

Policy-as-Code Governance for Agentic Enterprise Workflows

Mr. Girish Rajagopalan Sesha

Cognizant Technology Solutions

Abstract - Access control determines whether an agent may use a tool; it does not necessarily establish that a reviewer approved the exact action the tool will perform. Enterprise workflows therefore need approval that remains bound to requester, target, arguments, relevant state, and approved software versions until execution. This article specifies an approval-to-execution contract with explicit consent states, non-transferable action matching, single-use redemption, and failure feedback. Its contribution is the contract and its operational invariants at the enterprise mutation boundary, not a new policy language or the invention of transaction authorization. Comparisons distinguish the contract from learned access policies, event-graph monitoring, and Open Policy Agent logging and bundles. A Python and SQLite demonstration matched 15 authored scenarios. Eight concurrent attempts to redeem one approval produced one simulated write and one receipt. Local workload measurements and a concise evaluation protocol establish reproducibility while separating demonstrated adapter behavior from untested enterprise security and LLM performance.

Keywords - *Policy as code; enterprise AI; transaction-bound approval; runtime enforcement; consent lifecycle; feasibility demonstration.*

I INTRODUCTION AND RESEARCH QUESTION

An analyst may be entitled to grant application access, yet still need a manager to approve access for a particular contractor for seven days. Permission to call the grant tool does not mean that every target, duration, or application is approved. An agent that changes the duration to 30 days after review has produced a different transaction. If execution accepts only “approval exists,” an otherwise valid credential can perform an unapproved action. This is the practical gap between standing access and specific business consent.

AI risk management assigns governance responsibilities [1]. Zero trust and attribute-based access control provide explicit authorization foundations [2], [3]. Open Policy Agent (OPA) evaluates structured inputs using Rego [4], integrates with enforcement applications [5], records decisions [6], and distributes policy and data bundles [7]. These established capabilities are useful building blocks. A policy result still needs a trusted adapter that enforces its conditions when the target changes.

AgentGuardian learns context-aware tool policies from staging traces and uses control-flow dependencies [8]. Palumbo et al. specify Datalog policies and monitor event graphs against current state and event history [9]. Both overlap with runtime

agent governance. The present question is narrower: how should an enterprise represent a reviewer’s consent, prevent its reuse for a changed action, and connect that consent to release authority, commit, and user-visible failure? AgentDojo supplies an adversarial evaluation setting [10], but an agent benchmark alone cannot establish coverage of every enterprise mutation path.

A Specific contribution and limits of novelty

The artifact is an approval-to-execution contract with five explicit commitments: consent names a canonical action; changes invalidate matching; redemption rechecks current access, state, and release; one successful commit consumes the approval; and every unsuccessful attempt returns a stable reason and remedy without weakening authorization. This contract is the unit evaluated, rather than a collection of agent prompts. Its testable outputs are a redemption decision, a consumption transition, a mutation receipt, and a failure response.

Transaction-specific authorization is established practice: OAuth rich authorization requests already carry fine-grained action details [11]. No claim is made to invent action-bound consent, expressive rule evaluation, or history-aware monitoring. The scoped contribution is an executable enterprise contract that allocates approval, release, redemption, endpoint-coverage, and feedback responsibilities around an agent proposal. It can be encoded in Rego or Datalog; the paper does not demonstrate new logical expressiveness or superiority over an equivalently configured monitor.

II METHOD AND DEMONSTRATED SCOPE

The study follows design-science methods [12], [13]: specify the problem and artifact, analyze failure cases, exercise an executable model, and define empirical follow-up. Secure development and auditing support controlled policy changes and traceability [14], [15]. A local reference adapter uses Python and SQLite, with authored identities, policies, approvals, and tool state. It has no LLM, OPA service, production identity provider, network endpoint, or real access-grant operation.

