

Cognous Control Plane: Bounded Authorization, Effect-Time Revalidation, and Durable Records for AI-Agent Workflows

André de Lima
Cognous
Seattle, Washington
`andredelima@cogno.us`

October 2026

Abstract

AI agents increasingly operate through tools, APIs, databases and enterprise systems. This creates a runtime control problem that ordinary agent frameworks do not solve: before an agent's proposed action takes effect, an enterprise needs to know which declared action it corresponds to, whether independently supplied institutional authority permits it, whether that authority is still current at the moment of effect, what the executor attempted, and what the destination was independently observed to contain afterwards. Cognous Control Plane is a public Python reference implementation for that layer. Its bounded authorization workflow binds a proposal to an exact Cognous Action Manifest 1.1 declaration, resolves Authority Context through a trusted resolver interface, records a runtime decision, revalidates decision-critical inputs immediately before dispatch, and keeps decisions, attempts, observations and reconciliation as distinct durable records under a locked local record store. It retains the original run-record toolkit: a deterministic policy gate with evaluation traces, blocked-action, authority and reliance records, replay bundles, semantic validation, redaction, HMAC-signed export and a command-line interface. This whitepaper specifies the component's position in the 13-component Cognous Open Control Stack, its object model, authorization and revalidation semantics, the default C0 boundary and its residual check-to-commit race, the optional same-host C1 authority/effect profile, recovery semantics for uncertain outcomes, persistence guarantees, optional compatibility profiles, evidence status and limits. It is intentionally scoped. The selected stack integration is a bounded synthetic reference with one host and a synthetic destination; it is not a production deployment, compliance certification, payment system or evidence of institutional adoption.

1 Introduction

AI-agent systems are moving from conversational assistance toward delegated execution. Agents now call tools, retrieve records, propose refunds, update systems, invoke APIs and coordinate workflows across enterprise boundaries. This shift changes the governance problem. It is no longer sufficient to inspect final text, and it is no longer sufficient to record that a policy once said *allow*. The material question becomes whether a proposed action was authorized by the right institution, whether that authorization was still valid when the effect was attempted, and what is actually known about the effect afterwards.

A governed agent action typically contains several distinct events:

1. an action surface is declared before runtime;

2. an agent proposes a specific operation against that declaration;
3. institutional authority for the operation is resolved from a trusted source;
4. a runtime decision authorizes or holds the operation;
5. decision-critical inputs are revalidated immediately before dispatch;
6. an executor attempts the effect under a stable effect identity;
7. the executor’s acknowledgement arrives, fails or is lost;
8. the destination is observed independently of that acknowledgement;
9. reconciliation classifies the observation without renewing authority;
10. the retained records are reconstructed and packaged for review.

Most agent frameworks focus on orchestration. Most model runtimes focus on inference. Most logging systems record events after they occur. Cognous Control Plane occupies a narrower layer: it decides whether a declared proposal is authorized under independently resolved authority, rechecks that decision at effect time, and preserves the record of what followed.

The guiding design principle is:

No material agent effect should be attempted on the strength of an earlier decision alone. The decision must be bound to the exact operation, the authority must come from outside the request, the critical inputs must be revalidated before dispatch, and uncertainty about the outcome must be preserved rather than resolved by assumption.

The component was previously published as *Cognous Agent Control Plane*. It is now titled Cognous Control Plane and is maintained at <https://github.com/cogno-us/cognous-control-plane>. Package and tool names did not change: the Python distribution is `agent-control-plane`, the import package is `agent_control_plane` and the command-line tool is `acp`. Record schema identifiers are also unchanged.

The public surface now has two layers. The first is the *bounded authorization-to-effect workflow*, implemented by the class `BoundedAuthorizationWorkflow`; it is the interface the stack selects as “bounded authorization/effect 0.1”. The second is the *run-record toolkit* from the first public release, built around frames, action proposals, a policy gate, a run recorder and replay bundles; it remains available for experiments and simple integrations, but its *allow* result is a policy-gate result, not an institutional grant.

2 Problem Statement

Enterprise AI-agent adoption creates a runtime control gap. The system may know that an agent produced a result, but it may not know, in a structured and reconstructable way, what the agent attempted, under which authority, at which time, and with what outcome.

2.1 Proposed Versus Executed Actions

An agent may propose an action that is not executed. Without explicit records of proposals and held decisions, refused behavior disappears from the audit trail. The absence of a bad outcome is not evidence that a risky action was stopped.

2.2 Declaration Versus Permission

A tool name, a conforming manifest, a valid signature or a blockchain inclusion proof describes or attests to something; none of them is a grant. A control layer must keep the declared action surface, the evidence about it and the institutional permission to act as separate inputs.

2.3 Authority Ambiguity

Agents operate under delegated authority. A system needs an explicit binding of which actor, acting for which principal, held which grant revision, with which approvals and under which policy versions, at the time of decision.

