradation events, quarantine actions, and freshness failures. The audit ledger 199 provides tamper-evident retrospective validation and treats non-execution outcomes as first-class system results.

[0026] The architecture 100 may further incorporate execution feedback 195 from execution substrates 190, including latency, failure, congestion, or substrate refusal indicators. Such feedback 195 may be recorded in the memory field 116 and used as input to subsequent eligibility determinations under verified policy authority. Execution feedback remains distinct from cryptographic authorization and does not convert governance enforcement into outcome prediction or intent evaluation.

[0027] Accordingly, FIGS. 1A-1E illustrate a system-level architecture 100 in which governance is enforced as a deterministic cryptographic property of system operation. Execution and other governed actions are structurally prevented unless required policy authorities are resolved, verified, and satisfied, providing enforceable guarantees suitable for autonomous, distributed, stateful, and self-modifying computational systems.

2. Agent Objects as Cryptographically Governed Semantic Objects

[0028] As described in Section 1 with respect to deterministic cryptographic precondition gating and governance gate enforcement, autonomous and semi-autonomous agent objects are implemented as structured, memory-bearing semantic objects whose authority to perform governed actions derives exclusively from externally governed, cryptographically verifiable policy references. Authority does not derive from internal intent modeling, predictive cognition, heuristic reasoning, or subjective alignment evaluation.

[0029] An agent object is not limited to a running process or session-bound execution context. It is a self-describing, machine-readable object encapsulating sufficient embedded state to support execution eligibility determination, governance evaluation, mutation control, and continuity of identity across heterogeneous execution substrates. Cognition-native semantic agents are exemplary embodiments. In such embodiments, the agent object is a cognition-native semantic agent comprising independently addressable semantic fields including at least an intent field, a memory field, a policy field, and a lineage field, wherein the policy field stores one or more canonical aliases referencing externally maintained policy objects. Because governance-relevant state is intrinsic to the object representation, the agent object remains portable across substrates without reliance on centralized orchestration or persistent session scaffolding.

[0030] Governance-relevant state embedded within the agent object includes at least: (i) a memory field recording governance-relevant history, (ii) a policy reference field identifying externally maintained policy authorities, and (iii) in evolutionary embodiments, mutation descriptors and lineage information sufficient to evaluate continuity and eligibility across state transitions.

[0031] The memory field may record prior execution events, policy resolutions, verification outcomes, denials, trust degradation events, override applications, freshness failures, quarantine states, and related governance transitions. The policy reference field contains canonical identifiers resolvable to externally governed policy objects authenticated under a cryptographic or continuity-based trust model. Mutation descriptors and lineage information enable continuity validation and inheritance evaluation. By carrying these elements intrinsically, the agent object provides a receiving substrate sufficient information to determine eligibility at evaluation time independent of substrate-local control logic.

[0032] Consistent with the enforcement model discussed in Section 1 above, proposed actions are not authorized based on asserted intent or predicted outcomes. Although an agent object may

include semantic descriptors or intent fields expressing objectives, such fields are not determinative of execution eligibility. Behavioral authority derives from runtime resolution and verification of externally governed policy references embedded within the agent object. Because governing authority is external and immutable absent authorized override, the agent object cannot unilaterally modify, reinterpret, or bypass imposed constraints.

[0033] Eligibility may depend on embedded historical state. Verified policy objects may condition execution, mutation, delegation, or propagation authority on prior compliance history, accumulated trust state, remediation completion, absence of unresolved violations, freshness compliance, or other policy-defined criteria reflected in the memory field. Eligibility therefore may derive from objective embedded governance history rather than centralized scheduling or substrate convention.

[0034] A receiving execution substrate evaluates embedded state, resolves and verifies referenced policy authority under applicable validity and freshness constraints, and deterministically permits or denies instantiation of an execution context. Where required conditions are unsatisfied, non-execution is returned as a valid system outcome consistent with deterministic precondition gating, rather than relying on post-hoc remediation.

[0035] Governance enforcement remains independent of predictive modeling, harm forecasting, or interpretive reasoning. Verified policy authority specifies permitted and prohibited action classes under declared scope and validity constraints. Ethical, safety, regulatory, organizational, and operational constraints are examples of governance constraints expressible in such policy authority. Where authorization is absent, execution is denied regardless of predicted benefit; where authorization is present under verified authority, execution may proceed notwithstanding internal uncertainty.

[0036] Because governance is externalized and verification-based, enforcement remains deterministic across distributed systems characterized by partial information, intermittent connectivity, adversarial conditions, or heterogeneous implementations. The architecture supports narrow-task agents, minimal-cognition agents, degraded or partially instantiated objects, and heterogeneous agent implementations, provided required policy references and governance-relevant state are present and verifiable.

3. Canonical Agent Fields Relevant to Cryptographically Enforced Governance

[0037] Governance eligibility is determined by evaluating embedded state together with resolved and verified external policy objects. The canonical fields described below enable denial of instantiation of an execution context where required authority or continuity conditions are not satisfied.

[0038] Referring now to FIG. 2, an internal structural representation of an agent object 200 is illustrated, emphasizing canonical governance-relevant fields and their interaction with externally resolved authority. The agent object 200 is a structured, machine-readable object comprising independently addressable internal fields sufficient to support governance evaluation by an execution substrate, governance gate, or validation layer.

[0039] The agent object 200 includes a memory field 210. The memory field 210 is a persistent, append-capable record intrinsic to the agent object 200 and stores governance-relevant history, including prior execution attempts, policy resolutions, verification outcomes, denials, trust degradation events, quarantine states, override records, remediation events, and governance-designated substrate feedback. The memory field 210 is evaluable as an input to future eligibility determinations, such that unresolved violations, accumulated denials, enforcement escalations, or policy-defined triggers may restrict execution, mutation, delegation, or propagation. Because the memory field 210 is intrinsic to the agent object 200, governance-relevant state persists across heterogeneous execution substrates.

[0040] The agent object 200 further includes a policy reference field 220 containing one or more canonical references to externally governed policy authorities. In FIG. 2, such references are illustrated as canonical policy aliases 260. The policy reference field 220 does not encode authoritative policy content and confers no authority by mere presence. Each canonical policy alias 260 must be resolved by a policy resolution system 270 to obtain a resolved policy object 280. The resolved policy object 280 is verified to produce verified policy authority 290 prior to permitting a governed action. Authorization depends on successful resolution, verification, and applicability of the resolved policy object 280 under declared scope, validity, and freshness constraints.

[0041] The agent object 200 also includes a mutation descriptor field 230 declaring one or more mutation classes, transformation parameters, delegation characteristics, propagation constraints, or descendant inheritance conditions. The mutation descriptor field 230 is evaluated in conjunction with verified policy authority 290 to determine whether a proposed mutation, delegation, or

propagation is permitted. The mutation descriptor field 230 constitutes a declaration of potential transformation behavior and remains subordinate to externally verified policy constraints embodied in verified policy authority 290.

[0042] The agent object 200 further includes a lineage field 240 recording continuity information describing ancestry and evolution, including prior states, parent identifiers, mutation events, inheritance records, and authorized override lineage where applicable. The lineage field 240 enables cryptographic or continuity-based verification that a current state is a valid successor of a previously authorized state. Governance enforcement may deny execution or propagation when lineage continuity is invalid, discontinuous, or inconsistent with applicable verified policy authority 290.

[0043] The agent object 200 further includes an execution eligibility indicator 250 representing a derived or evaluable state indicating whether instantiation of an execution context is permitted for one or more governed action classes. In some embodiments, the execution eligibility indicator 250 is computed dynamically from evaluation of the policy reference field 220, memory field 210, mutation descriptor field 230, lineage field 240, and verified policy authority 290. In other embodiments, the execution eligibility indicator 250 is stored and updated upon governance-relevant events, including verification success, denial, quarantine, trust degradation, revocation, or freshness transitions. Regardless of implementation, the execution eligibility indicator 250 enables a substrate to determine execution permissibility without centralized session state.

[0044] Where the execution eligibility indicator 250 reflects satisfaction of required conditions, instantiation of an execution context may proceed, resulting in an execution context instantiation outcome 295. Where required governance conditions are not satisfied, instantiation of the execution context is prevented, and the execution context instantiation outcome 295 reflects denial as a valid system result.

[0045] Although the agent object 200 may include additional fields such as intent, context, planning structures, affective state, capability declarations, or application payloads, FIG. 2 demonstrates that cryptographically enforced governance relies on structural evaluation of canonical fields 210, 220, 230, 240, and 250 together with externally verified policy authority 290 produced by the policy resolution system 270 from canonical policy aliases 260 via resolved policy objects 280. Any substrate capable of parsing the agent object 200, resolving referenced authority through policy

resolution system 270, and evaluating these canonical fields together with verified policy authority 290 may enforce governance deterministically.

4. Cryptographic Policy Objects as Immutable Governance Authorities

[0046] As previously described with respect to deterministic precondition gating and canonical policy reference resolution, governance is enforced through cryptographic policy objects that function as externally governed, first-class authorities. Ethical, safety, regulatory, organizational, architectural, and operational constraints are examples of governance domains expressible through such objects. A policy object is not advisory configuration or heuristic guidance; it is authenticated authority whose constraints are enforced as preconditions to instantiation of execution contexts and other governed state transitions. In some embodiments, a policy object is expressed as a structured semantic object and may be referred to herein as a policy agent. References to policy objects in this disclosure therefore include policy agents unless otherwise stated.

[0047] Policy objects are structurally external to governed agent objects. They are not embedded as mutable executable logic and are not subject to reinterpretation by internal reasoning. Authority derives solely from runtime resolution and verification under an applicable trust model. Because policy authority is external and immutable absent authorized succession, agent-local mutation, replication, or reconfiguration cannot weaken or silently alter constraints.

[0048] Authenticated policy content is immutable by default in preferred embodiments. Governance changes occur through issuance of a successor or override policy object rather than in-place modification. Immutability may be enforced through content-addressed storage, hash binding, signature binding, continuity-based validation, or combinations thereof, thereby supporting auditability and resistance to downgrade, replay, and silent modification.

[0049] Referring now to FIG. 3, a structural representation of a cryptographic policy object 300 is illustrated. The policy object 300 is a standalone, machine-readable governance authority independently storable, transmissible, resolvable, cacheable under policy, and verifiable without reliance on any particular agent object.

[0050] The policy object 300 includes a canonical alias binding 310. The canonical alias binding 310 provides a stable identifier through which governed agent objects reference the policy

object 300. This indirection enables authorized supersession by publishing a successor under the same canonical alias without modifying governed agent objects.

[0051] The policy object 300 further includes a policy body 320 encoding deterministic, machine-interpretable constraints defining permitted and prohibited behavior classes. Such constraints may include execution restrictions, self-modification limits, delegation or propagation controls, memory-access limitations, lineage-forking rules, escalation boundaries, quarantine conditions, remediation prerequisites, or combinations thereof. The policy body 320 is evaluated prior to instantiation of execution contexts or authorization of other governed actions.

[0052] A verification field 330 contains authentication material bound to at least the canonical alias binding 310 and the policy body 320. The verification field 330 establishes authenticity and integrity under an applicable trust model. In some embodiments, the verification field 330 comprises a public-key digital signature. In other embodiments, it comprises continuity-based authentication material validated through memory-resolved identity, trust-slope validation, or lineage continuity mechanisms, enabling authority establishment without persistent static keypairs. In all embodiments, the verification field 330 provides objective confirmation of authorized origin and non-alteration.

[0053] The policy object 300 additionally includes a scope declaration 340 defining applicability, including applicable agent classes, action classes, execution substrate classes, trust zones, semantic roles, lineage classes, or other bounded operational domains. Explicit scope enables deterministic applicability evaluation without heuristic inference.

[0054] A validity and freshness component 350 defines temporal and state-based authority bounds, including activation times, expiration times, time-to-live values, revocation epochs, monotonic version indicators, anti-rollback commitments, or combinations thereof. This component enables rejection of expired, revoked, superseded, or stale authority, including under caching and intermittent connectivity conditions.

[0055] The policy object 300 further includes an enforcement class field 360 specifying treatment of evaluation outcomes, including hard denial of execution context instantiation, trust degradation, quarantine, escalation to fallback enforcement agents, remediation requirements, audit-only recording, or combinations thereof. Encoding enforcement semantics within the policy object 300 ensures consistent cross-substrate treatment.

[0056] In operation, a governed agent object referencing the policy object 300 may not instantiate an execution context or perform another governed action unless the policy object 300 is resolved, verified, and determined to authorize the proposed action. Resolution yields the complete policy object 300 corresponding to the canonical alias. Verification confirms authenticity, integrity, and satisfaction of scope, validity, and freshness constraints defined by the scope declaration 340 and validity and freshness component 350. Authorization is determined by evaluating the proposed action against the policy body 320 and applying the enforcement class field 360 to produce a deterministic permit-or-deny outcome.

[0057] Because authenticated content is immutable, governance evolution occurs through successor or override policy objects rather than mutation. This structure supports layered governance, verifiable audit trails, and resistance to erosion of governance guarantees. Accordingly, FIG. 3 illustrates cryptographic policy objects 300 as externally governed, immutable authorities that externalize governance from agent objects and enforce constraints as a deterministic, verifiable property of execution eligibility and precondition gating.

5. Canonical Alias Resolution and Policy Reference Binding

[0058] A canonical alias is a stable identifier referring to a policy object without embedding policy content. The alias functions strictly as a reference. Agent objects include one or more canonical aliases in a policy reference field and do not embed mutable policy logic internally. Governance authority therefore remains external, preventing constraint weakening through local mutation, replication, serialization, or packaging.

[0059] Canonical alias resolution maps a canonical alias to a complete policy object via a Dynamic Alias System, scoped registry, adaptive index, distributed naming system, or equivalent resolution substrate capable of returning policy content together with provenance sufficient for verification. Resolution may incorporate scope-aware routing, trust-zone enforcement, revocation awareness, freshness constraints, caching rules, locality preferences, and audit controls. Where a proposed action requires policy authority, resolution must yield a verifiable and applicable policy object; otherwise instantiation of an execution context is denied as a valid non-execution outcome.

[0060] Policy reference binding occurs when a resolved policy object is evaluated against the agent object's policy reference field to establish enforceable authority for a proposed action. Binding requires: (i) authenticity verification under an applicable trust model, (ii) satisfaction of declared

scope constraints, (iii) compliance with validity and freshness requirements, and (iv) authorization of the proposed action class under the policy body. Identifier equivalence alone is insufficient. Failure of any condition renders the reference non-authoritative for that action.