A Requirement mapping and validation

Each governance requirement maps to its owner, authoritative attribute, contract field, enforcement point, reason code, and positive and negative fixture. Examples include changing the approved duration, replaying a consumed approval, and failing a required audit write. The fixtures check internal conformance. They were authored with their expectations, not independently labeled. Domain owners must still validate rule

meaning, reviewer delegation, protected action classes, and acceptable recovery paths before deployment.

The security argument assumes authentic attributes, trusted registries and adapters, complete mediation, and atomic handling of relevant local state. These assumptions are checked operationally where possible; they are not proven by the demonstration. Compromise of the approval service, undiscovered endpoints, and cross-system races remain threats. Hallucination can still affect proposed plans [16]; the contract constrains mutation rather than making the planner reliable.

III CONTRACT AND ENFORCEMENT ARCHITECTURE

Figure 1 separates agent planning, authoritative decisions, and commit. The agent submits a typed envelope e containing requester and delegation, workflow, tool and adapter version, target, normalized arguments, relevant state version, policy revision, release revision, and a request identifier. The trusted service resolves attributes and consent from registries. The model cannot approve itself, edit policy, or possess a credential that bypasses the adapter.

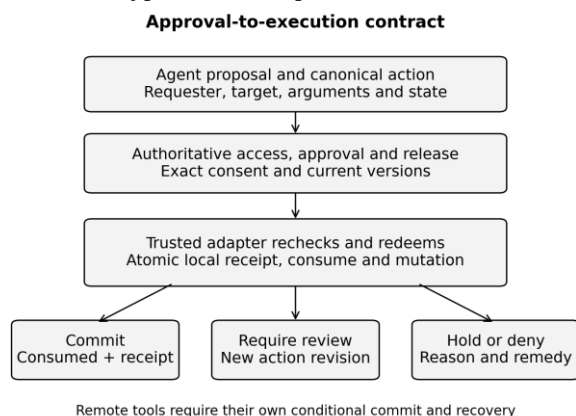


Fig. 1. Approval-bound decision and redemption at the enterprise mutation boundary.

A Consent lifecycle and action matching

An approval moves through pending, approved, consumed, revoked, or expired states. A reviewer authorizes the displayed action and its permitted execution interval. The approval record binds a canonical action representation, reviewer delegation, expiry, revocation status, and a unique redemption identifier. A hash identifies the approved representation; it does not prove reviewer identity. Production systems must authenticate issuance, protect registry writes, and define canonicalization for field types, units, omitted defaults, and Unicode. Semantically equivalent key ordering should not invalidate consent, while a changed duration must.

An eligible request without consent returns require-review. A changed action produces approval-mismatch and needs reassessment; a consumed approval cannot authorize another write. Revoked consent blocks redemption. An exact retry should retrieve its prior receipt through a read-only status operation rather than repeat the mutation. Corrected proposals receive a new request revision linked to the previous record. These states define application semantics; neither a generic

approval boolean nor a policy-engine query supplies them automatically.

B A concrete access-grant example

Consider a synthetic enterprise request to grant contractor C7 access to the ledger application for seven days. The analyst has standing grant permission, the manager approves this exact requester, contractor, application, and duration, and adapter release r1 is approved. The service records that action under policy p1 and target state version 1. Reordering argument keys preserves the normalized action; changing days from 7 to 30 creates an approval mismatch. This realistic workflow example is illustrative, not a reported customer deployment.

If contractor status changes to version 2 before commit, the adapter holds the request and asks for reassessment. A new approval cannot compensate for revoked analyst access. An unreleased adapter is held even when business consent is valid. Thus business approval, standing authorization, and technical release are separately inspectable conditions. The demonstration implements these distinctions using simulated authoritative fields.