2.4 Stale Authority at Effect Time

Grants are revoked, approvals withdrawn, policies superseded and evidence goes stale. A decision recorded earlier is not a permanent credential. Unless critical inputs are rechecked before the effect, a withdrawn approval may be discovered only after the action has happened.

2.5 Uncertain Outcomes

A timeout or lost acknowledgement does not mean an effect failed. The destination may already have committed. Treating uncertainty as failure and resubmitting can create duplicate effects; treating a single “not found” reading as proof of absence ignores in-flight requests that may still commit.

2.6 Opaque Policy Decisions and Reliance

A decision that merely says allow, block or escalate may be insufficient for review. Reviewers need to know which inputs and rules produced it and which external sources the agent relied on.

2.7 Unreconstructable Runs

When a run cannot be reconstructed, investigation becomes forensic guesswork. Reviewers need connected records of proposals, decisions, attempts, observations and reconciliation that downstream tools can import without rewriting.

2.8 Unsafe Export and Lost Records

Records may contain sensitive targets, payloads or reasons, and concurrent writers can silently lose records if storage is not transactional. A control layer should support redacted export and protect its own record history.

3 Design Goals

Cognous Control Plane is designed around twelve goals.

1. **Declaration binding.** A runtime proposal is bound to an exact Manifest declaration, target, payload, adapter and requested permissions.
2. **Independent authority.** Authority is resolved through a trusted resolver interface, never taken from the incoming request.

3. **Deterministic binding.** Identical decision inputs produce the same authorization binding and the same deterministic effect identity.
4. **Effect-time revalidation.** Decision-critical inputs are resolved again and compared before dispatch.
5. **Visible refusals.** Held decisions and blocked actions are first-class records.
6. **Distinct lifecycle facts.** Decision, attempt, acknowledgement, destination observation and reconciliation are separate records.
7. **Conservative recovery.** Unknown outcomes stay unknown; observed absence does not authorize retry.
8. **Durable local records.** Record transactions survive concurrent cooperating writers and process interruption on supported local filesystems.
9. **Explicit assurance levels.** Default, optional and unclaimed ordering guarantees are named and kept apart.
10. **Reconstructability.** Retained records can be imported by downstream replay and evidence tools.
11. **Safe export.** Legacy run records can be validated, redacted and signed for controlled review.
12. **Bounded extensibility.** Optional profiles and adapters are added without silently changing the default path.

4 Non-Goals

The architecture is deliberately bounded. Cognous Control Plane is not:

- an agent framework, planner or model runtime;
- a source of institutional authority, identity or mandate;
- an executor or destination (the stack's constrained executor is the Cognous Execution Runtime);
- a dashboard or full policy authoring system;
- a compliance product or certification;
- a production identity, credential or key-management system;
- a payment system or evidence of external settlement;
- a distributed transaction coordinator or exactly-once delivery mechanism;
- a security boundary by itself, or a substitute for application-level authorization;
- a guarantee that model outputs are true, safe or complete.

It evaluates only the authority, policy and evidence supplied to it. The reference resolver is deterministic fixture infrastructure (`SyntheticResolver`), and it reports itself as unauthenticated by design. If configured grants, policies or role mappings are incomplete or wrong, the Control Plane will apply them faithfully; it will not infer missing governance intent.

5 System Overview

5.1 Position in the Cognous Open Control Stack

The Cognous Open Control Stack comprises 13 components in six responsibility groups: 01 Knowledge & Provenance; 02 Institutional Governance; 03 Declaration & Policy Control; 04 Constrained Execution; 05 Replay & Evidence; and 06 Reasoning & Human Oversight, with Cognous Governed Exchange (GAX/IMX) as cross-cutting integration. The groups are a responsibility map, not an execution sequence. Control Plane belongs to group 03 with the Cognous Action Manifest.

The supported workflow is one synthetic refund. Local durable transport delivers a message to the GAX recipient adapter, which produces a Manifest-bound proposal. A separately configured resolver supplies Authority Context (the profile defined in Cognous Institutional Governance). The Control Plane decides and revalidates before the Cognous Execution Runtime reaches a synthetic SQLite destination. Replay Bundle reconstructs the retained producer records, and Governance Evidence Pack and ODES derive review artifacts from them.

Table 1 summarizes what Control Plane is responsible for and what its outputs do not establish.

Aspect	Statement
Owens	Runtime decision against resolved authority; effect-time revalidation; decision, attempt, observation and reconciliation records
Consumes	Manifest 1.1 declaration; Authority Context 0.1.0 and resolver-supplied grant, identity, mandate, approval, policy, conflict and evidence status
Hands to	Constrained executor (exact operation and effect identity); Replay Bundle importer (retained records)
Does not establish	A completed destination effect because authorization passed; institutional authenticity of resolver data; external settlement

Table 1: Control Plane responsibilities within the stack.