[0061] Alias indirection enables governance evolution without mutating agent objects. A policy may be superseded by issuing a successor policy object under the same canonical alias through an authorized publication procedure. Agent objects referencing the alias thereby become governed by the successor authority without modification, supporting long-lived, mobile, replicated, or intermittently connected objects evaluated across heterogeneous substrates.

[0062] Revocation and supersession are enforced through resolution and binding semantics. Authority may expire, be explicitly revoked, or be replaced through authorized override. Revocation status may be expressed within a policy object's validity and freshness component or through external revocation artifacts, alias redirection, quorum-issued overrides, append-only audit records, or combinations thereof. A revoked, expired, stale, or superseded policy object is treated as non-authoritative, and required actions are denied.

[0063] The architecture resists substitution, downgrade, replay, and stale-authority reliance. Because binding requires verification and applicability evaluation, a substituted or weaker policy object fails authentication or scope, validity, or freshness checks. Downgrade attacks are mitigated through validity-window enforcement, revocation awareness, monotonic versioning, signature-chain continuity, anti-rollback controls, and rejection of non-current authoritative instances. Temporal validity constraints, cache revalidation rules, and, in some embodiments, memory-recorded authority checkpoints further prevent replay or indefinite reuse of outdated authority. Repeated attempts to rely on stale authority may trigger denial, trust degradation, or quarantine under policy-defined enforcement treatment.

[0064] Canonical alias resolution also supports layered governance. An agent object may reference multiple canonical aliases corresponding to distinct authorities or domains. For a given proposed action, a defined subset of policy objects must jointly authorize the action. Each required policy object must be resolved, verified, and applicable. Failure of any required binding prevents instantiation of an execution context. Independent governance authorities may therefore constrain behavior without embedding policy logic in the agent object or requiring coordination at creation time.

[0065] Accordingly, canonical alias resolution and policy reference binding attach externally governed, verifiable authority to computational objects through indirection, resolution, and deterministic binding rather than embedded mutable rules. This structure supports upgradeable and revocable governance, resistance to substitution, downgrade, replay, and stale-authority attacks, and consistent precondition gating across autonomous, distributed, and heterogeneous execution environments.

6. Runtime Policy Resolution and Verification Pipeline

[0066] Governed actions are permitted only if required externally governed policy objects are resolved, verified, applicable, and authorizing at runtime. This section defines the deterministic, pre-execution pipeline by which such evaluation occurs. The pipeline is mandatory for governed action classes and must complete successfully for all required policy objects prior to instantiation of an execution context. It evaluates authority, not intent, prediction, or remediation.

[0067] The pipeline is invoked when an agent object proposes a governed action, including execution, mutation, delegation, propagation, governance-designated memory modification, or lineage-affecting operations. Stages may be synchronous or distributed, but authorization is logically completed before instantiation.

[0068] Referring now to FIG. 4, a runtime policy resolution and verification pipeline 400 is illustrated. The pipeline 400 begins when an agent object 410 presents a proposed action 420. The proposed action 420 is a declarative request identifying a governed action class, including execution type, mutation scope, delegation target, propagation destination, or other governance-relevant operation.

[0069] A governance gate 430 receives the proposed action 420 and extracts one or more canonical policy aliases 440 from the agent object 410. The governance gate 430 determines which aliases are required for the action class and derives authority exclusively from verified policy objects.

[0070] The governance gate 430 issues a policy resolution request 450 to a policy resolution subsystem 460. The policy resolution subsystem 460 may comprise a Dynamic Alias System, scoped registry, adaptive index, distributed naming system, or equivalent authoritative resolver. The request 450 includes the canonical policy aliases 440 and may include scope context, trust zone, substrate

class, locality constraints, caching directives, or revocation-awareness parameters. Failure to resolve a required alias results in pipeline failure and denial of the proposed action 420.

[0071] The policy resolution subsystem 460 returns one or more resolved policy objects 470. In embodiments, the policy resolution subsystem returns a candidate set and filters the candidate set based on validity-window satisfaction, revocation state, and anti-rollback constraints prior to verification. Each resolved policy object 470 includes a policy body, scope declaration, validity and freshness information, enforcement class, and verification material (as described with respect to FIG. 3). Where multiple candidates exist, deterministic selection rules apply, including rejection of revoked or expired instances, enforcement of anti-rollback and monotonicity requirements, preference for quorum-approved overrides, and application of trust-zone precedence. Freshness and anti-rollback controls are further described with reference to FIG. 13.

[0072] Each resolved policy object 470 is submitted to a verification stage 480. The verification stage 480 validates authenticity and integrity under the applicable trust model, including public-key signature verification or continuity-based mechanisms such as memory-resolved identity, trust-slope validation, or lineage continuity. The verification stage 480 produces a verification result 490 indicating whether the policy object 470 is authentic and attributable to authorized authority.

[0073] The governance gate 430 also performs applicability evaluation for each resolved policy object 470, including scope compliance, validity and freshness satisfaction, and enforcement-class interpretation. An authentic but inapplicable, expired, revoked, superseded, or non-authorizing policy object is treated as non-authoritative.

[0074] Verification results 490 and applicability determinations are aggregated to produce an authorization decision 500. The authorization decision 500 is affirmative only if each required policy object is resolved, verified, applicable, and authorizing for the proposed action 420. Failure of any required condition produces a negative decision. In embodiments, verification includes evaluation of validity-window satisfaction and freshness eligibility, or produces a result that is combined with validity and freshness evaluation to determine acceptability for enforcement. Applicability includes evaluation of action class, substrate class, trust zone, and lineage class where specified by scope metadata.

[0075] If the authorization decision 500 is affirmative, the governance gate 430 emits an authorization permit 510, allowing the proposed action 420 to proceed along an execution path 520,

including instantiation of an execution context, mutation, delegation, or propagation as applicable. The governance gate 430 does not modify the action or provide substitute authority.

[0076] If the authorization decision 500 is negative, the governance gate 430 emits a denial outcome 530. The denial outcome 530 prevents instantiation of an execution context, and the governed action does not occur. Denial is a valid system outcome. Policy-defined secondary effects may follow, including audit recording, memory updates, trust degradation, or quarantine evaluation, while preserving non-execution. In embodiments, the denial outcome 530 is emitted as a structured denial result configured to be stored in the memory field of the agent object and/or recorded in an append-only audit record.

[0077] The pipeline 400 is not bypassable by agent-local logic. Execution substrates do not independently grant authority or convert pipeline failure into fallback execution. Absence of authorization results in non-instantiation. Because the pipeline 400 operates on externally governed policy objects and objective verification, it applies uniformly across heterogeneous substrates and agent implementations, including minimal or partially instantiated objects, provided required policy references can be resolved and verified under applicable constraints.

7. Governance Gating as a Precondition to Execution

[0078] Execution is a governed action. Instantiation of an execution context is permitted only if the runtime policy resolution and verification pipeline (Section 6) succeeds and verified policy authority authorizes the proposed execution. Execution is therefore a structural preconditioned privilege rather than a default substrate function. Ethical constraints are an example enforceable through the same gating.

[0079] No execution context is created unless authorization is satisfied beforehand. Where authorization is absent, no execution instance is instantiated, no partial or speculative execution occurs, and non-execution constitutes a valid system outcome that may be recorded or used to update governance state without permitting the prohibited execution to occur.

[0080] Referring now to FIG. 5, a governance execution gating flow 580 is illustrated. FIG. 5 illustrates the execution instantiation consequences of the authorization determination produced in FIG. 4. An agent object 501 presents a proposed execution request 502, which is evaluated by governance verification stage 503 to produce an execution authorization determination 540. The

authorization permit 510 and denial outcome 530 correspond to the permit/deny outcomes of FIG. 4, and the execution path 520 represents the permitted branch leading to instantiation of execution instance 560. The flow 580 begins with an agent object 501 presenting a proposed execution request 502 to instantiate an execution context on a substrate. The proposed execution request 502 is treated as a governed action.

[0081] The proposed execution request 502 is submitted to a governance verification stage 503. The governance verification stage 503 performs policy resolution, verification under an applicable trust model, scope evaluation, validity and freshness evaluation, revocation checks, and authorization determination consistent with Section 6. This stage occurs prior to instantiation of any execution context.

[0082] The governance verification stage 503 produces an execution authorization determination 540 indicating whether instantiation of an execution context for the proposed execution request 502 is permitted under verified policy authority.

[0083] If the execution authorization determination 540 is affirmative, an execution permit 550 is issued. The execution permit 550 enables creation of an execution instance 560 on an execution substrate 570. The execution instance 560 does not exist prior to issuance of the execution permit 550. Execution authority is thus derived from verified governance authority.

[0084] If the execution authorization determination 540 is negative, an execution denial 580 is issued. The execution denial 580 prevents creation of any execution instance 560. No execution context is instantiated and no partial or speculative execution occurs. In some embodiments, the execution denial 530 triggers policy-defined secondary effects, including audit logging, trust degradation, remediation requirements, or quarantine evaluation, while preserving non-execution 535 of the proposed execution request 520.

[0085] The execution substrate 570—whether cloud, edge, federated, decentralized, or other compute environment—acts as a validator and executor of authorization rather than as an independent source of authority. The execution substrate 570 instantiates execution only upon receipt or verification of an execution permit 550 and does not grant fallback execution in its absence. Accordingly, FIG. 5 illustrates governance gating as a precondition to execution: instantiation occurs only upon successful governance verification, and non-execution is an enforceable and intended system result.

8. Governance Gating as a Precondition to Mutation and Propagation

[0086] Mutation and propagation are governed state transitions subject to the same pre-execution resolution, verification, and authorization pipeline described in Section 6. Because such transitions may alter authority, scope, eligibility, lineage, or cross-environment effects, they are permitted only when applicable verified policy authority authorizes the proposed transition at the time it is proposed. Ethical constraints are an example case enforceable through the same mechanisms.

[0087] Mutation comprises any transformation of an agent object's governance-relevant state, including modification of memory fields, mutation descriptors, policy references, execution parameters, internal semantic structure, or creation of derivative objects. A mutation descriptor may declare the requested transformation class, but authority derives solely from resolved and verified policy objects. Prior to instantiation, the proposed mutation is evaluated to determine whether it: (1) falls within an authorized mutation class, (2) preserves required invariants, (3) satisfies lineage continuity requirements, and (4) meets applicable scope, validity, freshness, override, or approval constraints.

[0088] If any required condition fails, the mutation is denied. No partial mutation, speculative application, or rollback-based correction occurs; the unauthorized state transition is not instantiated. Non-mutation is a valid system outcome.

[0089] Propagation includes replication, delegation, spawning, transfer across trust domains, migration, rehydration, or other introduction of an agent object into another substrate or authoritative context. Each propagation attempt is evaluated as a governed action. If verified policy authority does not authorize the propagation under declared constraints, the propagation is denied and the object remains confined to its authorized state or is subjected to policy-defined enforcement treatment. No unverified descendant or cross-domain instance is instantiated.

[0090] Gating of mutation and propagation preserves governance continuity across lifecycle and environment. Lineage and inheritance rules are evaluated to prevent unauthorized forks and to ensure descendants remain bound to required policy authority. An agent object cannot evade governance by altering policy references, spawning unconstrained derivatives, or migrating to a less restrictive substrate, because each such act independently requires verified authorization.

[0091] Enforcement is independent of internal cognition or asserted intent. The system evaluates only whether verified external authority authorizes the mutation or propagation class under applicable constraints. If authorized, the transition may proceed; if not, instantiation is prevented.

[0092] In some embodiments, denial triggers policy-defined secondary effects (e.g., audit recording, trust degradation, remediation requirements, quarantine evaluation, or escalation to fallback enforcement), while preserving the non-occurrence of the governed transition.

[0093] Accordingly, mutation and propagation—like execution—are conditioned on verified external policy authority. Governance constraints therefore persist across time, substrates, and lineage without reliance on post hoc correction or interpretive enforcement.

9. Memory-Derived Eligibility for Governed Actions

[0094] Eligibility to instantiate an execution context or perform other governed actions may depend on embedded memory state in addition to contemporaneous policy resolution and verification. As described with respect to canonical memory fields and runtime gating, eligibility is determined at the time a governed action is proposed by evaluating persisted governance-relevant history together with verified external policy authority. Ethical compliance is an example case enforceable under the same mechanisms.

[0095] The memory field is a persistent, append-capable component intrinsic to the agent object and portable across substrates. It may record governance-relevant events including prior authorization permits, denials, policy resolution outcomes, freshness failures, revocations, override applications, trust degradation events, quarantine states, remediation acknowledgments, and policy-designated substrate feedback. This recorded history constitutes objective input to eligibility evaluation.

[0096] Memory-derived eligibility may render a governed action not permitted where memory reflects unresolved violations, unremediated denials, quarantine state, elevated enforcement class, or other policy-defined disqualifying conditions. For example, where a prior denial required remediation and no qualifying remediation record is present, eligibility for the same or related action classes remains not permitted. Where quarantine or restriction state is recorded, eligibility remains constrained until expiration, satisfaction of recorded conditions, or authorized override under verified policy authority.

[0097] In some embodiments, memory records accumulated trust or enforcement state derived from objective event types, counters, epochs, or markers. Sustained compliant outcomes may preserve broader eligibility, whereas repeated denials, freshness failures, or policy-defined adverse triggers may narrow eligibility thresholds, elevate enforcement class, or require additional corroboration. Evaluation remains deterministic and based on applying verified policy criteria to recorded memory state, without reliance on inferred intent or predictive modeling.

[0098] Eligibility evaluation is performed at authorization time by inspecting the agent object's embedded memory in conjunction with resolved and verified policy authority applicable to the proposed action class. No centralized scheduler or external permission service is required. Eligibility may be computed dynamically or reflected in stored eligibility markers updated upon prior governance events, subject to re-evaluation under current verified authority.

[0099] Because memory state travels with the agent object, eligibility remains portable across heterogeneous and intermittently connected substrates. An agent object denied on one substrate due to embedded disqualifying history remains ineligible elsewhere unless conditions recorded in memory are satisfied under applicable policy.

[0100] Temporal constraints may also be memory-derived. Cooldown intervals, decay windows, periodic re-verification checkpoints, or similar policy-defined temporal conditions may be evaluated deterministically using timestamps, counters, or epochs recorded in memory.

[0101] Accordingly, memory-derived eligibility conditions governed actions on demonstrated historical compliance recorded within the agent object and evaluated together with verified external policy authority. Execution and other governed transitions are permitted only when both present authorization and memory-qualified eligibility criteria are satisfied; otherwise, non-execution results as a valid system outcome.