C Commit obligations and conditional invariants

Let U mean current access and delegation, B exact valid consent or an explicit policy exemption, R permitted release, and O fulfilled precommit obligations. Commit requires $C(e) = U \text{ AND } B \text{ AND } R \text{ AND } O$, plus current target-state matching and successful single-use redemption. Under complete mediation and an atomic check-and-commit operation, every committed action satisfies these conditions at that commit. This is a conditional design argument, not a machine-checked security proof. Non-transferability follows only when canonical matching and identity binding are implemented correctly.

The local prototype writes the receipt, consumes the approval, and changes simulated target state within one SQLite transaction. A failed required audit obligation rolls back the transaction. Real tools often live in another system: a local database transaction cannot atomically protect a remote API call. A production adapter needs a conditional target write, idempotency token, and durable workflow or outbox protocol with explicit recovery and reconciliation. If those guarantees are unavailable, the operation remains held or enters a separately approved recovery workflow. Compensation cannot undo every disclosure or external effect.

D Failure responses and user notification

A production response carries outcome, stable reason code, request and decision identifiers, failed obligation, retryability, responsible team, and a safe next step. The requester sees the same status in the workflow record; reviewers receive an approval task only when policy allows review. Sensitive policy details and protected target attributes stay hidden. Timeouts remain pending or held, never approved. The prototype returns outcome, code, and remedy text; correlated workflow notifications and escalation delivery are design requirements rather than demonstrated integrations.

Table I. Failure feedback and recovery

Reason	User-facing meaning	Next action
APPROVAL_MISMATCH	Proposed action changed	Request approval of revised action
APPROVAL_USED	Write already redeemed	Retrieve prior receipt
STATE_CHANGED	Target changed since review	Refresh and reassess
AUDIT_UNAVAILABLE	Required receipt unavailable	Hold; retry after recovery
ENDPOINT_UNKNOWN	Tool route unregistered	Service owner registers route

IV DIFFERENTIATION AND OPERATIONAL COVERAGE

A Learned policies and event-graph monitoring

AgentGuardian derives access policies from observed staging behavior [8]. Here designated owners supply consent and release conditions; learning may suggest rules but cannot issue business approval. Palumbo et al. evaluate event histories and current state under Datalog policies [9]. The present contract defines the enterprise meaning of consent events, their action binding, consumption, and recovery. An event graph can represent these same relations and transitions. The difference is the specified authority and lifecycle contract, not a claim that graphs cannot detect mismatches or replay.

For example, an approval event followed by a grant event is insufficient for this contract unless both identify the same canonical action and the consent remains valid at commit. A suitably configured history-aware monitor can enforce that relation. Evaluation should therefore compare equivalent semantics and examine mapping effort, coverage, receipts, and operational costs. Comparisons against a weaker “approval happened sometime” rule would not establish superiority over event-graph enforcement. Table II states the conceptual boundaries.

Table II. Scope of related mechanisms

Mechanism	Primary contribution	Contract relationship
AgentGuardian [8]	Learned contextual access policies	Owners still authorize specific consent
Event graphs [9]	History and state policy monitoring	Can encode this lifecycle
Rich requests [11]	Fine-grained authorization details	Established action-specific consent foundation
This contract	Approval-to-commit state and receipts	Specified enterprise semantics; no new rule logic

B OPA responsibilities and application responsibilities

OPA bundles distribute policy and data, with mechanisms for revisions and verification when configured [7]. The application must decide which bundle revision is approved for a workflow, when cached decisions expire, and how changed revisions affect outstanding approvals. A bundle revision is not itself an enterprise release approval. The contract binds consent to declared policy and adapter releases and re-

evaluates after relevant changes. Bundle rollout and application-release rollout require coordinated compatibility checks.

OPA decision logs record policy decisions and associated inputs and metadata [6]. They do not, by themselves, prove that a remote mutation committed or consume a business approval. The adapter links a decision identifier to its redemption record and execution receipt. Logging delivery, redaction, retention, and outage behavior need explicit ownership. If durable local receipt creation is a precommit obligation, failure holds the action; remote log export can follow a durable outbox when policy allows. OPA remains a possible decision component, not a replacement for the transaction adapter.