5.2 Bounded Runtime Pattern

The bounded pattern is:

Action Manifest 1.1 declaration

```
-> RuntimeProposal (exact action, target, payload, adapter)
-> Resolver: Authority Context, grant, identity, mandate,
            approvals, policy, conflicts, evidence, role mapping
-> RuntimeDecision (authorized | hold) + AuthorizationBinding
-> Effect-time revalidation (resolve again, compare binding)
-> Pre-dispatch observation and reconciliation
-> EffectAttempt (stable effect_id, new attempt_id)
-> Destination apply -> acknowledged | partial | unknown | failed
-> EffectObservation -> ReconciliationResult
-> BoundedRecordStore (locked local JSON) -> downstream import
```

5.3 Run-Record Toolkit Pattern

The retained toolkit pattern is:

Agent task -> PolicyConfig -> Frame -> ActionProposal
 -> PolicyGate -> PolicyDecision + PolicyEvaluationTrace
 -> allow | block | escalate -> optional ToolAdapter execution
 -> RelianceRecord -> RunRecord -> ValidationReport
 -> ReplayBundle -> optional redacted, signed or persisted export

In both patterns the system sits beside an agent. The agent still performs planning and generation; the Control Plane decides, records and packages the governance-relevant events around the agent's proposed actions.

6 Core Object Model

6.1 Bounded Workflow Objects

Table 2 lists the objects of the bounded authorization-to-effect workflow at the selected revision.

Object	Role
RuntimeProposal	Manifest ID, version and digest; actor and principal; action and adapter IDs; target; payload and payload commitment; requested permissions; amount, unit and effect count; Authority Context and requirement references; temporal, correlation, risk and evidence fields
AuthorizationBinding	Decision-critical bindings: proposal commitment; manifest ID, version and digest; actor and principal; action, adapter and target; payload commitment; requested permissions, amount, unit and effects; Authority Context identity, version and status; requirement ID and commitment; grant ID and revision; policy versions; role-mapping version and digest
RuntimeDecision	Decision ID, deterministic effect ID, result, reasons, decision time and binding (null unless authorized)
EffectAttempt	New attempt ID for every execution or recovery try, linked to effect and decision; status attempted, acknowledged, unknown, failed or partial
EffectObservation	Destination-reported state (absent, applied, partial, unknown) for an effect, with observation time
ObservationPolicy	Required maximum observation age and future clock tolerance
ReconciliationResult	Classification of one observation: applied, observed_absent or hold, with acceptance flag, reasons and retry_eligible
BoundedRunRecord	Append-only collections of decisions, attempts, observations and reconciliations for one run

Table 2: Bounded workflow objects.

The `RuntimeDecision` result type admits *authorized*, *hold* and *deny*; the reference `decide()` method emits *authorized* when no reason holds the proposal and *hold* otherwise. Commitments use SHA-256 over sorted, compact UTF-8 JSON with non-finite numbers rejected, prefixed `sha256:`, matching the Manifest 1.1 convention.

6.2 Run-Record Toolkit Objects

The toolkit objects are unchanged in purpose from the first release.

- A `Frame` is the immutable execution context for a run (task, actor, environment, allowed and blocked tools, policy version).
- An `ActionProposal` records a tool action before execution.
- A `PolicyDecision` records allow, block or escalate with a policy name, reason, deterministic fingerprint and optional trace identifier; a `PolicyEvaluationTrace` holds the ordered rule evaluations (`PolicyRuleEvaluation`).
- A `BlockedAction` is created for every block decision.
- An `AuthorityRecord` grants run- and actor-scoped permission scopes, optionally expiring.
- A `RelianceRecord` records dependence on a tool, database, file, API, user input, model output or other source.
- A `RunRecord` aggregates these; a `ReplayBundle` packages a completed run; a signed bundle (`SignedReplayBundle`) adds HMAC-SHA256 export-integrity metadata.
- Validation issues and reports, redaction and policy configuration, and tool execution results support validation, redaction, configuration and adapter execution.

These toolkit objects and the bounded objects are separate record families. In particular, the toolkit's `ReplayBundle` is a Control Plane export format; it is distinct from the Cognous Replay Bundle component's Reconstruction Bundle format discussed in Section 16.

7 Authorization Semantics

7.1 Supported Upstream Interfaces

The bounded workflow is pinned and tested against exact upstream conventions: Action Manifest 1.1; Authority Context 0.1.0 from Cognous Institutional Governance, whose serialized interface status remains `proposed_pending_governor_review`; BitRep (Cognous Evidence Attestation) verification contract v1; and The Index (Cognous Evidence Registry) local blockchain reference. A BitRep verification result establishes only its own verification assertions, and chain inclusion is a provenance property. Neither issues authority, and the Control Plane does not duplicate either component's verification or consensus.

7.2 Ownership

The Manifest declares the proposed action envelope. The Control Plane resolves current inputs and makes a runtime decision. Institutions own mandates, grants, approval records and status sources. An execution adapter enforces the exact checked operation and exposes delivery and reconciliation behavior. A conforming manifest, valid signature, accepted verification result or chain inclusion is therefore insufficient by itself to execute an effect.