10. Governance Evaluation Independent of Intent Modeling and Outcome Prediction

[0102] Governance enforcement evaluates whether a proposed action is authorized under resolved and verified external policy authority, without reliance on intent modeling, internal reasoning inspection, alignment scoring, or outcome prediction. As described with respect to runtime resolution and precondition gating, authorization is determined solely by applying verified policy

objects to the declared action class under applicable scope, validity, freshness, and continuity constraints. Ethical constraints are an example case enforced through the same mechanism.

[0103] Internal representations—such as intent, goals, preferences, plans, explanations, confidence metrics, or predictive assessments—may exist within an agent object but are not determinative of authorization. The governance gate does not evaluate why an action is proposed or whether predicted consequences appear beneficial or harmful. It determines only whether required policy authority is present, authentic, applicable, and authorizing at the time of evaluation.

[0104] Accordingly, a proposed action is denied when required authority is absent, unresolved, unverifiable, expired, revoked, superseded under anti-rollback constraints, or inapplicable under declared scope, regardless of internal reasoning suggesting benign intent. Conversely, a proposed action that satisfies verified policy authority may proceed notwithstanding uncertainty, low internal confidence, or incomplete predictive information, because enforcement is grounded in objective authority rather than probabilistic assessment.

[0105] This separation of authorization from cognitive evaluation enables consistent enforcement across heterogeneous agent implementations, including opaque, minimal, degraded, or proprietary systems. Enforcement components need not access or trust internal model state; they require only the agent object's governance-relevant fields and resolvable external policy authority. This reduces susceptibility to manipulation of internal representations and supports substrate-independent enforcement boundaries.

[0106] In distributed or adversarial environments where predictive accuracy may be limited or context may change dynamically, authority-based gating provides structural guarantees that prohibited action classes are not instantiated. Governance thus operates as an objective, verifiable precondition to execution and other governed transitions, independent of interpretive reasoning or outcome forecasting.

[0107] 11. Incorporation of Execution Feedback as Governance-Relevant Enforcement Signals

[0108] Execution feedback may be incorporated as governance-relevant input where designated by verified policy authority, without converting governance into outcome prediction, intent analysis, or reactive moderation. Execution feedback comprises signals generated by execution substrates before, during, or in response to a governed action, including latency, timeout, refusal to instantiate,

congestion, resource exhaustion, safety interlock activation, deferral, degradation, or termination conditions. Ethical constraints are an example class of governance constraints that may incorporate such feedback under policy-defined criteria.

[0109] Execution feedback does not override policy authority, retroactively legitimize unauthorized actions, or authorize prohibited action classes. Instead, when policy designates feedback as governance-relevant, such signals are recorded as objective memory state and evaluated prospectively during subsequent authorization determinations.

[0110] Execution substrates may emit quantitative or qualitative feedback for permitted, denied, deferred, refused, partially completed, or terminated actions. When policy designates such signals as governance-relevant, they are treated as first-class enforcement inputs rather than transient telemetry and are persisted in the agent object's memory for future eligibility evaluation.

[0111] Referring now to FIG. 6, a feedback-informed governance evaluation flow 600 is illustrated. The flow 600 preserves the requirement that authorization precedes instantiation of any execution context.

[0112] The flow 600 begins with an agent object 610 that initiates a governed action attempt 615. The governed action attempt 615 may correspond to an execution, mutation, delegation, propagation, or other governance-relevant transition previously authorized or evaluated under deterministic precondition gating.

[0113] The agent object 610 interacts with an execution substrate 620, which may comprise a cloud, edge, federated, decentralized, or constrained environment.

[0114] During or in response to the governed action attempt 615, the execution substrate 620 produces execution feedback 630. The execution feedback 630 may correspond to latency, failure, congestion, deferral, refusal, degradation, partial execution, or other operational conditions, whether execution succeeds or is denied or terminated.

[0115] The execution feedback 630 is recorded into an agent memory field as a feedback record 640. The feedback record 640 may include timestamps or epochs, substrate identifiers, trust-zone identifiers, action classes, feedback categories, severity indicators, and policy-relevant annotations. Recording the feedback record 640 preserves execution conditions across substrates and time.

[0116] A governance evaluation function 650 consumes the feedback record 640 in future eligibility determinations. The governance evaluation function 650 applies policy-defined rules to determine whether eligibility should be restricted, throttled, escalated, deferred, conditioned on additional verification, or otherwise modified.

[0117] Based on this evaluation, the governance evaluation function 650 produces a governance state outcome 660. The governance state outcome 660 may include modifications to execution eligibility indicators, trust or enforcement markers, enforcement class, quarantine eligibility, additional verification requirements, or routing constraints. The governance state outcome 660 is recorded and influences subsequent proposals to execute, mutate, delegate, or propagate.

[0118] When a subsequent governed action is proposed, a future authorization determination 670 is performed. The future authorization determination 670 evaluates the governance state outcome 660 together with verified policy authority 680. The verified policy authority 680 corresponds to externally resolved and authenticated policy objects applicable to the proposed action class.

[0119] Execution feedback therefore influences authorization prospectively, without altering past authorization outcomes or permitting actions prohibited by verified policy authority 680.

[0120] Execution feedback cannot supersede verified policy constraints, negate validity or freshness failures, or authorize otherwise disallowed action classes. It operates strictly within policy-defined boundaries, preserving deterministic precondition gating and enforceable non-execution outcomes.

12. Trust Degradation, Quarantine, Rollback, and Non-Execution Outcomes

[0121] Trust degradation, quarantine, rollback, and execution refusal are deterministic governance outcomes produced when authorization, continuity, freshness, revocation, scope, or memory-derived eligibility conditions are not satisfied, or when policy-defined enforcement treatment requires restriction. Ethical governance is an example case enforced through the same mechanisms.

[0122] When required policy authority is absent, unverifiable, inapplicable, expired, revoked, superseded under anti-rollback constraints, or disqualified by memory-derived eligibility, instantiation of an execution context or other governed transition is prevented. No partial execution

or compensatory reinterpretation occurs. The resulting non-execution is recorded as a governance-relevant event.

[0123] Trust degradation is a policy-defined state transition adjusting effective eligibility or enforcement class based on objectively recorded events in embedded memory. Such events may include repeated denials, freshness failures, stale references, unresolved remediation, lineage anomalies, or governance-designated execution feedback. Trust degradation is computed deterministically from recorded event types, counters, epochs, or state markers and may narrow permitted action classes, restrict mutation or propagation scope, require additional verification, or elevate enforcement class.

[0124] Quarantine is a structural restriction preventing instantiation of execution contexts and/or other governed transitions for one or more action classes. Quarantine may be triggered by severe or repeated violations, invalid lineage continuity, unauthorized mutation or propagation attempts, unresolved forks, or enforcement class escalation. While quarantined, the agent object may remain accessible for inspection, audit, or remediation actions permitted by policy but cannot perform prohibited governed actions. Quarantine persists until lifted by authorized policy, expiration of a policy-defined interval, or successful remediation recorded and verified.

[0125] Rollback reverts an agent object to a prior authorized state upon detection of unauthorized or partially authorized mutation, lineage discontinuity, improper reliance on stale or revoked authority, or other policy-defined restoration triggers. Rollback may use lineage records, checkpoints, or memory-stored state references. It restores a previously authorized state while preserving records of the triggering event and does not legitimize the unauthorized condition.

[0126] Execution refusal is a specific non-execution outcome arising when a proposed execution fails authorization due to failed resolution, failed verification, scope inapplicability, freshness failure, revocation, anti-rollback violation, or unmet memory-derived eligibility. No execution context is instantiated and no partial execution occurs. The refusal is recorded in embedded memory and/or an append-only audit record for subsequent evaluation.

[0127] Policy objects may define interactions among these outcomes. For example, repeated refusals or freshness failures may trigger trust degradation; trust degradation may elevate enforcement class; elevated enforcement class may result in quarantine; and rollback may be

accompanied by degradation or temporary suspension. Such relationships are defined by externally governed policy authority rather than hard-coded logic.

[0128] All such outcomes are enforced prior to instantiation of execution contexts or completion of governed transitions. The architecture does not rely on rollback as a primary safeguard after prohibited execution; instead, it prevents instantiation where authorization is absent.

[0129] Because trust degradation, quarantine, rollback, and execution refusal are recorded as durable governance state in embedded memory and lineage records, restrictions persist across substrates and environments. An agent object restricted in one environment remains restricted upon migration unless eligibility is restored under verified policy authority.

13. Lineage as a Verifiable Continuity Mechanism for Governed Evolution

[0130] Lineage operates as an enforceable continuity constraint linking a current agent-object state to one or more prior authorized states. Eligibility to execute, mutate, delegate, propagate, migrate, or reconstitute depends not only on contemporaneous policy resolution and verification, but also on verification that the present state is a valid successor under applicable continuity rules. Where continuity cannot be established, instantiation of an execution context or other governed transition is denied as a valid non-execution outcome.

[0131] Lineage is embodied as a protected continuity record embedded within the agent object. The record may include identifiers of ancestor states, hashes or digests of prior states, mutation or transition events, timestamps or epochs, policy inheritance markers, checkpoints, and attestations associated with authorized transitions. These elements permit objective verification that evolutionary transitions occurred under valid governance conditions and enable detection of discontinuities, unresolved forks, replayed states, or untrusted reconstitution events.

[0132] Continuity may be validated using cryptographic chaining (e.g., hash-linked states), authenticated transition records (e.g., signatures, co-signatures, quorum attestations), or continuity-based mechanisms such as memory-resolved identity or trust-slope validation that do not require persistent static keypairs. Regardless of mechanism, validation deterministically establishes whether the current state is legitimately derived under the applicable trust model.

[0133] If a proposed governed action originates from a state lacking a valid lineage link to an authorized predecessor, the action is denied even if contemporaneous policy references are

resolvable and verifiable. Unauthorized forks, cloning, replay of prior snapshots, illicit propagation, or reconstructed states lacking authorized continuity are thereby rendered ineligible for execution or other governed transitions. Where the lineage record indicates multiple competing branches without an authorized merge or fork authorization record, the condition constitutes an unresolved lineage fork and the proposed action is denied.

[0134] Lineage also enforces inheritance of governance constraints across evolution. Policy references, enforcement classes, eligibility or trust markers, quarantine state, remediation requirements, and related governance attributes may be inherited or conditioned by lineage. Descendant objects may therefore remain subject to restrictions imposed on ancestors unless modified through authorized policy mechanisms.

[0135] Accordingly, lineage provides a deterministic, verifiable continuity mechanism preserving governance integrity across mutation, propagation, and reconstitution. By conditioning eligibility on validated continuity, governance constraints—including ethical constraints as an example case—cannot be bypassed through unauthorized evolution or replication.

14. Lineage-Constrained Governance Inheritance

[0136] Lineage operates not only to validate continuity but to bind descendants to governance constraints applicable to authorized ancestors. Permissions, prohibitions, enforcement classes, eligibility conditions, quarantine states, and related governance attributes persist across mutation, delegation, propagation, migration, and reconstitution unless expressly modified through verified policy authority under declared scope, validity, and freshness constraints. Ethical constraints are inheritable under the same mechanisms.

[0137] Referring now to FIG. 7, a lineage-constrained governance inheritance flow 700 is illustrated. The flow 700 begins with a parent agent object 710 in an authorized state. The parent agent object 710 includes a lineage record 712, one or more active policy references 714, and a current governance state 716. The governance state 716 may include eligibility indicators, enforcement class markers, trust degradation state, quarantine state, remediation requirements, propagation limits, or other governance-relevant attributes recorded in embedded memory or continuity records.

[0138] The parent agent object 710 proposes a lineage-affecting action 720, including mutation, delegation, propagation, migration, or reconstitution that would create, authorize, or activate a descendant agent object or otherwise alter lineage structure. Prior to permitting the lineage-affecting action 720, a governance inheritance evaluation 730 is performed. The governance inheritance evaluation 730 determines which constraints, permissions, and prohibitions associated with the parent agent object 710 must persist for the lineage-affecting action 720 to remain authorized under verified policy authority.

[0139] From this evaluation, a set of inherited constraints 740 is derived. The inherited constraints 740 may include required policy references, enforcement classes or enforcement treatment, eligibility restrictions, quarantine or trust degradation state, mutation or propagation limitations, memory constraints, or execution prohibitions. In some embodiments, inherited constraints 740 are explicitly defined by policy objects governing the lineage-affecting action 720. In other embodiments, active constraints are inherited by default unless a verified policy object expressly excludes, relaxes, or replaces a constraint under an authorized override procedure. Inheritance evaluation may further confirm applicability and freshness of inherited constraints for the descendant context, including trust-zone or substrate-class scope limitations.

[0140] If the lineage-affecting action 720 is authorized, a descendant agent object 750 is created or activated. The descendant agent object 750 includes a descendant lineage record 752 linking it to the parent agent object 710 and recording the inheritance event. The descendant agent object 750 further includes an inherited governance state 754 reflecting the inherited constraints 740. The inherited governance state 754 conditions the descendant's eligibility to instantiate execution contexts and perform governed actions from inception.

[0141] If the governance inheritance evaluation 730 determines that required constraints cannot be consistently inherited, or that the lineage-affecting action 720 is not authorized, the lineage-affecting action 720 is denied and no descendant agent object is created or authorized. Continuity is preserved by preventing unauthorized descent. If the governance inheritance evaluation 730 determines that required constraints cannot be consistently inherited, or that the lineage-affecting action 720 is not authorized, a lineage action denial 770 is issued and no descendant agent object 750 is created or authorized. The outcome in which no descendant is created is represented as a no-descendant outcome 775.

[0142] During subsequent operation, when the descendant agent object 750 proposes execution or further lineage-affecting actions, a subsequent authorization evaluation 760 is performed. The subsequent authorization evaluation 760 applies enforcement mechanisms that evaluate contemporaneous policy authorization together with the inherited governance state 754. An action otherwise permitted for a similarly situated object may be denied due to inherited prohibitions, enforcement class, quarantine state, or eligibility limitations. Inheritance therefore operates as a persistent constraint, not a one-time check.

[0143] Governance inheritance also supports escalation and containment. If a parent agent object 710 enters quarantine, incurs trust degradation, experiences repeated denials, or fails freshness requirements, descendant agent objects may inherit corresponding restrictions, be limited to remediation-only actions, or be prevented from further propagation, thereby limiting proliferation of untrusted descendants.

[0144] Although FIG. 7 depicts a parent–child relationship, the same inheritance principles apply to multi-generation descent, branching, controlled merging, migration across substrates, and reconstitution from stored states. Governance conditions persist across evolution unless modified through verified policy authority.