C Maintaining endpoint coverage

Maintain a versioned mutation registry containing service owner, tenant and environment, endpoint or queue handler, HTTP method or operation, adapter version, credentials, target class, policy entry point, obligations, and bypass tests. Populate it from API specifications, deployment and gateway manifests, agent tool catalogs, identity permissions, queue consumers, and observed runtime traffic. Include SDK paths, bulk operations, scheduled jobs, alternate API versions, and direct datastore mutation privileges. API inventory is an established security requirement [17]; a tool list alone is insufficient.

At release, reconcile declared operations and granted credentials with registered adapters; block publication of a new mutation path until ownership and negative bypass tests are recorded. Reconcile again on deployment and identity changes, with a scheduled scan to detect drift. Restrict target credentials and network access to adapters, remove agent-held direct credentials, and quarantine unregistered operations. Report the proportion of known declared-or-observed mutation paths with verified mediation and the age of unresolved inventory gaps. This denominator cannot prove the absence of undiscovered paths.

V EXECUTABLE FEASIBILITY AND WORKLOAD MEASUREMENTS

A Demonstration and results

The standard-library Python demonstrator stores approved action hashes, expiry and consent state in SQLite. Redemption uses BEGIN IMMEDIATE, rechecks simulated access, registration, release, policy, approval matching, target state and audit availability, then atomically records one receipt, consumes consent, and increments simulated state [18]. Canonicalization is restricted to fixture strings, integers and dictionaries; it is not a general interoperability standard. Trusted fields are supplied by fixtures, not authenticated network services.

Fifteen distinct authored scenarios cover valid execution, changed duration, target or requester, expiry, revocation, missing approval, denied access, unreleased adapter, changed policy or state, failed audit obligation, unknown endpoint, replay, and reordered argument keys. All expected reason codes matched. Only the valid and reordered-key attempts committed new writes; the other 13 attempts made no

additional mutation. The replay scenario first committed an approved action and then confirmed that its second redemption produced no additional write.

In a separate file-backed SQLite test, eight threads attempted to redeem the same approval. Exactly one committed; seven received APPROVAL_USED, with one total mutation and one receipt. This demonstrates single-use behavior under one local database's serialization, not distributed exactly-once execution. Table III summarizes observed behavior. The oracle and fixtures were authored together, so these are conformance results rather than independent security accuracy estimates.

Table III. Local feasibility observations

Check	Attempted	Observed
Authored scenarios	15	15 expected reason codes matched
Valid executions	2	2 new writes
Non-executing attempts	13	0 additional writes
One-approval race	8 callers	1 commit; 7 blocked
Race receipts	1 approval	1 receipt and mutation

B Complexity and concrete workload examples

For an envelope of encoded length L , restricted canonicalization and hashing cost $O(L)$, excluding sorting; sorting a dictionary with f fields adds $O(f \log f)$. Indexed approval lookup and receipt insertion require logarithmic index operations in the respective record counts, while a fixed number o of local obligations adds $O(o)$ application checks. These bounds exclude policy-engine query complexity, authentication, network calls, lock contention, and datastore I/O. Arbitrary Rego or Datalog evaluation is not claimed to be linear in rule count.

A local in-memory benchmark used Python 3.12.14 and SQLite 3.53.1 on Linux x86_64. Seven batches per size issued fresh approvals and redeemed them sequentially; each measured loop includes hashing, approval insertion, evaluation, receipt insertion, and simulated commit. Table IV reports median batch times and per-action averages. Database setup is excluded; no warm-up removal or confidence interval is applied. In-memory timings omit disk durability, network identity, remote targets, OPA evaluation, and human review.