7.3 Resolver Interface

The workflow requires independently supplied resolvers for: Authority Context and requirement binding; current grant ID, revision, status and authority basis; acting identity, principal and delegation-chain validity; issuer mandate; approval records bound to grant revision, proposal commitment and policy versions; policy versions and status and explicit authority-conflict status; required evidence status and freshness; and an explicit, institution-scoped, versioned reviewer-role mapping where Manifest labels differ from Authority Context role identifiers.

Resolver lookup arguments are not treated as proof that a returned record belongs to the requested object. Returned grant, identity, mandate, approval, policy, conflict and evidence records are checked against the requested identifiers, source references, institution and authority domain. Grant, policy, conflict, identity, mandate and approval observations have explicit maximum ages; required evidence uses its Authority Context `max_age_seconds`; observations beyond the configured future clock tolerance are rejected.

7.4 Scope, Validity and Consequence

The reference uses exact finite target and permission matching; no wildcard, hierarchical inference, fuzzy matching or unit conversion is implemented. A proposal must fit the Manifest declaration, the Authority Context requirement envelope and the issued grant; a grant broader than the requirement does not widen it, and the execution envelope uses the narrowest represented `max_effects`. Grant validity is exclusive at `expires_at`. Suspended, revoked, unknown, stale or revision-mismatched status holds the effect. The Manifest consequence tier must equal the requirement tier. Unknown or ambiguous role mappings and unresolved, unknown or stale authority conflicts hold the effect. Approval records must bind the exact proposal commitment and policy versions and satisfy actor-independence when requested. Delegated authority is accepted only when the identity resolver returns an authenticated valid chain; the synthetic resolver performs no cryptography.

7.5 Legacy Policy Gate

The toolkit `PolicyGate` retains its deliberately small, deterministic rule set (Table 3).

No.	Condition	Result	Rationale
1	tool is explicitly blocked	block	explicit block list takes priority
2	tool is not in allowed tools	escalate	unknown tools require review
3	external send without matching authority	block	outbound actions require explicit authority
4	read with an allowed tool	allow	allowed reads are permitted
5	write with write authority	allow	write requires scoped authority
6	no rule matches	escalate	unknown cases are not silently allowed

Table 3: Run-record toolkit policy-gate rules.

The deterministic fingerprint is a SHA-256 digest of canonical inputs: tool name, action type, target, sorted allowed and blocked tools, policy version and sorted active authority scopes, and, since the bounded-authorization work, payload content. Two changes protect the toolkit from being mistaken for the bounded contract: the fingerprint now includes the payload, and `execute_with_control()` rejects an adapter whose name or action type does not match the approved proposal. Existing recorded *allow* results remain historical policy-gate results; they are not retroactively converted into institutional grants.

8 Policy Evaluation Traces

`evaluate_with_trace()` returns both a `PolicyDecision` and a `PolicyEvaluationTrace` recording the ordered rule path: blocked tool, unknown tool, external-send authority, read allowed tool, write authority and default escalation. A trace shows which rule matched, which earlier rules did not, the result and why. `RunRecorder` stores traces by default in both `RunRecord` and toolkit `ReplayBundle` exports.

In the bounded workflow, the equivalent explanatory surface is the decision’s reason list together with the binding it records. A held decision carries reason codes and no binding; an authorized decision carries the full binding that effect-time revalidation must reproduce.

9 Lightweight Policy Configuration

`PolicyConfig` remains a small JSON-backed portability helper, not a policy language. It contains a policy version, allowed and blocked tools, an `authority_required` mapping from action type to required scopes, and a default action. Two helpers bridge configuration to runtime records:

- `frame_from_policy_config()` creates a `Frame` while requiring the caller to supply task, actor and environment;
- `authority_records_from_policy_config()` creates `AuthorityRecord` objects for a supplied run, actor, source and optional expiry.

The example `examples/policy_config_demo.py` exercises these helpers.

This configuration feeds only the toolkit gate. Institutional requirements for the bounded workflow come from the Authority Context and resolver, not from `PolicyConfig`.

10 Authority, Evidence and Reliance

Three concepts are kept distinct.

Authority answers whether this actor, for this principal, under this grant revision, approvals and policy, may perform this exact operation now. In the bounded workflow it is resolved from institutional sources at decision time and again at effect time. In the toolkit, `AuthorityRecord` provides a simpler run-scoped stand-in.

Evidence answers whether required supporting facts are present and fresh. Evidence status is an input to authorization, not a substitute for it. A valid signature, chain inclusion, message receipt, reasoning instruction or evidence-package digest never authorizes execution.

Reliance answers what external source the agent depended on. A reliance record documents dependence; it does not grant permission. This distinction prevents the common failure of treating source access as authorization.