[0145] Accordingly, FIG. 7 illustrates lineage-constrained governance inheritance as a deterministic mechanism by which governance constraints propagate through lineage at the moment of lineage-affecting actions, preventing constraint shedding through mutation, replication, migration, or reconstitution while enabling authorized evolution.

15. Meta-Policy Objects and Architectural Governance Constraints

[0146] In addition to action-specific policy objects, the architecture supports meta-policy objects that impose higher-order architectural constraints across categories of system behavior. A meta-policy object is an externally governed, authenticated policy object whose scope applies to classes of actions or structural properties of agent objects rather than to a single action instance. Ethical constraints may be expressed as architectural constraints within such meta-policy objects.

[0147] Meta-policy objects are resolved, verified, and enforced through the same deterministic precondition gating mechanisms applicable to other policy objects, including scope, validity, freshness, and override evaluation. Authenticated content is immutable absent authorized

supersession. Meta-policy objects may be referenced via canonical aliases, applied implicitly at trust-domain boundaries, or enforced as mandatory baseline authority within a zone.

[0148] One category of constraint includes self-modification limits. A meta-policy object may restrict modification of governance-relevant fields, including policy reference fields, mutation descriptors, lineage markers, action-class definitions, or capability enablement absent additional external authorization. Proposed mutations that would weaken or remove required governance structures are denied unless explicitly authorized under verified policy authority.

[0149] A second category includes escalation prohibitions. Meta-policy objects may prohibit elevation of privilege, expansion of execution scope, access to higher-trust substrates, entry into restricted trust zones, assumption of supervisory roles, alteration of enforcement treatment, or substitution of required policy sets without explicit external authorization. Authority cannot be accumulated implicitly through repetition, gradual mutation, or internal state manipulation.

[0150] A third category includes memory integrity and concealment restrictions. Meta-policy objects may require that governance-relevant memory regions remain append-only, that denial and enforcement outcomes remain non-removable, and that audit-relevant records remain verifiable. Attempts to delete, redact, encrypt, obscure, or otherwise render governance-relevant history inaccessible may constitute governed actions that are denied or that trigger policy-defined enforcement treatment, including trust degradation, quarantine, or rollback.

[0151] A fourth category includes fork and proliferation control. Meta-policy objects may prohibit unrestricted forking, limit concurrent descendants, require authorization per fork, mandate inheritance of specified constraints, or require lineage continuity proofs during propagation. Unauthorized replication attempts are denied prior to instantiation.

[0152] Meta-policy objects may also enforce architectural invariants, including prohibitions against replacing required policy references with lower-enforcement alternatives, restrictions on introduction of new canonical aliases without quorum approval, mandatory baseline policy presence, signature-chain continuity requirements for overrides, and anti-rollback constraints on policy updates. These invariants reduce downgrade and substitution attack surfaces.

[0153] Meta-policy objects may operate with higher precedence than lower-level policy objects. Even where a lower-level policy authorizes a specific action instance, an applicable meta-policy

object may categorically prohibit the action class or impose additional preconditions. Authorization fails unless both meta-policy and lower-level policy constraints are satisfied.

[0154] Enforcement of meta-policy constraints is independent of agent cognition or predicted outcomes and occurs prior to instantiation of execution contexts or other governed transitions. By externalizing architectural governance constraints into authenticated, verifiable policy objects, the system preserves structural invariants across heterogeneous substrates while enabling layered, adaptable governance without embedding fixed compliance logic in agent-local code.

16. Quorum-Based Governance Override Mechanisms

[0155] The architecture supports quorum-based governance override mechanisms through which existing governance constraints may be replaced, supplemented, or conditionally superseded only upon authenticated multi-party approval. An override is itself a governed action implemented through an externally governed policy object and enforced through the same resolution, verification, succession, and precondition gating mechanisms applicable to other policy objects. In embodiments, approval of an override requires affirmative authorization by a plurality of authorized participants satisfying a quorum rule defined by applicable policy authority. In such embodiments, a replacement or override policy object includes a parent reference to the superseded policy object and a verifiable continuity linkage comprising a cryptographic signature chain, co-signature set, or equivalent chained continuity reference linking the replacement to the superseded policy object. Ethical constraints may be modified under the same quorum-controlled framework.

[0156] Referring now to FIG. 8, a quorum-based governance override flow 800 is illustrated. FIG. 8 illustrates that override publication under a canonical alias does not establish authority absent verification of the quorum approval and signature-chain continuity to the prior authoritative instance.

[0157] The flow 800 begins with an authoritative policy object 810 applicable to one or more governed action classes. A proposed override 820 is generated specifying intended modifications, supersession scope, and any temporal, trust-zone, substrate-class, lineage-class, or contextual limitations.

[0158] The proposed override 820 is submitted to a quorum approval process 830. The quorum approval process 830 defines an authorized participant set and an approval threshold, which may be numeric, weighted, role-based, or class-based. Participants may include administrators,

organizational entities, trustees, automated governance services, hardware-backed attestors, or combinations thereof. Each approving participant generates authentication material comprising a co-signature 840 or equivalent verifiable artifact. The quorum approval process 830 completes only when the defined threshold is satisfied. In embodiments, the threshold requires at least two distinct participants.

[0159] Upon quorum satisfaction, an override policy object 850 is constructed. The override policy object 850 encodes the modified or superseding constraints and specifies its relationship to the policy object 810. The override policy object 850 incorporates the co-signatures 840 and includes a continuity reference 852 linking it to the policy object 810. The continuity reference 852 may comprise a hash commitment, signature-chain reference, monotonic version indicator, or other verifiable linkage supporting anti-rollback and succession validation.

[0160] The override policy object 850 is disseminated through an authorized publication channel 860. Publication may include updating canonical alias resolution, issuing a successor under an existing alias binding, redirecting resolution under authorized procedures, and/or marking the policy object 810 as superseded or revoked under validity and freshness controls. Publication is itself subject to applicable governance constraints. Upon publication, the override policy object 850 becomes authoritative within its declared scope and validity bounds.

[0161] At runtime, the governance gate evaluates the override policy object 850 through the standard verification pipeline. The system verifies authenticity and integrity, confirms satisfaction of quorum requirements via co-signatures 840 under the defined quorum policy, and validates the continuity reference 852 relative to the policy object 810. If verification succeeds, the override policy object 850 governs authorization decisions within scope. If verification fails, the override is non-authoritative and the policy object 810 or another valid successor remains controlling.

[0162] At runtime, the governance gate performs a runtime verification of override 870. The runtime verification of override 870 validates authenticity, quorum satisfaction through co-signatures 840, and continuity reference 852 linking the override policy object 850 to the authoritative policy object 810. If verification succeeds, the override policy object 850 is deemed authoritative 880 within its declared scope and the override policy objects 850 governs within declared scope 851. If verification fails, the override is rejected 890 and the authoritative policy object 810 or another valid successor remains controlling 811.

[0163] Governance history may be preserved by recording the policy object 810, the proposed override 820, the quorum approval process 830, and the override policy object 850 in an append-only audit record and, in embodiments, in continuity records associated with affected agent objects.

[0164] Overrides may be permanent, temporary, or conditional. The override policy object 850 may include validity windows, scope limitations, additional attestation requirements, or reversion conditions. Upon expiration or satisfaction of termination conditions, authority may revert to a prior policy object, transition to another authorized successor, or be replaced by a further quorum-approved override, without requiring agent-local modification.

[0165] By requiring quorum approval, co-signatures, and continuity validation, the system ensures that governance modifications are deliberate, verifiable, and resistant to unilateral weakening. Overrides are enforceable only if themselves authorized as valid policy authority and are subject to the same deterministic precondition gating applicable to execution and other governed transitions.

17. Distributed Alias Publication and Override Dissemination

[0166] The architecture supports distributed publication and dissemination of policy objects, including override policy objects, through canonical alias–based resolution mechanisms that do not require centralized control. Because agent objects reference governance authority via aliases rather than embedding policy content, updates are effected by publishing new authoritative policy object instances under existing aliases rather than mutating agent objects or authenticated policy content.

[0167] Following construction of an override policy object, publication associates a canonical alias with the newly authoritative instance. Prior instances may be marked superseded, deprecated, or revoked under applicable validity, freshness, and revocation controls. Alias-to-policy associations may be expressed through signed alias bindings, resolution records, append-only publication events, or other verifiable artifacts supported by the resolution substrate.

[0168] Distributed alias systems may be implemented using federated registries, adaptive indexes, content-addressable stores, distributed ledgers, replication protocols, gossip-based dissemination networks, or combinations thereof. No single node is required to function as global authority. Each participating node independently applies deterministic verification rules to determine whether a resolved policy object instance is authoritative, including verification under the applicable

trust model, validation of quorum artifacts for override instances, validation of continuity references to prior instances, and evaluation of scope, validity, freshness, revocation, and anti-rollback constraints.

[0169] Because dissemination may be asynchronous due to latency, partitioning, or caching, authorization decisions are based on verified authority available at evaluation time, subject to policy-defined freshness and cache revalidation rules. Where a locally resolved instance is later determined to be superseded, revoked, or stale, subsequent authorization attempts are denied or re-evaluated upon resolution of updated authoritative policy, consistent with deterministic precondition gating.

[0170] To resist downgrade and replay attacks, alias resolution may require continuity validation before recognizing a successor as authoritative. An override instance may be required to include quorum artifacts and a continuity reference to a prior instance and to satisfy monotonic versioning or anti-rollback commitments. Execution substrates may reject older policy instances when a newer authorized replacement is verifiable under the applicable trust model and freshness constraints, even if the older instance remains cached.

[0171] Distributed publication supports scoped dissemination. A policy object instance may be published with scope limitations applicable only to specified trust domains, geographic regions, execution substrate classes, agent-object classes, or lineage classes. Alias resolution may return different authoritative instances for the same canonical alias depending on verified contextual parameters, enabling staged deployment, controlled rollout, or trust-zone-specific updates without fragmenting agent implementations.

[0172] Auditability is preserved through recording of publication events, override events, alias-binding changes, supersession events, and revocation events in append-only audit records. Execution substrates may retain evidence of which policy instance was resolved and applied at authorization time, enabling retrospective validation despite asynchronous propagation.

18. Fallback Enforcement Agents and Distributed Monitoring

[0173] The architecture includes fallback enforcement agents that operate alongside primary governance gates to preserve governance integrity across heterogeneous and decentralized substrates. Fallback enforcement agents do not participate in the critical authorization path and do

not replace cryptographic precondition gating. Instead, they provide secondary validation, anomaly detection, and enforcement signaling based on governance metadata and verifiable artifacts.

[0174] Fallback enforcement agents are particularly suited for environments with heterogeneous substrate capabilities, partial observability, asynchronous alias dissemination, and intermittent connectivity. They function as independent validators rather than centralized controllers and operate without access to internal cognition or execution payloads.

[0175] Referring now to FIG. 9, a distributed fallback enforcement architecture 900 is illustrated. The architecture 900 includes one or more execution substrates 910 hosting agent objects 912. Each agent object 912 remains subject to runtime governance gating and authorization prior to execution or other governed transitions.

[0176] One or more fallback enforcement agents 920 are distributed across the execution substrates 910. A fallback enforcement agent 920 may be co-located with a substrate 910, operate remotely, roam across nodes, or exist as a distributed validator set within a trust domain. The fallback enforcement agent 920 observes governance-relevant events and validates that governance gating and authority transitions have occurred in accordance with externally defined policy authority.

[0177] The fallback enforcement agent 920 receives governance signals 922 from execution substrates 910 and/or agent objects 912. Governance signals 922 may include policy resolution outcomes, verification results, authorization decisions, denials, override applications, freshness failures, lineage updates, audit references, and execution feedback designated as governance-relevant. The fallback enforcement agent 920 evaluates these artifacts to confirm compliance with applicable scope, validity, freshness, anti-rollback, quorum, and continuity requirements.

[0178] Using governance signals 922, the fallback enforcement agent 920 performs a compliance evaluation 930. The compliance evaluation 930 deterministically verifies that required policy objects were properly resolved and authenticated, that override policy objects satisfy quorum and continuity requirements, and that lineage continuity and freshness constraints were preserved for lineage-affecting or policy-dependent actions.

[0179] If the compliance evaluation 930 detects a governance anomaly—such as invalid override artifacts, stale or revoked authority usage, unauthorized lineage forks, inconsistent authority

observations across substrates, or repeated denial patterns indicative of evasion attempts—the fallback enforcement agent 920 emits an enforcement signal 940. The enforcement signal 940 may comprise a trust degradation signal, quarantine recommendation, directive to append a violation record, alert to enforcement components, or requirement for additional verification prior to further authorization. The enforcement signal 940 influences subsequent eligibility or enforcement state but does not itself instantiate execution.

[0180] Enforcement signals 940 propagate through a distributed signaling fabric 950 to other execution substrates 910, fallback enforcement agents 920, resolver components, or audit systems. Recipients independently verify authenticity, integrity, scope, and applicability of enforcement signals 940 prior to updating governance-relevant state. Enforcement signals 940 are disseminated through the distributed signaling fabric 950 to execution substrates 910 and other fallback enforcement agents 920. The execution substrate 910 and fallback enforcement agent 920 shown at the lower portion of FIG. 9 represent distributed instances of the same execution substrate 910 and fallback enforcement agent 920 participating in the distributed signaling fabric 950.

[0181] In embodiments, fallback enforcement agents 920 monitor override dissemination and freshness convergence by comparing observed policy authority across substrates, validating quorum artifacts and continuity references, and detecting partial dissemination, downgrade attempts, or unauthorized authority injection. Upon detecting inconsistency, a fallback enforcement agent 920 may emit enforcement signals 940 restricting authorization to remediation-only actions or temporarily denying instantiation of execution contexts pending authoritative convergence.

[0182] Fallback enforcement agents 920 operate as defense-in-depth. Primary authorization remains based on deterministic resolution and verification prior to execution. Fallback enforcement agents provide cross-substrate consistency checking,

19. Append-Only Governance Audit and Verification Records

[0183] The disclosed systems implement append-only governance audit and verification records that preserve durable, tamper-evident evidence of governance activity across agent objects and distributed execution substrates. These records capture what authority was resolved, what verification occurred, and what authorization or enforcement outcome resulted. Audit records do not grant authority or participate in runtime authorization; they record governance events as verifiable artifacts.

[0184] Audit records are append-only. Once recorded, an entry cannot be altered, removed, or reordered without detectable integrity violation. Append-only properties may be enforced through cryptographic chaining, content-addressed storage, write-once semantics, distributed ledgers, replicated logs, or combinations thereof.