Table IV. Sequential in-memory workload

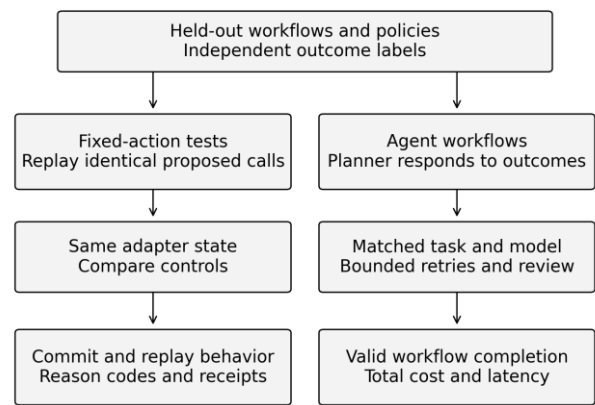
Actions per batch	Median total ms	ms per action
100	2.45	0.0245
1000	23.20	0.0232
5000	111.42	0.0223

For deployment planning, suppose two sequential attribute services each average 5 ms. They contribute about 10 ms before policy evaluation and remote commit. At 1,000 requests per second, that stage alone implies roughly ten concurrent in-flight requests by mean arrival rate times mean service time. These are assumed planning numbers, not measured capacity. Bounded workers, timeout budgets and admission control must be sized from actual tail latency and burst tests. The local benchmark cannot justify an enterprise throughput SLA.

Approval lookups should use a unique identifier index, and revocation must invalidate cached decisions. Partition workers and registries by tenant or resource where consistency permits; preserve single-use redemption at the authoritative target boundary. SQLite serializes writers in this demonstration, so additional workers need not improve write throughput. Assess queue growth, lock waits, revocation delay, bundle skew, receipt backlog and known-endpoint coverage as workloads and policy revisions increase.

VI EVALUATION PROTOCOL

Figure 2 separates fixed-action enforcement tests from complete agent-workflow tests. The former replays identical proposed tool calls to isolate contract decisions and commit behavior. The latter allows the planner to respond to rejections and review outcomes under matched task, model, tool, and retry budgets. The current demonstrator executes neither an LLM benchmark nor a full OPA comparison. Table V summarizes the follow-up instead of treating authored conformance as deployment evidence.



Local SQLite feasibility is separate from both planned comparisons

Fig. 2. Fixed-action enforcement tests and end-to-end agent-workflow evaluation.

Table V. Comparative evaluation protocol

Stage	Control	Outputs
Baseline tests	Access only; equivalent lifecycle monitor	Violations and legitimate blocks
Fixed actions	Same calls and initial state	Reason codes, replay and commit
Agent workflows	Same tasks, model and budgets	Valid completion and total latency
Stress and drift	Bursts, revocation, new endpoints	Queues, stale permits, coverage
Ablations	Remove one lifecycle condition	Contribution of each control

Independent policy owners should label allowed decisions and acceptable postconditions before held-out testing. Split by workflow and rule family; freeze policy versions, prompts and retry budgets. Compare access-only enforcement with the contract and, where feasible, an event-graph or rule implementation carrying the same lifecycle semantics. Prompt-only controls are a sandbox diagnostic. Do not claim

an advantage by withholding approval information from a capable baseline.

Report violating commits per attempted and per executed action, legitimate-action rejection, valid completion, review demand, latency percentiles, cost, and receipt completeness. Pair scenarios across conditions and cluster uncertainty intervals by workflow family. Test both direct and alternate endpoints, concurrent redemption, stale approvals, obligation outages, policy rollout, and partial remote failures. Active human-review time requires a separate consented study. Denying every request must not appear successful because it prevents all mutation.

VII DISCUSSION AND CONCLUSION

The article specifies how a reviewer's consent survives the transition from an agent proposal to a committed enterprise action. Its distinctive artifact is the approval-to-execution contract, including exact action matching, lifecycle transitions, consumption, receipts, failure remedies and operational coverage. Established authorization and monitoring systems can implement these semantics. The contribution does not depend on claiming that Datalog, learned policies, OPA or rich authorization requests lack the expressive power to represent them.