11 Recording Workflows

11.1 Bounded Workflow

A typical bounded workflow is:

1. construct `BoundedAuthorizationWorkflow` with the Manifest, resolver, destination adapter, record store and an `ObservationPolicy`;
2. call `decide()` with a `RuntimeProposal`; the decision is appended to the record store;
3. call `execute()` with the proposal, the persisted decision and the adapter ID;
4. if the outcome is uncertain, call `reconcile()` for the original effect;

5. hand the retained `BoundedRunRecord` to downstream importers.

`execute()` refuses unless the supplied decision exactly equals the persisted decision, the decision is authorized with a binding, and its effect ID equals the deterministic effect ID derived from that binding.

11.2 Run-Record Toolkit

The toolkit `RunRecorder` workflow is unchanged: start a run, add authority records, propose an action, evaluate it and record a trace, create a blocked-action record if blocked, execute only if allowed, record reliance, complete the run, export the run record and generate a replay bundle. The recorder rejects actions from another run and actions not proposed through it.

12 Effect-Time Revalidation: the C0 and C1 Boundary

The stack names its ordering guarantees explicitly. Table 4 states the current claim for each level.

Level	Current claim
C0 (selected default)	Independently resolve authority and revalidate decision-critical inputs immediately before dispatch. Pre-effect revalidation, <i>not</i> destination-commit atomicity; a check-to-commit race remains.
C1 (optional)	Same-host cooperating authority/effect ordering through one participating SQLite transaction boundary. Not enabled by default; not an external destination guarantee.
C2/C3	No claim. No remote commit protocol, distributed coordinator, external settlement finality or production exactly-once guarantee.

Table 4: Ordering levels as stated by the stack release status.

12.1 C0: Revalidate, Then Dispatch

On the default path, `execute()` loads and verifies the immutable persisted decision, resolves current authority, policy, approval and evidence again, requires the resolved binding to equal the persisted binding, validates adapter identity, reconciles current destination state, records an attempt and calls the destination adapter. A changed payload, target, actor, adapter, manifest, grant revision or status, approval, policy, role mapping or required evidence prevents the effect. After revalidation the workflow does not reread Authority Context or grant data: the destination receives only the frozen proposal fields and values from the validated binding.

There is no transaction shared with the destination commit. A later check-to-commit inventory qualification on the Control Plane repository makes this explicit in both directions. Revocation, policy supersession, approval withdrawal, or unknown or stale evidence injected *before* revalidation fails closed with no destination effect. The same changes injected *after* revalidation but before destination commit can still be followed by an applied synthetic effect; the test asserts this residual race as a limitation rather than treating it as a failure. A mutation after a committed effect does not rewrite that historical effect.

The reference `LocalRefundDestination` closes only its own deduplication and cumulative-count race, using a process-local lock and atomic file replacement. It enforces `max_effects` cumulatively per grant for threads sharing one destination object; this is not a distributed budget and does not protect multiple processes or remote destinations.

12.2 C1: Optional Same-Host Authority/Effect Claim

The opt-in profile `urn:cognous:profiles:local-authority-effect:0.1.0-proposed` is the Control Plane side of a same-host boundary that orders authorization-critical invalidation against one protected SQLite effect. Two APIs are distinct. `materialize_local_execution_claim()` returns a validated but non-executable snapshot. `provision_local_execution_claim()` defines the trusted handoff: while the resolver's `authority_effect_handoff` excludes authority-invalidating writers, the Control Plane performs a final resolve, captures one coherent snapshot, validates active and current grant, approval, policy and evidence projections, constructs the claim and provisions it into the participating executor store. Only then does that local store become authoritative for later profile mutations.

The claim binds institution and domain, actor and principal, grant ID and revision, decision and effect IDs, the exact operation commitment, approval, policy, evidence and requirement commitments, budget and maximum effects, a validity interval, the profile version and a complete claim commitment. Possession of a claim does not authorize execution. The Cognous Execution Runtime's matching profile validates and consumes the claim, checks current authoritative rows and budget, and inserts the protected effect inside one `BEGIN IMMEDIATE` SQLite transaction. If an invalidating writer commits first, execution holds; if the effect commits first, a later revocation is historical ordering after the effect.

The selected Control Plane revision contains this profile, but the default bounded workflow does not enable it. A separate qualification run passed 73 tests against its reviewed source revisions; that run is profile evidence and is not added to the default release count. C1 requires cooperating writers on one host. It does not establish cross-host or distributed transactions, remote revocation, external destination atomicity, production identity or production readiness.

12.3 Effect Identity Versus Business Intent

The deterministic effect ID prevents the same effect from being applied twice at the reference destination. It does not deduplicate equivalent business intent across distinct, separately authorized operation identities: two valid proposals for the same underlying refund can both execute on the default path. A refund-intent profile in the selected Execution Runtime source binds one intent to its original effect and dispatch claim, but it is a separate optional profile, not enabled by default and not composed with C1; the two profiles cannot share a database.