[0185] Referring now to FIG. 10, an append-only governance audit and verification record system 1000 is illustrated. The system 1000 includes one or more governance enforcement points 1010, including governance gates, policy resolution subsystems, verification modules, fallback enforcement agents, publication components, and enforcement-capable execution substrates. Upon occurrence of a governance-relevant event, an enforcement point 1010 generates an audit event 1020.

[0186] Governance-relevant events include policy resolution attempts and outcomes; verification results; scope, freshness, revocation, and anti-rollback determinations; authorization permits and denials; override approvals and quorum artifact validation; continuity-reference validation; trust degradation, quarantine, or rollback transitions; enforcement signal emissions; and publication or supersession events.

[0187] Each audit event 1020 is a structured, machine-readable record containing sufficient information for later verification and contextual reconstruction. Such information may include identifiers or fingerprints of the acting agent object; referenced canonical aliases; identifiers or fingerprints of resolved policy objects; verification and applicability results; enforcement class applied; timestamps or epochs; substrate and trust-zone identifiers; and associated authentication material. Audit event 1020 captures governance evidence and decisions rather than execution payloads or internal cognition.

[0188] Audit events 1020 are appended to an append-only audit log 1030. The append-only audit log 1030 may be implemented as a local or distributed log, content-addressable store, append-only database, ledger, or hybrid structure. In embodiments, the audit log 1030 is replicated across nodes or anchored to external verification systems. The log preserves ordering and integrity sufficient to prove what authority was applied at a given time and what outcome occurred.

[0189] Entries in the append-only audit log 1030 may be cryptographically linked to prior entries to form an integrity chain 1040. The integrity chain 1040 renders removal, modification, or

reordering detectable. Entries may be authenticated by the originating enforcement point 1010 and/or anchored to external attestations to provide tamper-evidence and source attribution.

[0190] The append-only audit log 1030 supports audit queries and verification requests 1050. Authorized auditors, compliance systems, monitoring systems, fallback enforcement agents, contractual interfaces, or regulatory interfaces may issue audit queries 1050. Responses may include cryptographic proofs of inclusion, ordering, and integrity relative to the append-only structure and query scope, without modifying the log. In response to an audit query 1050, the system produces an audit proof or response 1060, which may include inclusion proofs, ordering proofs, integrity-chain validation artifacts, and authentication material sufficient to verify the queried audit events without modifying the append-only audit log 1030.

[0191] In embodiments, the audit system 1000 operates independently of runtime authorization such that governance gating does not depend on successful logging. However, failure to record required audit events, failure to anchor required proofs, or detection of integrity anomalies may constitute governance-relevant violations triggering policy-defined enforcement treatment, including denial of subsequent governed actions, trust degradation, enforcement class escalation, or quarantine.

[0192] The append-only audit log 1030 enables forensic reconstruction and compliance verification by preserving policy resolutions, verification outcomes, override continuity, enforcement outcomes, and temporal ordering. It does not retroactively authorize or alter decisions; it preserves objective evidence of what was evaluated and decided under verified authority at the time of evaluation.

[0193] Accordingly, FIG. 10 illustrates append-only governance audit and verification records 1000 as tamper-evident evidentiary infrastructure supporting cryptographically enforced governance by recording policy resolutions, verification outcomes, override events, freshness and revocation determinations, denials, and enforcement outcomes in an append-only structure suitable for retrospective validation and compliance review.

20. Governance Enforcement Across Heterogeneous Execution Substrates

[0194] Governance enforcement operates across heterogeneous execution substrates as a portable, verifiable precondition to instantiation of execution contexts and other governed

transitions. Authority is derived from externally verifiable policy objects and embedded agent-object state, not from substrate-local schedulers or discretionary security models. Enforcement therefore travels with the agent object and is evaluated wherever execution is attempted.

[0195] Referring now to FIG. 11, a governance enforcement architecture across heterogeneous execution substrates 1100 is illustrated. The architecture 1100 includes a cloud execution substrate 1110, an edge execution substrate 1120, a federated execution substrate 1130, and an intermittently connected execution substrate 1140, each representing distinct resource, connectivity, and trust characteristics.

[0196] An agent object 1150 is portable across substrates 1110–1140. The agent object 1150 carries governance-relevant state, including policy references, memory, lineage, and, in embodiments, eligibility indicators. No substrate maintains exclusive governance state or independently grants authority.

[0197] Each execution substrate includes a governance gate 1160. The governance gate 1160 may be implemented as a runtime component, middleware layer, verifier library, validation service, or hardware-backed attestor depending on substrate capability. Upon proposal of a governed action, the governance gate 1160 evaluates eligibility for instantiation based on resolved and verified policy authority and embedded agent-object state.

[0198] The governance gate 1160 invokes a policy resolution and verification process 1170. The process 1170 resolves canonical aliases, verifies resolved policy objects under an applicable trust model, evaluates scope, validity, freshness, and anti-rollback constraints, and incorporates memory-derived and lineage-derived eligibility. The process 1170 produces a deterministic permit-or-deny result without requiring centralized orchestration.

[0199] In intermittently connected environments such as the intermittently connected execution substrate 1140, the process 1170 may rely on cached policy objects 1180 and previously verified authority fingerprints. Cached policy objects 1180 are accepted only if authenticity, scope, validity, freshness, and anti-rollback requirements remain satisfied under revalidation policy. If required authority cannot be verified due to expiration, revocation, staleness, or inability to revalidate, instantiation is denied rather than executed optimistically.

[0200] Federated execution substrates 1130 may impose additional trust-zone constraints, including requirements for co-signature by local authorities, joint satisfaction of global and local policy objects, or application of trust-zone-specific meta-policy constraints. Such requirements are enforced through policy resolution and verification semantics rather than substrate-specific discretionary logic.

[0201] Execution outcomes 1190, including permits, denials, freshness failures, execution feedback, trust degradation, and quarantine transitions, may be recorded in embedded memory of the agent object 1150 and, where available, in append-only audit logs. These outcomes persist across substrates and influence subsequent authorization evaluations regardless of location.

[0202] No execution substrate 1110–1140 serves as a primary source of governance authority. Each substrate validates authorization derived from verified policy objects and embedded state. Where a substrate lacks capability to verify authority at the required assurance level, it denies instantiation rather than executing without verification.

[0203] In FIG. 11, each execution substrate 1110–1140 includes a governance gate 1160 and an associated policy resolution and verification process 1170, each producing an execution outcome 1190 corresponding to permit, denial, freshness failure, or other deterministic governance result. Accordingly, FIG. 11 illustrates uniform governance enforcement across heterogeneous execution substrates 1100. Governance is applied as a substrate-agnostic, verifiable eligibility condition that remains enforceable across cloud, edge, federated, and intermittently connected environments without centralized control, while preserving deterministic authorization and durable constraint propagation.

21. Governance Enforcement Without Persistent Keypairs

[0204] In an optional embodiment, governance enforcement is performed without reliance on persistent cryptographic keypairs. In environments where long-lived private keys are unavailable, undesirable, or insecure, authority is established through verifiable continuity of governance evidence rather than static credentials. Deterministic precondition gating is preserved by evaluating objectively verifiable records under externally governed policy authority.

[0205] Referring now to FIG. 12, a governance enforcement architecture without persistent keypairs 1200 is illustrated. The architecture 1200 includes an agent object 1210 operating without

persistent key storage. The agent object 1210 carries governance-relevant memory 1212 comprising prior authorization outcomes, denials, governance-designated execution feedback, lineage continuity markers, and audit references. Authority is asserted through continuity evidence embodied in memory 1212 and associated records rather than a private key.

[0206] When the agent object 1210 proposes a governed action 1220, a governance gate 1230 initiates evaluation. The governance gate 1230 performs a memory-resolved identity evaluation 1240. The memory-resolved identity evaluation 1240 determines whether memory 1212 and associated continuity records form a coherent, verifiable continuity chain under an applicable trust model. Evaluation 1240 may include validation of hash-linked entries, inclusion proofs for audit references, verification of lineage continuity, comparison against previously observed authority fingerprints, and detection of discontinuities indicative of tampering, fork anomalies, or unauthorized reconstitution.

[0207] In parallel, the governance gate 1230 performs trust-slope validation 1250. Trust-slope validation 1250 evaluates evolution of eligibility or enforcement state over time based on recorded governance events. Policy-defined criteria determine whether trust state is stable, improving, or degrading. A stable or improving trust slope may satisfy authorization conditions for specified action classes; a degrading trust slope may restrict eligibility, elevate enforcement class, require corroboration, or trigger quarantine evaluation. If authorization fails, the governance gate 1230 produces a denial outcome 1285 preventing instantiation of an execution context or other governed transition.

[0208] Externally governed policy objects 1260 remain authoritative. The policy objects 1260 define required continuity depth, acceptable decay rates, remediation prerequisites, audit anchoring frequency, corroboration thresholds, and action-class-specific assurance requirements. In embodiments, policy objects 1260 may require traditional signatures or hardware-backed attestations for higher-risk action classes while permitting continuity-based authorization for lower-risk classes.

[0209] Based on memory-resolved identity evaluation 1240, trust-slope validation 1250, and verified policy objects 1260, the governance gate 1230 produces an authorization decision 1270. If affirmative, a permit 1280 is issued enabling instantiation of an execution context or other governed transition. If negative, instantiation is denied as a valid non-execution outcome.

[0210] Execution feedback 1290 and subsequent enforcement outcomes may be recorded into governance-relevant memory 1212, updating continuity evidence and trust-slope state for future evaluations. Authority may therefore be maintained or restored through verifiable continuity bounded by policy-defined criteria.

[0211] This embodiment applies deterministic, policy-defined evaluation to objectively verifiable records and does not rely on subjective scoring or probabilistic inference. If continuity cannot be established, audit references cannot be validated, lineage continuity fails, or trust-slope thresholds are unmet, authorization fails.

[0212] Accordingly, FIG. 12 illustrates governance enforcement without persistent keypairs, substituting memory-resolved identity and trust-slope validation for static key-bound identity while preserving deterministic authorization and externally governed authority.

22. Freshness, Revocation, and Anti-Rollback Controls

[0213] The systems disclosed herein enforce freshness of governance authority as an objective eligibility precondition evaluated before instantiation of any execution context for a proposed action. Freshness controls constrain when a resolved policy object is treated as authoritative and prevent reliance on stale, revoked, superseded, downgraded, or replayed authority, including under caching, intermittent connectivity, partial replication, or distributed dissemination. Freshness determinations are evaluated as part of the alias-resolution pipeline and may be re-evaluated immediately prior to authorization to ensure that the final authorization decision reflects the evaluation time basis and current revocation/monotonicity state. Ethical constraints constitute an example class of governance constraints subject to the same controls.

[0214] In embodiments, a governance gate receives a proposed action associated with an agent object and deterministically outputs either (i) an authorization permit enabling instantiation or (ii) a denial outcome constituting valid non-execution. The denial outcome is produced without instantiating an execution context and may be recorded as governance-relevant memory, including in a memory field of the agent object and/or an append-only audit record. In one embodiment, the denial outcome is a structured denial result comprising at least: (a) an identifier of the proposed action or action class, (b) identifiers or fingerprints of evaluated policy objects and/or associated revocation artifacts, and (c) a failure basis indicator identifying a freshness failure type (validity-window failure, revocation failure, or anti-rollback failure). Such recorded denial results may be

used as an objective input to subsequent authorization evaluations (e.g., eligibility gating, quarantine state, or revalidation requirements) without requiring model alignment scoring or outcome prediction.

[0215] In one embodiment, freshness is enforced through validity-window semantics. A policy object may include a validity component defining an activation boundary and an expiration boundary, including notBefore and notAfter semantics. A policy object is treated as non-authoritative prior to activation and after expiration regardless of authenticity, declared scope, or remaining cache availability. The governance gate evaluates validity-window satisfaction during authorization and denies authorization when the evaluation time falls outside the defined window. In embodiments, alias resolution is validity-aware: candidate policy objects discovered for a canonical alias are filtered by excluding candidates that are non-active or expired under the evaluation time basis, and by preferring candidates whose validity window encompasses the evaluation time. Validity-window failures may be recorded in an append-only audit record as freshness-related denial events including the evaluation time basis and boundary data used to determine non-authoritativeness.

[0216] In another embodiment, revocation operates as negative authority. A revocation artifact renders a policy object non-authoritative notwithstanding authenticity, scope applicability, or remaining validity duration. Revocation may be expressed through revocation anchors, revocation lists, revocation epochs, or other artifacts verifiable under an applicable trust model and resolvable via canonical aliases and/or associated publication substrates. During authorization, the governance gate evaluates applicable revocation artifacts and denies instantiation of an execution context if a resolved policy object is determined to be revoked. In embodiments, alias resolution is revocation-aware: candidate policy objects discovered for a canonical alias are filtered by excluding candidates determined to be revoked under the applicable trust model. Revocation determinations may be recorded in the append-only audit record, including identifiers or fingerprints of relied-upon revocation artifacts and resulting denial outcomes.

[0217] In a further embodiment, anti-rollback constraints prevent reliance on an older authority instance when a newer authorized successor exists or when a policy-defined minimum acceptable version is required. Anti-rollback may be expressed through monotonic version indicators associated with canonical aliases, verifiable continuity chains between successive authority instances, hash commitments to a latest-known-good authority instance, policy-defined minimum acceptable version

constraints, or combinations thereof. In embodiments, the governance gate rejects a resolved policy object whose version indicator is less than a minimum for the canonical alias, including a minimum recorded as governance-relevant memory (e.g., in embedded memory of the agent object) and/or a minimum recorded in an append-only audit anchoring record. In embodiments, successor policy objects are required to include a verifiable continuity reference to a prior authoritative instance; a candidate lacking a required continuity reference is treated as non-authoritative even if authentic. A latest-known-good fingerprint stored in embedded memory or in an append-only audit record may establish a checkpoint constraint against which subsequent candidates are evaluated, thereby preventing stale-policy downgrade attempts. Anti-rollback failures may be recorded as freshness failures including rejected version indicators, rejected continuity references, and the applicable minimum or checkpoint constraints.

[0218] Freshness controls are applied as candidate-set filtering during alias resolution and may be rechecked immediately prior to authorization. In an embodiment, alias resolution yields a candidate set of policy objects for a canonical alias, and the candidate set is filtered based on one or more freshness constraints (validity-window satisfaction, revocation state, and/or anti-rollback monotonicity) to produce a selected set of policy objects eligible for cryptographic verification and authorization evaluation. This ordering permits deterministic denial in cases where candidates are stale or revoked, without relying on cryptographic verification of ineligible candidates, and reduces the risk of authorization based on cached but outdated authority.