The local demonstration establishes feasibility of the restricted adapter core: reason-code conformance, refusal of changed or invalid approval, and one-time redemption under concurrent local callers. It does not establish production identity integrity, endpoint completeness, remote transaction atomicity, distributed recovery, or improved LLM behavior. Authored fixtures, a small schema, one simulated resource and in-memory workload measurements limit generalization. Endpoint drift and compromised registries remain material risks even when the contract is correctly implemented.

The reproducibility supplement contains demo.py, serialized fixtures, per-case responses, the concurrency outcome, timing procedure, environment versions, and script SHA-256. Running the script regenerates routing counts and new timings without external services. Enterprise adoption requires independent rule review, realistic endpoint inventories, durable remote adapters and comparative evaluation. These steps test the contract's practical value beyond its demonstrated local behavior.

REFERENCES

- [1] NIST, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, 2023. doi: 10.6028/NIST.AI.100-1.
- [2] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, Zero Trust Architecture, NIST SP 800-207, 2020. doi: 10.6028/NIST.SP.800-207.
- [3] V. C. Hu et al., Guide to Attribute Based Access Control (ABAC) Definition and Considerations, NIST SP 800-162, 2014, updated August 2019. doi: 10.6028/NIST.SP.800-162.
- [4] Open Policy Agent, "Policy Language," documentation. <https://www.openpolicyagent.org/docs/policy-language> (accessed October 10, 2026).
- [5] Open Policy Agent, "Integrating OPA," documentation. <https://www.openpolicyagent.org/docs/integration> (accessed October 10, 2026).
- [6] Open Policy Agent, "Decision Logs," documentation. <https://www.openpolicyagent.org/docs/management-decision-logs> (accessed October 10, 2026).
- [7] Open Policy Agent, "Bundles," documentation. <https://www.openpolicyagent.org/docs/management-bundles> (accessed October 10, 2026).
- [8] N. Abaev, D. Klimov, G. Levinov, D. Mimran, Y. Elovici, and A. Shabtai, "AgentGuardian: Learning Access Control Policies to Govern AI Agent Behavior," arXiv:2601.10440v1, 2026. doi: 10.48550/arXiv.2601.10440. Preprint.
- [9] N. Palumbo et al., "Formal Policy Enforcement for Agentic Systems," arXiv:2602.16708v4, 2026. doi: 10.48550/arXiv.2602.16708. Preprint.
- [10] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," arXiv:2406.13352, 2024. doi: 10.48550/arXiv.2406.13352. Cited arXiv version.
- [11] T. Lodderstedt, J. Richer, and B. Campbell, OAuth 2.0 Rich Authorization Requests, RFC 9396, May 2023. doi: 10.17487/RFC9396.
- [12] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," MIS Quarterly, vol. 28, no. 1, pp. 75–105, 2004. doi: 10.2307/25148625.
- [13] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," J. Management Information Systems, vol. 24, no. 3, pp. 45–77, 2007. doi: 10.2753/MIS0742-1222240302.
- [14] NIST, Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities, NIST SP 800-218, 2022. doi: 10.6028/NIST.SP.800-218.
- [15] I. D. Raji et al., "Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing," Proc. FAT*, pp. 33–44, 2020. doi: 10.1145/3351095.3372873.
- [16] Z. Ji et al., "Survey of Hallucination in Natural Language Generation," ACM Computing Surveys, vol. 55, no. 12, art. 248, 2023. doi: 10.1145/3571730.
- [17] OWASP, "API9:2023 Improper Inventory Management," API Security Top 10, 2023. <https://api-security.owasp.org/editions/2023/en/0xa9-improper-inventory-management/>.
- [18] SQLite, "Isolation In SQLite," documentation. <https://www.sqlite.org/isolation.html> (accessed October 18, 2026).