13 Attempts, Observations and Reconciliation

Authorized, attempted, acknowledged, observed and verified are not synonyms. The reference never emits a *verified* state because no independent verifier is implemented.

13.1 Observation Validation

Destination observations are evidence, not authorization. A workflow that relies on them must be constructed with an explicit `ObservationPolicy`, and reconciliation requires a timezone-aware evaluation time; missing policy or time fails closed. Before an observation can establish applied, partial or absent state, the Control Plane checks that its effect ID exactly matches, that `observed_at` is present, timezone-aware, within the configured age and not beyond the future clock tolerance, that positive destination evidence names the same effect consistently, and that an absence carries no contradictory state. Rejected observations remain attached to the `ReconciliationResult` with reasons and `observation_accepted=false`; they are not added to the accepted observations.

13.2 Reconciliation Results

A fresh matching applied observation yields *applied*. A fresh matching absence yields *observed_absent* with `retry_eligible=false`; it proves only that the destination did not report the effect at that observation boundary, not that an earlier dispatch cannot commit later. Anything insufficient, invalid, unknown, unavailable, partial or contradictory yields *hold*. Reconciliation never executes an effect, renews a decision or creates a grant or approval. The historical `safe_to_retry` value remains readable in old records but is no longer emitted and is never upgraded into retry permission.

13.3 Execution Compatibility

Before first dispatch the destination observation must validate under the policy, and a fresh validated absence permits only that initial dispatch. If any prior attempt exists for the effect, observed absence does not permit resubmission; a future, destination-specific retry-eligibility mechanism with idempotency or confirmed termination semantics would be required, and none is implemented. A lost acknowledgement is reconciled against destination state: an already applied effect is recorded as acknowledged without re-application, and a partial effect is held. After dispatch, `execute()` returns the attempt and either a validated observation or `None`; `None` means no validated observation was established, not that the effect was rolled back or is safe to repeat.

13.4 Stack-Level Recovery Semantics

The stack's recovery semantics build on these records. Unknown acknowledgement does not imply absence. Original-effect reconciliation can resolve delivery from accepted applied evidence without a replacement dispatch, while preserving the interrupted acknowledgement history. Changed authority can deny a current recovery request before destination observation; historical execution evidence remains historical, and denial neither erases it nor renews permission. Late-commit cases in the hub qualification show an original request committing after two absence observations. No rollback, cancellation safety, termination confirmation, remote finality or exactly-once delivery is claimed.

14 Persistent Record Transactions

`BoundedRecordStore` protects each initialization, load and append with an exclusive Linux `flock` on a stable `<record-name>.lock` sidecar. The lock covers reload, validation, append, temporary-file flush and `fsync`, atomic replacement and parent-directory `fsync`. Each transaction opens and closes its own descriptor; process death releases the lock. Instances never cache a document for a later mutation, and raw historical fields, including extensions not exposed by current readers, are preserved.

The supported boundary is narrow. All participants must use the repaired implementation under the same service user, with one canonical path in a trusted writable directory. Linux is required; `ext2/3/4`, `XFS`, `Btrfs`, `tmpfs` and local overlay mounts are accepted subject to working locking, rename and `fsync` semantics, while `NFS`, `SMB/CIFS`, `FUSE` and unknown types are rejected, with no unlocked fallback. Corrupt records raise errors and are never replaced by an empty store. Termination before replacement leaves the previous document authoritative; termination after replacement leaves a complete new document visible even if the caller never saw success. An append is not idempotent: repeating a call after an uncertain result may append again.

Qualification terminated processes on both sides of real replacement and required a separate waiting process to recover within a bounded deadline. This is process-interruption evidence, not hardware or power-

loss certification. Individual record transactions are protected; the whole decide, execute and reconcile workflow is not atomic, and no cross-host persistence, distributed budget or exactly-once claim follows.

The toolkit retains its separate minimal persistence interface and filesystem adapter, which store run records and replay bundles as JSON files for tests, examples and small integrations.

15 Framework Integration, Tool Adapters and Optional Compatibility

15.1 Mock Framework Integration

The mock framework demo remains a dependency-light pattern, not a vendor adapter. A mock agent proposes actions; each is recorded through `RunRecorder`, evaluated by the gate, executed only if allowed and followed by reliance records, after which the run is exported. Its purpose is architectural clarity, not framework coverage.

15.2 Toolkit Tool Adapter Execution

The `ToolAdapter` protocol and `execute_with_control()` helper still execute an adapter only after an *allow* decision, capture adapter exceptions in `ToolExecutionResult.error`, and record reliance after successful execution. They now also reject adapter name or action-type substitution before execution. New integrations that require bounded authority should use `BoundedAuthorizationWorkflow`.