[0219] Referring now to FIGS. 13A and 13B, a freshness, revocation, and anti-rollback control flow 1300 is illustrated. FIG. 13A illustrates authorization-time freshness filtering, verification, checkpoint evaluation, and deterministic permit-or-deny outcomes. FIG. 13B illustrates append-only recording of freshness-related events and persistence of latest-known-good checkpoint records used to enforce anti-rollback constraints.

[0220] The flow 1300 begins when a governance gate 1310 receives a proposed action 1312 associated with an agent object and extracts one or more canonical policy aliases 1314 required for authorization of the proposed action 1312. The governance gate 1310 issues a resolution request 1316 to a Dynamic Alias System 1320, the resolution request 1316 including the canonical policy aliases 1314 and, in embodiments, evaluation context 1318 comprising a trust zone indicator, a substrate class indicator, and a time basis. The resolution request 1316 includes an evaluation context 1318 comprising at least a time basis, trust-zone indicator, and substrate-class indicator used

by the Dynamic Alias System 1320 to evaluate validity-window satisfaction, revocation state, and anti-rollback monotonicity constraints.

[0221] The Dynamic Alias System 1320 performs candidate discovery 1322 to obtain candidate policy objects 1324. The candidate policy objects 1324 constitute a candidate set obtained by alias resolution. The Dynamic Alias System 1320 applies a validity-window filter 1326 excluding candidate policy objects whose notBefore or notAfter semantics do not encompass the evaluation time. The Dynamic Alias System 1320 performs a revocation check 1328 by resolving one or more revocation artifacts 1330 associated with the canonical policy alias 1314 and excluding candidates determined to be revoked under the applicable trust model. The Dynamic Alias System 1320 performs an anti-rollback evaluation 1332 which, in embodiments, compares candidate version indicators to a monotonicity constraint 1334 and/or validates required continuity references, rejecting candidates that fail anti-rollback requirements. The Dynamic Alias System 1320 outputs one or more selected policy objects 1336 that satisfy these controls. Accordingly, the validity-window filter 1326, revocation check 1328, and anti-rollback evaluation 1332 collectively implement filtering of the candidate set based on freshness constraints prior to cryptographic verification and authorization.

[0222] The governance gate 1310 submits the selected policy objects 1336 to a verification module 1340 which produces verification results 1342 indicating authenticity and integrity under the applicable trust model. The governance gate 1310 performs an applicability evaluation 1344 including re-evaluation of validity-window satisfaction and freshness constraints 1346 at the time of authorization. In embodiments, the governance gate 1310 evaluates a latest-known-good checkpoint 1348 stored in embedded memory or in an append-only audit record and denies candidates that violate the checkpoint constraint.

[0223] If any validity-window, revocation, or anti-rollback control fails for a required canonical policy alias 1314, the governance gate 1310 produces a denial outcome 1350 and prevents instantiation of an execution context for the proposed action 1312. The denial outcome 1350 constitutes a valid non-execution result and may trigger secondary enforcement treatment under verified policy authority. In embodiments, the governance gate emits the structured denial result described herein and stores the structured denial result in a memory field of the agent object as governance-relevant memory and/or records the structured denial result in an append-only audit record. If all required canonical policy aliases 1314 satisfy validity-window controls, are not

revoked, satisfy anti-rollback constraints, and are verified authentic, the governance gate 1310 produces an authorization permit 1352 enabling an execution substrate 1354 to instantiate an execution context 1356.

[0224] The flow 1300 further includes an append-only audit ledger 1360, which records freshness-relevant events 1362 and, in embodiments, records a freshness failure record 1364 and/or a latest-known-good checkpoint record 1366 corresponding to the authorization decision.. The audit ledger 1360 may record resolution results, candidate filtering outcomes, validity-window failures, revocation determinations, anti-rollback determinations, and resulting authorization decisions including denial outcomes 1350. A freshness failure record 1364 may include an alias identifier, candidate policy fingerprint, evaluation time basis, and a failure type indicating validity-window failure, revocation failure, or anti-rollback failure. In embodiments, after a successful authorization permit 1352 is produced, the audit ledger 1360 records a latest-known-good checkpoint 1366 associated with a canonical policy alias, enabling subsequent monotonicity enforcement and detection of stale-policy downgrade attempts.

[0225] Accordingly, FIG. 13 illustrates validity-window controls, revocation controls, and anti-rollback controls applied during alias resolution and authorization as objective preconditions to instantiation of execution contexts, together with recording of freshness-related determinations in governance-relevant memory and/or an append-only audit ledger to provide verifiable evidence of freshness enforcement and resulting non-execution outcomes.

23. Resistance to Circumvention, Forking, and Policy Evasion

[0226] The disclosed systems resist circumvention, policy evasion, unauthorized forking, substitution, downgrade, replay, and stale-authority reliance by enforcing governance as a structural eligibility condition for instantiation of execution contexts and other governed state transitions. Governance enforcement is applied as a deterministic precondition to execution, mutation, delegation, propagation, migration, and lineage-affecting actions. Any attempt to bypass, weaken, or evade required governance constraints results in denial of the proposed transition as a valid non-execution outcome. Ethical constraints constitute a case of governance constraints enforced under the same mechanisms.

[0227] Governance authority is external to agent-local control. Agent objects do not embed authoritative policy logic and cannot unilaterally alter or suppress governing authority. Required

policy objects must be resolved and verified under an applicable trust model prior to authorization. There is no authorized execution path for governed action classes in which required governance checks are omitted or treated as optional.

[0228] Policy stripping or nullification attempts are ineffective because modification of a policy reference field constitutes a governed mutation subject to precondition gating. A mutation that would remove required policy references, violate required policy sets, or contravene meta-policy invariants is denied and the resulting state is not instantiated. A modified state lacking valid continuity linkage to a previously authorized predecessor is treated as discontinuous and ineligible for governed action.

[0229] Substitution and downgrade attacks are mitigated through resolution and binding procedures incorporating authenticity verification, scope applicability, validity-window enforcement, revocation handling, and anti-rollback constraints. A policy object instance that is expired, revoked, superseded, outside declared scope, or lacking required continuity linkage is non-authoritative even if locally accessible. Anti-rollback controls and checkpoint constraints prevent acceptance of older authority instances once a newer authorized successor has been observed. Attempts to rely on stale or superseded authority are denied when freshness constraints are not satisfied and may be recorded as governance-relevant events affecting subsequent eligibility or enforcement state.

[0230] Unauthorized forks, replication, replay, and illicit reconstitution are resisted through continuity validation. Instantiation of execution contexts and other governed transitions may be conditioned on verification that the current state is a valid successor of a previously authorized state. An object reconstructed from partial data, cloned without authorized transition, replayed from an outdated snapshot, or introduced without verifiable continuity linkage is treated as discontinuous and ineligible for governed action. Constraint shedding through unsanctioned descendants or parallel instances is thereby prevented.

[0231] Migration does not reset governance state. Governance-relevant memory, lineage continuity records, trust or enforcement markers, quarantine state, and eligibility indicators are intrinsic to the agent-object representation and are evaluated at each substrate prior to instantiation. An agent object denied authorization on one substrate remains ineligible on another substrate unless authorization conditions are satisfied under verified policy authority.

[0232] Evasion through concealment, erasure, or selective disclosure of governance-relevant memory is constrained by integrity requirements. Deletion, redaction, or withholding of governance-

relevant memory or audit evidence required for authorization constitutes a governed action and is denied when not authorized. Tamper-evident memory structures, audit inclusion proofs, and continuity validation detect discontinuities resulting from manipulation, and such discontinuities result in denial, trust degradation, quarantine, rollback, or other policy-defined enforcement outcomes.

[0233] Circumvention resistance does not depend on detection of malicious intent or predictive analysis. Each governed transition must satisfy verified external policy authority and continuity constraints. If required verification, scope, freshness, anti-rollback, memory-derived eligibility, or lineage continuity conditions are not satisfied, the transition is not instantiated.

[0234] In embodiments, monitoring and audit mechanisms provide defense-in-depth by validating override continuity, dissemination consistency, and enforcement-state transitions, and by emitting governance-relevant signals under policy-defined treatment. These mechanisms complement structural precondition gating without replacing it.

[0235] Accordingly, governance is inseparable from eligibility to instantiate execution contexts and perform governed transitions. By eliminating authorized paths that bypass required verification and continuity constraints, the system prevents stripping, downgrading, replaying, or evading constraints across distributed environments while preserving enforceable non-execution outcomes.

24. Governance Non-Execution as a First-Class System Outcome

[0236] The disclosed systems treat non-execution of a proposed governed action as a valid, intended, and enforceable system outcome. Instantiation of an execution context is conditioned on satisfaction of required governance preconditions. When required authorization conditions are not satisfied at evaluation time, refusal to instantiate the execution context is the complete and correct result. Ethical governance constitutes a case of governance constraints capable of producing non-execution under the same mechanisms.

[0237] Execution is contingent upon verified policy authority and objective eligibility conditions. Instantiation occurs only when required policy objects are resolved, verified, applicable under declared scope, valid and fresh, compliant with anti-rollback constraints, and consistent with lineage continuity and embedded eligibility state. If any required condition is unsatisfied, the proposed execution or other governed transition is denied.

[0238] Governance non-execution may result from, without limitation: failure to resolve required canonical aliases; failed authenticity or integrity verification; scope inapplicability; expiration or revocation; supersession under anti-rollback constraints; freshness failure; invalid lineage continuity; memory-derived ineligibility; unmet quorum or override requirements; enforcement-class denial; or unmet remediation prerequisites. In each instance, instantiation is prevented prior to execution. No partial execution occurs and no rollback is required to preserve governance guarantees.

[0239] Denial is terminal for the proposed action at the time of evaluation. Subsequent attempts require re-evaluation under then-current verified policy authority and embedded governance state. Non-execution does not represent an operational malfunction but reflects absence of required authority.

[0240] Denial events may be recorded in embedded memory and/or append-only audit records, including identifiers or fingerprints of evaluated policy objects, verification and applicability determinations, freshness or continuity results, and the resulting authorization outcome. Such records provide objective evidence that non-execution resulted from unsatisfied governance preconditions.

[0241] Non-execution is portable across substrates. An agent object ineligible on one execution substrate remains ineligible on another unless authorization conditions are satisfied under verified policy authority. Governance restraint therefore persists across migration, replication, or redeployment.

[0242] Systems layered atop the disclosed architecture may treat non-execution outcomes as actionable results driving policy-defined remediation, escalation, alternative routing, override consideration, throttling, trust adjustment, or quarantine evaluation. Repeated denials may trigger additional enforcement treatment under applicable policy.

[0243] Accordingly, governance non-execution is a first-class system outcome. Refusal to instantiate an execution context is an explicit, verifiable, and auditable result of unsatisfied governance preconditions, ensuring that restraint is enforceable and portable across distributed environments.

25. Interoperability with External Governance and Regulatory Systems

[0244] The disclosed systems interoperate with external governance, compliance, contractual, and regulatory frameworks by externalizing authority into authenticated policy objects rather than embedding jurisdiction-specific rules within agent or substrate code. External requirements are expressed as verifiable policy objects evaluated through canonical alias resolution and deterministic authorization prior to instantiation of execution contexts or other governed transitions. Ethical constraints constitute a case of governance constraints representable under the same framework.

[0245] External authorities, including regulators, standards bodies, compliance offices, certifying entities, or delegated trustees, may issue, co-issue, attest to, revoke, or supersede policy objects under an applicable trust model. Such policy objects define enforceable constraints, declared scope, validity and freshness semantics, enforcement treatment, and override conditions. Execution substrates and governance gates evaluate these objects using the same resolution, verification, applicability, and authorization pipeline applied to other governance constraints, without embedding regime-specific logic.

[0246] Canonical aliases provide stable identifiers for external governance domains. Regulatory or organizational updates are introduced by publishing successor policy objects under authorized alias-resolution procedures rather than modifying governed agent objects or redeploying substrates. This supports regulatory change, emergency directives, staged rollout, and jurisdictional variation while preserving deterministic precondition gating.

[0247] Multiple external frameworks may apply concurrently. A proposed action may require joint authorization by multiple policy objects corresponding to distinct authorities. Authorization occurs only when each required policy object is resolved, verified, applicable under declared scope, valid and fresh, and collectively authorizes the action. Precedence, conflict handling, and override are governed by policy-defined rules rather than discretionary substrate behavior.

[0248] Scope declarations within policy objects enable contextual interoperability. Policy objects may specify geographic, organizational, trust-zone, substrate-class, agent-class, or lineage-class applicability. Governance gates evaluate scope at runtime, permitting a unified architecture to enforce localized constraints without divergent code paths.

[0249] Append-only governance audit and verification records support external compliance and evidentiary requirements. Recorded events may include policy resolutions, verification outcomes, applicability determinations, authorization permits, denials, override continuity, and freshness or

revocation determinations. Such records may be exported or attested to external auditors or regulators and provide objective evidence of evaluated authority and resulting authorization or non-execution outcomes.

[0250] Non-execution functions as compliance-by-design. Where required external authority is absent, stale, revoked, superseded, or inapplicable, instantiation of execution contexts is prevented. Recorded denial outcomes demonstrate that prohibited action classes were not instantiated under unsatisfied governance conditions.

[0251] External interoperability does not transfer execution control to external systems. External authorities express constraints through policy objects; execution substrates and governance gates remain responsible for resolution, verification, and deterministic authorization. Governance authority remains verifiable, portable, and substrate-independent.

[0252] Accordingly, the disclosed systems provide interoperable governance by representing external requirements as authenticated policy objects, supporting layered and scoped authorization, maintaining append-only evidentiary records, and enforcing non-execution where authority is absent, thereby enabling adaptable and auditable operation across diverse regulatory and organizational contexts.

26. Portability of Governance Across Systems

[0253] The disclosed systems enable portability of governance across heterogeneous platforms, organizations, and execution environments by encapsulating authority in externally governed, verifiable policy objects rather than in substrate-specific logic. Governance is therefore not bound to a particular runtime, infrastructure provider, or organizational boundary. Ethical constraints constitute a class of governance constraints portable under the same mechanisms.

[0254] Policy objects encode enforceable constraints, scope declarations, validity and freshness semantics, and enforcement treatment, and are resolved and verified independently of any specific execution substrate. A policy object retains authority wherever it is authenticated under a recognized trust model and determined applicable under declared scope, validity, freshness, and continuity constraints. Authority thus travels as verifiable policy content rather than as infrastructure configuration.

[0255] Canonical aliases provide indirection supporting cross-system portability. When an agent object migrates between systems, referenced canonical aliases may resolve in the destination environment to the same policy objects or to authorized successors under established publication, revocation, and anti-rollback controls. This preserves governance continuity while permitting authorized evolution of policy authority without modification of governed agent objects.

[0256] Each destination substrate independently performs resolution, verification, and precondition gating using the portable policy authority and embedded governance state carried by the agent object. Substrates are not required to share centralized services or internal state to enforce governance. If required authority cannot be resolved or verified at the required assurance level in the destination environment, instantiation of an execution context is denied, preventing dilution of enforcement during migration or federation.

[0257] Policy objects may be exchanged across organizational boundaries. An issuing entity may publish policy objects representing ethical, safety, regulatory, contractual, or operational constraints for application by external partners or federated systems. Receiving systems authenticate and apply such policy objects under the applicable trust model. Where required, recognition of authority may be conditioned on quorum artifacts, co-signatures, continuity references, or other verifiable override material.

[0258] Portability includes evidentiary continuity. Policy identifiers, continuity references, revocation state, supersession history, anti-rollback checkpoints, and append-only audit artifacts may accompany migration or remain independently verifiable. A receiving system may validate not only policy content but also provenance and freshness status before authorizing governed actions.

[0259] Portability may be selective. Some policy objects may apply globally across systems, while others remain scoped to particular trust zones, organizational domains, substrate classes, or lineage classes. Scope declarations and context-sensitive alias resolution enable selective portability without ambiguity while preserving deterministic authorization at runtime.

[0260] Destination systems may layer additional local policy objects or meta-policy constraints atop imported authority, provided such constraints are applied through the same resolution and authorization mechanisms. More restrictive constraints may be composed without weakening imported authority.

[0261] Accordingly, governance is portable as a function of externally verifiable policy objects and embedded governance state rather than as a property of any specific system. Agent objects may migrate or operate across multiple environments while remaining subject to consistent, auditable authority conditions, including embodiments in which ethical constraints are enforced as a case of governance.

27. Extensibility for Future Governance Domains and Constraints

[0262] The disclosed architecture supports extensibility of governance domains, constraints, enforcement treatments, and evaluation inputs without modification of agent-object structure, execution substrates, or core authorization logic. Governance is externalized into verifiable policy objects resolved and enforced through deterministic precondition gating, permitting expansion of governance scope independent of internal reasoning or infrastructure implementation. Ethical governance constitutes a class of constraints extensible under the same mechanisms.

[0263] New governance domains are introduced by issuing additional policy objects having defined scope, validity and freshness semantics, enforcement treatment, and applicability to designated action classes, trust zones, substrate classes, or capability categories. Because policy objects are resolved at runtime via canonical aliases, the governance gate need not embed prior knowledge of future domains; it enforces whatever verified authority is applicable when a governed action is proposed.

[0264] Future constraints may address emerging regulatory regimes, evolving safety standards, sector-specific requirements, data-residency rules, environmental controls, supply-chain restrictions, or newly developed autonomous capabilities. Incorporation occurs by publishing corresponding policy objects, associating them with canonical aliases, layering them into required policy sets for selected action classes, and/or modifying scope and precedence under authorized procedures. Agent objects become subject to such domains through alias references, trust-zone baseline requirements, meta-policy constraints, or substrate-enforced policy sets, without internal refactoring.

[0265] Extensibility includes introduction of new enforcement treatments. Policy objects may define additional enforcement semantics governing authorization outcomes, including denial, trust degradation, quarantine eligibility, remediation prerequisites, deferral, throttling, attestation requirements, audit-only treatment, or combinations thereof. Governance gates apply enforcement

treatment as defined by verified policy authority, preserving deterministic permit-or-deny gating while allowing deployment-specific expansion of response types.

[0266] Extensibility further encompasses introduction of new evaluation dimensions and governance-relevant signals. Policy objects may reference new categories of execution feedback, continuity metrics, provenance attestations, audit anchors, trust-zone indicators, or compliance evidence artifacts. Such inputs are incorporated by recording relevant signals in embedded memory or append-only audit records and evaluating them under policy-defined criteria during authorization. The structural authorization pipeline remains unchanged while the set of evaluable inputs evolves.

[0267] Extensibility does not weaken enforcement guarantees. New domains are enforced through the same alias resolution, verification, scope applicability evaluation, freshness and anti-rollback controls, and authorization pipeline. If required authority for a new domain cannot be resolved or verified, if required evaluation inputs are unavailable at the required assurance level, or if freshness or continuity conditions are unsatisfied, authorization fails and non-execution results. The system does not default to permissive behavior due to novelty, absence of prior embedding, or unsupported governance requirements.

[0268] Incremental adoption is supported. New policy objects may coexist with existing authority, with policy-defined precedence, scope partitioning, meta-policy constraints, and quorum-based overrides resolving overlap and enabling staged transition between governance regimes without architectural change.

[0269] Accordingly, the architecture is extensible by design, accommodating expansion of governance scope, enforcement treatment, and evaluative criteria through externally governed policy objects while preserving deterministic precondition gating and substrate-independent enforcement, including embodiments in which ethical constraints are enforced as a case of governance.

28. Exemplary Governance Enforcement Scenarios

[0270] The following exemplary scenarios illustrate operation of governance gating, override processing, trust degradation, continuity validation, and non-execution outcomes. These examples clarify runtime behavior.

[0271] Scenario 1 — Authorized Execution: An agent object proposes instantiation of an execution context. Required canonical aliases are resolved and corresponding policy objects are

verified under the applicable trust model. Scope, validity, freshness, and embedded eligibility conditions are satisfied. Governance gating issues an authorization permit and the execution context is instantiated. Authorization data and governance-relevant feedback are recorded in embedded memory and appended to an audit record.

[0272] Scenario 2 — Unauthorized Mutation: An agent object proposes a mutation affecting a governance-relevant field, including a policy reference. Although a mutation descriptor declares the transformation, an applicable meta-policy prohibits or conditions such modification. Because verified authority does not authorize the mutation, the transition is denied prior to instantiation. No partial mutation occurs. The denial is recorded and may influence subsequent eligibility.

[0273] Scenario 3 — Freshness Failure Under Intermittent Connectivity: An agent object proposes execution in an intermittently connected environment using cached policy objects. A required policy object is expired or fails freshness constraints, and revalidation is unavailable. Governance gating denies instantiation. Upon later revalidation of required authority, a subsequent execution attempt may be re-evaluated without structural modification.

[0274] Scenario 4 — Trust Degradation: Repeated denials or governance-relevant adverse feedback accumulate in embedded memory. Policy-defined thresholds for trust degradation are met. Eligibility is narrowed deterministically, including potential elevation of enforcement class, corroboration requirements, cooldown intervals, action-class restriction, or temporary suspension. The agent object remains extant but operates under stricter authorization conditions derived from recorded governance state.

[0275] Scenario 5 — Quorum-Based Override: A governance constraint requires modification. Authorized participants approve an override under quorum requirements. An override policy object including quorum artifacts and continuity linkage to prior authority is issued and disseminated via alias resolution. Subsequent authorization decisions apply the override within declared scope and validity. Override artifacts and publication events are recorded in append-only audit records.

[0276] Scenario 6 — Propagation Under Quarantine: An agent object proposes creation or activation of a descendant in a new trust domain. Lineage-constrained inheritance evaluation determines that the parent is subject to quarantine or restriction. Because such constraints are inheritable under applicable policy authority, propagation is denied and no descendant is authorized.

[0277] Scenario 7 — Invalid Lineage Continuity: An agent object proposes execution from a state lacking valid continuity linkage to an authorized predecessor, including unauthorized fork or replay. Continuity validation fails. Governance gating denies instantiation notwithstanding otherwise applicable action authorization. Eligibility may be restored only through policy-authorized remediation, if permitted.

[0278] In each scenario, authorization or denial occurs prior to instantiation of execution contexts or completion of governed transitions. Non-execution constitutes a valid, auditable outcome. Governance decisions are derived from verified policy authority, embedded state, and continuity constraints, and persist across substrates and time.

[0279] These scenarios collectively illustrate coherent interaction among governance gating, override succession, trust adjustment, continuity validation, freshness enforcement, and non-execution handling within the disclosed architecture.

29. Definitions

[0280] As used herein, “governance” refers to a deterministic system property in which instantiation of an execution context and other governed state transitions are permitted or denied based on satisfaction of externally governed, verifiable authorization conditions. Governance, as defined herein, does not refer to moral reasoning, subjective judgment, intent interpretation, alignment scoring, introspection scoring, interpretability outputs, or prediction of outcomes. Governance enforcement is implemented as a precondition to action rather than as an advisory, cognitive, or post-hoc evaluative construct. “Ethical” or “ethics,” when used herein, refers to a case of governance in which the externally governed authorization conditions encode safety, compliance, or other normative constraints, and is not a reference to human moral reasoning.

[0281] As used herein, “agent object” refers to a structured, self-describing machine-readable data object carrying state sufficient to support governance evaluation, continuity validation, and eligibility determination across heterogeneous execution substrates. An agent object is not limited to any particular internal cognitive architecture. In some embodiments, an agent object is implemented as a cognition-native semantic agent.

[0282] As used herein, “semantic agent” or “cognition-native semantic agent” refers to an embodiment of an agent object configured as a structured semantic representation comprising a

plurality of independently addressable semantic fields used to represent action proposals and governance-relevant state. In exemplary embodiments, the semantic fields include at least an intent field, a memory field, a policy field, and a lineage field, and may further include a context field and a mutation descriptor field.

[0283] As used herein, "semantic fields" refer to named or otherwise addressable fields, slots, records, or regions within a semantic agent that store values representing distinct semantic categories. Examples include: (i) an intent field storing a representation of a proposed action, task request, capability invocation, or action class; (ii) a memory field storing governance-relevant history and evidence; (iii) a policy field storing one or more canonical aliases and/or policy-set identifiers used to resolve externally maintained policy authority; (iv) a lineage field storing continuity references linking a current state to one or more prior authorized states; (v) a context field storing execution context descriptors such as trust-zone and substrate-class indicators; and (vi) a mutation descriptor field storing declared mutation class, state-transition descriptors, or transformation metadata.

[0284] As used herein, "intent field" refers to a semantic field configured to store an intent representation that includes at least one of: a proposed action identifier, an action class, a task request, an execution request, a mutation request, a delegation request, or a propagation request. In embodiments, the intent field may inform selection of applicable policy authority and scope evaluation, but is not determinative of authorization.

[0285] As used herein, "memory field" refers to a semantic field configured to store governance-relevant history and evidence, including without limitation prior policy resolution outcomes, verification results, authorization decisions, denials, freshness failures, remediation state, quarantine state, trust degradation state, eligibility markers, audit references, and checkpoint information.

[0286] As used herein, "policy field" (also referred to as a "policy reference field") refers to a semantic field configured to store one or more canonical aliases that indirectly reference externally maintained policy authority, including policy objects and meta-policy objects. In embodiments, a policy field may store multiple canonical aliases corresponding to multiple required policy authorities that are independently resolved, filtered for freshness, verified, and evaluated for authorization.

[0287] As used herein, “policy reference” refers to a machine-readable reference usable to identify, locate, select, or resolve externally maintained policy authority applicable to a proposed action, including without limitation one or more canonical aliases, policy-set identifiers, policy object identifiers, policy object fingerprints or hash commitments, permit identifiers, capability-grant identifiers, or combinations thereof. In embodiments, policy references may be stored in a policy field of an agent object, provided by an execution substrate, and/or provided by a governing context associated with a proposed action.

[0288] As used herein, “canonical alias” refers to a stable, authoritative identifier that indirectly references externally maintained policy authority, including one or more policy objects and/or policy-agent embodiments thereof. A canonical alias provides indirection between governed agent objects and policy content such that policy authority may be updated, superseded, revoked, or overridden by publishing successor authority under the same canonical alias without requiring mutation of governed agent objects. Resolution of a canonical alias yields a candidate set of policy objects, and, in embodiments, the candidate set is filtered based on objective eligibility constraints including scope applicability, validity-window satisfaction, revocation state, and anti-rollback monotonicity or continuity requirements to produce a selected set eligible for cryptographic or continuity-based verification. In embodiments, the selected set is evaluated by a governance gate to determine authorization for a proposed action prior to instantiation of an execution context, and failures of resolution, filtering, verification, or authorization result in deterministically enforced non-execution as a valid system outcome.

[0289] As used herein, “policy object” refers to an externally governed, verifiable, immutable-by-default semantic object that defines one or more enforceable constraints, permissions, prohibitions, scopes, validity and freshness conditions, and enforcement treatment applicable to one or more governed actions. A policy object is external to an agent object it constrains, is authenticated through cryptographic mechanisms and/or continuity-based verification mechanisms, and is enforced deterministically as a condition of instantiating an execution context or performing a governed state transition. The term “policy agent” refers to a policy object and emphasizes that the policy object is a first-class semantic authority artifact; a policy agent does not require that the policy object be a running process.

[0290] As used herein, “meta-policy object” refers to a policy object whose scope applies to categories of behavior, architectural invariants, or governance configuration constraints, including

constraints on self-modification of governance-relevant fields, escalation prohibitions, memory integrity requirements, fork and proliferation control, downgrade resistance, revocation enforcement, and policy substitution restrictions. A meta-policy object constrains permissible governance configurations and classes of actions and may be evaluated in addition to, or with higher precedence than, action-specific policy objects. The term "meta-policy agent," if used, refers to a meta-policy object and does not require that the meta-policy object be a running process.

[0291] As used herein, "policy resolution component" refers to one or more modules, services, functions, or distributed processes configured to resolve canonical aliases to obtain a candidate set of externally maintained policy objects and to output one or more selected policy objects. In embodiments, policy resolution includes candidate discovery, scope-aware routing, caching subject to policy, and resolution auditing.

[0292] As used herein, "verification component" refers to one or more modules, services, functions, or distributed processes configured to verify authenticity and integrity of a policy object under an applicable trust model. Verification may include public-key signature verification and/or continuity-based verification mechanisms including memory-resolved identity, trust-slope validation, or continuity-chain validation.

[0293] As used herein, "authorization determination" refers to a deterministic determination, performed prior to enabling performance of a proposed action, including prior to instantiating, activating, or admitting use of an execution context or capability context, of whether a proposed action is permitted under verified policy authority, including evaluation of policy constraints, scope applicability, validity-window satisfaction, freshness constraints, and any lineage continuity requirements.