15.3 Optional Microsoft Agent Governance Toolkit Profile

The repository includes an optional compatibility adapter for the Microsoft Agent Governance Toolkit, pinned to upstream revision 84ae6734, Python SDK `agent-control-specification` 0.4.0b0, at the `pre_tool_call` intervention point. The toolkit is conjunctive only: its *allow* is not a grant and cannot override a Cognous hold; its *deny* blocks the compatibility path; and warning, escalation, unavailable service, malformed response or evaluation error do not authorize. A transform may replace only the policy target used as the proposal payload, after which the payload commitment is recalculated and Cognous authority is evaluated afresh; authorization for the original proposal is never reused. The adapter's qualification uses the real pinned SDK types with an in-process runtime client, not a live remote policy service. It was merged after the revision selected by the stack and is not part of the core component lock.

16 Replay Bundles and Downstream Reconstruction

16.1 Toolkit Replay Bundles

The toolkit `ReplayBundle` packages a completed run record: frame, proposals, decisions, traces, authority and reliance records, blocked actions and final output. It supports post-run audit, debugging, policy regression testing, redaction and signing. It does not reconstruct the external world: tool outputs, database states and files may change after the run.

16.2 Reconstruction by the Replay Bundle Component

In the stack, the bounded records are reconstructed by the separate Cognous Replay Bundle component. Its Reconstruction Bundle 0.2.0 importer validates exact supported Control Plane and executor revision combinations and rejects unknown revisions rather than inferring compatibility from shape. It preserves proposal, decision and effect identity while keeping Control Plane and executor attempt namespaces separate, and

records whether the destination was observed applied, absent, partial or unresolved. Governance Evidence Pack and ODES then derive review artifacts from the reconstruction. Reconstruction is not policy reevaluation, renewed authority or independent verification of the destination.

17 JSON Schemas and Conformance

The repository publishes 14 JSON Schema files for the toolkit record formats: frame, action proposal, policy decision, policy rule evaluation, policy evaluation trace, authority record, reliance record, redaction configuration, tool execution result, validation issue, validation report, run record, replay bundle and signed replay bundle. Their identifiers are unchanged by the component rename. They support interchange, external structural validation and documentation of the public object boundary.

The bounded workflow objects are defined as Pydantic models in the package and exercised by its tests and by downstream importers pinned to exact revisions; they are not published as separate schema files. The repository also carries proposed-profile test vectors for the Decision Input Commitment record (Section 22). Schemas validate structure and allowed values; they do not prove that an agent's output is correct, that a policy is sufficient, that an action was safe or that a compliance obligation has been met.

18 Semantic Validation

Semantic validation reports for toolkit run records and replay bundles go beyond model parsing. They check that embedded actions, decisions, authority records and traces carry the correct run identifier; that decisions, blocked actions, traces and reliance references point to known actions; that blocked actions correspond to block decisions; that decision trace identifiers reference known traces and linked fingerprints match; and they warn on authority-actor mismatch and on completed runs without final output. Error-severity issues make a report invalid; warnings flag review concerns. This is implementation-level consistency checking, not proof of factual correctness, compliance or safety.

19 Redacted Exports

`RedactionConfig` supports redaction of action payloads, action targets, final output and recorded reasons (action, policy-decision, blocked-action and trace-rule reasons). Redaction returns copies and does not mutate originals. Identifiers, timestamps, decision results, policy names, fingerprints and trace relationships are preserved. Fingerprints are not recomputed: they correspond to the original decision inputs, not the redacted export. These helpers apply to toolkit records only; this package does not redact bounded-workflow records, and redaction for downstream review of reconstructed records belongs to the Replay Bundle component's export tooling.

20 Signed Replay Bundles

Signed toolkit replay bundles carry HMAC-SHA256 over a deterministic JSON serialization, computed with a caller-supplied secret and verified by recomputation. This detects modification after export. It is not production key management or a security boundary, and a valid signature, like any other evidence, does not authorize execution.

21 Command-Line Interface

The `acp` command-line interface provides utilities for toolkit exports. The repository’s current CLI exposes exactly six commands:

- `acp validate-run path/to/run_record.json`
- `acp validate-replay path/to/replay_bundle.json`
- `acp redact-run input.json -out output.json`
- `acp redact-replay input.json -out output.json`
- `acp sign-replay bundle.json -secret SECRET -out signed.json`
- `acp verify-signed-replay signed.json -secret SECRET`

The repository’s getting-started sequence, run from a fresh checkout with Python 3.11 or later in an activated virtual environment, is:

```
python -m pip install -e ".[dev]"
acp --help
acp validate-run examples/sample_run_record.json
python examples/policy_config_demo.py
```

These commands exercise local reference tooling only. The bounded workflow has no CLI; it is exercised through the Python API, the repository tests and the stack hub’s reference runner, which checks out the exact locked components and runs two isolated repetitions:

```
python tools/reference_release.py run --results-dir results/reference
```

22 Proposed Profiles and Post-Selection Source

The stack’s component lock selects Control Plane revision `d3dadee` with the interface “bounded authorization/effect 0.1”. Newer accepted repository work is not automatically selected. Readers should therefore distinguish three categories.