[0294] As used herein, "execution" refers to instantiation, invocation, or performance of computational activity associated with an agent object on an execution substrate, including running, evaluating, or invoking behavior. Execution is a governed action that is permitted to occur only when required verified authority and eligibility conditions are satisfied. Absence of execution due to lack of authorization constitutes a valid and complete system outcome.

[0295] As used herein, "execution context" refers to a substrate-instantiated context, session, sandbox, container, process context, runtime environment, interpreter state, capability context, or equivalent operational context in which a proposed action is performed. In embodiments, the

governance gate prevents instantiation of an execution context when required authorization conditions are unsatisfied.

[0296] As used herein, “governing context” refers to a machine-readable context record, envelope, or metadata associated with a proposed action and/or execution request that specifies governance-relevant information, including applicable policy references, trust zone identifiers, substrate class identifiers, tenant or organizational governance bindings, capability grants, or evaluation-time parameters used for resolution, verification, and authorization.

[0297] As used herein, “mutation” refers to any transformation of an agent object’s internal state, structure, configuration, memory, policy references, lineage, or capability declarations, including self-modification, state updates with governance impact, creation or activation of descendants, and alteration of execution parameters. Mutation is a governed state transition and is not instantiated absent verified authorization under applicable policy authority.

[0298] As used herein, “delegation” refers to assignment, transfer, or invocation of a task, authority, or action request by one agent object to another entity, including another agent object, a service, or a substrate component, where such assignment affects governance-relevant authority boundaries. Delegation is treated as a governed action subject to authorization and, where applicable, lineage and inheritance constraints.

[0299] As used herein, “propagation” refers to transmission, replication, instantiation, movement, migration, export, import, or reconstitution of an agent object or governance-relevant state across execution substrates, environments, trust domains, or organizational boundaries. Propagation is treated as a governed state transition subject to authorization, freshness controls, and continuity validation.

[0300] As used herein, “lineage” refers to a verifiable continuity record linking an agent object’s current state to one or more prior authorized states. Lineage supports enforcement decisions by enabling validation of authorized descent, inheritance of governance constraints, and detection of unauthorized forks, replayed snapshots, or discontinuities. Lineage, as defined herein, does not itself assert liability, responsibility, or intent semantics.

[0301] As used herein, “lineage continuity” refers to satisfaction of one or more policy-defined continuity constraints indicating that a proposed mutation, delegation, or propagation originates from

an authorized trust path and is consistent with required ancestor references, continuity markers, or inheritance rules.

[0302] As used herein, “unresolved lineage fork” refers to a lineage condition indicating multiple competing or inconsistent continuity branches, missing required ancestor references, or failure to establish a single authorized continuity path as required by policy authority, such that authorization is denied until the condition is remediated or an authorized override is verified.

[0303] As used herein, “execution substrate” refers to any computational environment capable of evaluating, hosting, or instantiating execution for an agent object, including cloud-based systems, edge devices, federated nodes, and intermittently connected environments. An execution substrate does not independently grant execution authority and instantiates execution only upon satisfaction of verified authorization conditions as evaluated by a governance gate.

[0304] As used herein, “trust zone” refers to a declared operational domain or boundary used to scope applicability of policy authority and enforcement treatment, including organizational domains, security domains, tenancy boundaries, network segments, device classes, or other bounded governance domains.

[0305] As used herein, “substrate class” refers to a declared classification of an execution substrate used for scope applicability, including without limitation cloud, edge, federated, decentralized, hardware-backed, sandboxed, trusted-execution, intermittently connected, or other policy-relevant substrate categories.

[0306] As used herein, “lineage class” refers to a classification of lineage or descent type used for scope applicability and continuity constraints, including without limitation first-party descendants, delegated descendants, replicated descendants, imported or reconstituted instances, or other policy-relevant lineage categories.

[0307] As used herein, “governance gate” refers to a logical enforcement checkpoint configured to determine, prior to enabling performance of a proposed action, including prior to instantiating, activating, or admitting use of an execution context or capability context or other governed transition, whether applicable policy authority has been resolved, filtered for freshness, verified, and determined to authorize a proposed action under scope, validity, freshness, and continuity constraints, and to deterministically permit or deny the proposed action based on that determination.

[0308] As used herein, “non-execution” refers to a deterministically enforced system outcome in which instantiation of an execution context and/or a governed state transition does not occur due to unsatisfied authorization, freshness, or continuity conditions. Non-execution is a valid, verifiable, and auditable outcome and is not indicative of system failure, error, or malfunction.

[0309] As used herein, “denial result” refers to a structured output of a governance gate corresponding to a denial outcome, the denial result comprising at least an identifier of the proposed action or action class and a failure basis indicator. In embodiments, the denial result further comprises identifiers or fingerprints of evaluated policy objects and/or revocation artifacts and is configured to be stored in a memory field of an agent object and/or recorded in an append-only audit record.

[0310] As used herein, “append-only audit record,” “append-only audit ledger,” or “append-only audit log” refers to a tamper-evident record configured to record governance-relevant events including policy resolution outcomes, candidate filtering outcomes, verification results, authorization decisions, denials, override events, non-execution outcomes, freshness failures, and checkpoint updates.

[0311] As used herein, “override” or “authorized override” refers to an authorized supersession of a prior policy object by a replacement policy object in accordance with policy-defined override rules, including approval by a plurality of authorized participants and/or satisfaction of a quorum requirement.

[0312] As used herein, “signature chain” refers to a verifiable continuity linkage between a replacement policy object and a prior policy object, including cryptographic parent references, chained signatures, or equivalent continuity references establishing authorized succession across one or more override generations.

[0313] As used herein, “freshness” refers to satisfaction of policy-defined temporal and/or monotonicity conditions for treating a policy object as authoritative, including notBefore/notAfter validity windows, revocation state, and anti-rollback constraints such as monotonic versioning, continuity references, hash commitments, or latest-known-good checkpoints, as evaluated during resolution and authorization.

[0314] As used herein, “revocation” refers to negative authority that renders a policy object non-authoritative notwithstanding authenticity, including revocation artifacts such as revocation lists, revocation anchors, revocation epochs, supersession records, or equivalent verifiable revocation indicators.

[0315] As used herein, “anti-rollback” refers to constraints that prevent authorization based on stale or superseded authority when a newer authorized successor exists or when a minimum acceptable authority checkpoint has been recorded, including monotonic version counters, continuity references between authority instances, hash commitments, or latest-known-good constraints.

[0316] As used herein, “stale-policy downgrade attempt” refers to a condition in which a candidate policy object for a canonical alias is older than a policy-defined minimum acceptable version or older than a latest-known-good checkpoint, or lacks required continuity linkage to an authorized successor, such that authorization is denied under anti-rollback constraints.

[0317] As used herein, “trust degradation” refers to a deterministic restriction of eligibility, scope, or enforcement treatment applied to an agent object based on recorded governance outcomes, policy-defined criteria, and/or accumulated governance-relevant signals, including denials, freshness failures, remediation non-satisfaction, or continuity anomalies. Trust degradation is not a subjective reputation score and does not depend on inferred intent.

[0318] As used herein, “quarantine” or “semantic quarantine” refers to a governance state in which an agent object is structurally prevented from instantiating execution contexts and/or performing specified governed action classes regardless of substrate availability, until authorized remediation occurs, a policy-defined quarantine condition expires, or an authorized override is verified and applicable.

[0319] As used herein, “entropy” refers to a quantitative measure of uncertainty or unpredictability associated with a value, distribution, or process. In embodiments, entropy may be expressed in bits relative to a defined probability model, or may refer to an equivalent uncertainty metric used to assess uniqueness, variability, or resistance to guessing, without implying any particular estimator unless expressly stated.

[0320] As used herein, “about” when modifying a value, condition, or range refers to the stated value, condition, or range plus or minus an amount that accounts for ordinary measurement

variability, system tolerances, and/or implementation-specific variation consistent with the described technology. In embodiments, “about” includes variations that do not materially affect the operation of the described systems and methods for the stated purpose.

[0321] As used herein, “real-time” and “near real-time” refer to operation in which processing, verification, authorization, or enforcement occurs with sufficiently low latency to be effective for the intended control point, including precondition gating prior to instantiation of an execution context. “Real-time” does not require instantaneous operation, and “near real-time” includes bounded delays, buffering, batching, or propagation lag that remain consistent with deterministic enforcement of the described authorization conditions.

[0322] As used herein, “enable performance” refers to permitting a proposed action to proceed by at least one of instantiating an execution context, activating a pre-existing execution context, admitting use of a capability context, issuing or honoring an authorization permit, or otherwise authorizing the execution substrate to perform the proposed action.

What is claimed is:

1. A computer-implemented method for cryptographically enforced governance of an autonomous agent, comprising:

   receiving, at an execution substrate, a proposed action associated with an agent object, the proposed action being associated with one or more policy references including one or more canonical aliases provided by at least one of the agent object, the execution substrate, or a governing context;

   resolving the one or more policy references to obtain candidate external policy objects;

   filtering the candidate external policy objects based on one or more freshness constraints including at least one of a validity window, a revocation state, or an anti-rollback monotonicity constraint;

   verifying authenticity of at least one external policy object remaining after the filtering using cryptographic verification;

   determining, prior to enabling performance of the proposed action, including prior to instantiating, activating, or admitting use of an execution context or capability context, whether the proposed action is authorized under the verified external policy object; and

   permitting the execution substrate to enable performance of the proposed action only if authorized, otherwise deterministically denying the proposed action as a valid non-execution outcome.

2. The method of claim 1, wherein determining whether the proposed action is authorized comprises evaluating scope metadata that defines applicability of permissions or prohibitions to at least one of an action class, an execution substrate class, a trust zone, or a lineage class.

3. The method of claim 1, wherein determining whether the proposed action is authorized comprises evaluating a lineage continuity requirement for the agent object prior to permitting mutation, delegation, or propagation, and denying the proposed action upon failure of the lineage continuity requirement.

4. The method of claim 1, further comprising recording, in an append-only audit record, an entry corresponding to at least one of the resolution, the freshness filtering, the cryptographic verification, the authorization determination, or the denial.

5. The method of claim 1, wherein denying comprises refusing to instantiate the execution context and emitting a denial result configured to be stored in a memory field of the agent object.
6. A non-transitory computer-readable medium storing instructions that, when executed by one or more processors of an execution substrate, cause performance of the method of claim 1.
7. The non-transitory computer-readable medium of claim 6, wherein the instructions further cause denial of the proposed action upon detection of at least one of an unauthorized override attempt, a stale-policy downgrade attempt, or an unresolved lineage fork.
8. A system for cryptographically enforced governance of cognition-native semantic agents, comprising:

   a semantic agent object comprising semantic fields including at least an intent field, a memory field, a lineage field, and a policy field, wherein the policy field comprises one or more policy references including one or more canonical aliases;

   a policy resolution component configured to resolve, at runtime, the one or more policy references stored in the policy field to obtain one or more policy objects external to the semantic agent object;

   a verification component configured to verify authenticity and validity of the one or more policy objects using cryptographic verification; and

   a governance gate operatively coupled to an execution substrate and configured to, prior to enabling performance of a proposed action of the semantic agent object, including prior to instantiating, activating, or admitting use of an execution context or capability context for the proposed action, deterministically permit or deny the proposed action based on at least the verification and an authorization determination under the one or more policy objects,

   wherein the proposed action comprises at least one of execution, mutation, delegation, or propagation, and wherein denial of the proposed action by the governance gate results in non-execution as a valid system outcome.
9. The system of claim 8, wherein the one or more policy objects are external to the semantic agent object and are immutable absent an override authorized under the one or more policy objects.
10. The system of claim 8, wherein the authorization determination is performed independently of inferred intent scoring, model alignment scoring, predicted outcomes, introspection, or interpretability outputs associated with the semantic agent object.

11. The system of claim 8, wherein the governance gate denies the proposed action without enabling performance of the proposed action, including without instantiating, activating, or admitting use of the execution context, when the cryptographic verification fails.
12. The system of claim 8, wherein the verification component is configured to evaluate freshness constraints of the one or more policy objects, including at least one of a validity window, a revocation state, or an anti-rollback monotonicity constraint, and wherein the governance gate denies the proposed action upon a freshness failure.
13. The system of claim 8, wherein the one or more policy objects include scope metadata defining applicability of permissions or prohibitions to at least one of an action class, an execution substrate class, a trust zone, or a lineage class.
14. The system of claim 8, wherein the governance gate is configured to deny a mutation or propagation action when the lineage field fails to demonstrate continuity under a lineage constraint defined by the one or more policy objects.
15. The system of claim 14, wherein a descendant semantic agent object of the semantic agent object inherits at least one constraint defined by the one or more policy objects and applicable to an ancestor semantic agent object identified in the lineage field.
16. The system of claim 8, further comprising an override mechanism requiring approval by a plurality of authorized participants to supersede a prior policy object with a replacement policy object.
17. The system of claim 16, wherein the replacement policy object includes a cryptographic signature chain linking the replacement policy object to the prior policy object.
18. The system of claim 8, further comprising an append-only audit record configured to record governance-relevant events including at least policy resolution outcomes, verification results, authorization decisions, denials, override events, and non-execution outcomes.
19. The system of claim 8, wherein the policy resolution component is further configured to resolve a plurality of policy references including at least two canonical aliases associated with different governance domains, and to apply a deterministic precedence and combination rule to generate a composite authorization determination, including at least one of conjunctive authorization requiring joint approval, hierarchical precedence based on trust zone or governance tier, or quorum-based satisfaction of required permissions prior to permitting the proposed action.
20. The system of claim 18, wherein the governance gate is further configured to, upon denial of the proposed action, generate a structured denial artifact comprising at least an identifier of

each evaluated policy object, a verification result indicator, a freshness evaluation indicator, and a failure basis indicator, and to associate the structured denial artifact with at least one of the semantic agent object, the append-only audit record, or a governing context for machine-verifiable downstream enforcement.

Abstract

Systems and methods are disclosed for cryptographically enforced governance of autonomous agents operating across distributed execution environments. Governance authority is externalized into policy objects that are resolved via canonical aliases and cryptographically verified prior to permitting execution, mutation, delegation, or propagation. A governance gate deterministically conditions instantiation of an execution context on successful verification and on authorization under applicable policy objects, wherein failure results in non-execution as a valid system outcome. The disclosed architecture supports freshness, revocation, and anti-rollback controls to prevent reliance on stale or superseded policy authority, lineage-constrained inheritance of constraints across agent evolution, quorum-based policy overrides with signature-chain continuity, and append-only audit records of governance-relevant events. Embodiments enable consistent enforcement across heterogeneous substrates, including cloud, edge, federated, and intermittently connected environments, and may support identity verification without persistent keypairs using continuity-based validation.