In the selected source but not enabled by default. The C1 claim profile (Section 12). The Decision Input Commitment record, profile `0.1.0-proposed`, is a standalone, non-authorizing sidecar with its own verifier (`tools/verify_decision_input_commitment.py`) and vectors. It would commit policy content and evaluation semantics, complete evidence obligations and provenance, context, adjudication time and resolver semantics under one closed commitment, using exactly one supported evidence classifier (`urn:cognous:resolver:required-evidence-v1`). It is not used by the live authorization path, does not issue or consume authority and does not prove that the live Control Plane used it.

Merged on the repository after the selected revision. These are accepted source changes that the current stack release does not include:

- tenant-bound authorization, contract generation `bounded-authorization-effect/0.2`, in which a Manifest proposal `tenant_id` is required, committed and bound, with no inference or default;
- the check-to-commit inventory qualification of C0 and C1 described in Section 12;

- durable effect-time refusal evidence: a minimized `EffectTimeRefusal` row with exact reason codes and changed-input versions, retained before the refusal is raised, without copying raw payload;
- negative resolution evidence: `resolution_inputs` on decisions, including holds, and refusals, recording each actual resolver read with a record commitment and an explicit `missing` flag;
- an addendum of 13 October 2026 that treats every destination error as *unknown* unless a trusted adapter raises `DefinitiveDestinationRejection` (the selected revision classifies only time-outs as unknown and other exceptions as failed), appends `AttemptCloseout` records when a later observation resolves an uncertain attempt, makes the Control Plane process clock the default evaluation time with a labeled test-only override, keeps durable per-store timestamp floors, and retains the original and resumed proposal horizons;
- the optional Microsoft Agent Governance Toolkit adapter.

None of these items changes the C0 race, the optional status of C1, the absence of C2/C3 claims or the operational-trust hold. Each would require separate integration qualification before the stack could select it.

23 Reference Implementation Status

Table 5 gives the status of each capability relative to the selected stack revision.

Capability	Status	Notes
Bounded authorization workflow	Selected default	bounded authorization/effect 0.1
Resolver interface and binding checks	Selected default	synthetic resolver is unauthenticated
Effect-time revalidation (C0)	Selected default	check-to-commit race remains
Observation validation and reconciliation	Selected default	<code>observed_absent</code> never authorizes retry
Locked record store	Selected default	local Linux, same user, one host
Local synthetic refund destination	Selected source	process-local cumulative cap
Local authority/effect claim (C1)	Optional, not enabled	same-host SQLite; 73-test separate run
Decision Input Commitment record	Proposal	non-authorizing; standalone verifier
Tenant-bound 0.2 contract	Repository main	not selected
Durable refusal and resolution-input evidence	Repository main	not selected
Unknown-outcome and clock addendum	Repository main	not selected
Microsoft AGT adapter	Optional, repository main	conjunctive only; not in core lock
Policy gate, traces, run records	Retained toolkit	allow is not an institutional grant
Toolkit replay, validation, redaction, signing	Retained toolkit	HMAC export integrity only
acp CLI	Retained toolkit	six commands
Toolkit filesystem persistence	Retained toolkit	local JSON storage

Capability	Status	Notes
Production resolver, credentials, key custody	Not implemented	deployment responsibility
Distributed budgets, remote finality, C2/C3	Not claimed	

Table 5: Capability status at the selected revision.

24 Evidence, Tests and Continuous Integration

Tests are part of the public trust surface: they show which claims are exercised by executable checks. The repository’s GitHub Actions workflow runs the full suite on Python 3.11 and 3.12, with separate workflows for the check-to-commit inventory, durable refusal evidence and the pinned Microsoft toolkit profile. Its tests cover the toolkit behavior from the first release and the bounded workflow’s adversarial cases: substitution, revocation, staleness, delegation, evidence, lost acknowledgements, restart, duplicates, partial delivery and concurrent attempts, together with record-store concurrency and interruption, the C1 claim profile and the proposed commitment verifier. The repository does not publish a single authoritative per-component test total, and none is asserted here.

Stack-level evidence is aggregate and specific to one evidence generation. For the selected lock profile, `merged-producers-v1`, the release evidence, full-candidate CI run 37694032916, resolved all 35 acceptance scenarios with required suites executed twice. A fresh reproduction on 13 October 2026 ran 1,009 Python testcases in each of two repetitions plus 20 Node tests; the matrix resolved 34 passed invariants and one characterized limitation, the business-intent deduplication case, which is reported as not established rather than prevented. The optional C1 run’s 73 tests are separate. An earlier figure of 915 tests per repetition belongs to a historical persistence generation and is not a current count. None of these figures is a measure of production readiness, deployment benefit or compliance.

Release closeout froze feature intake on 13 October 2026. Public-preview approval is pending; final publication and operational trust (tracked as #30) remain on hold.

25 Security Considerations

Control Plane narrows specific risks and leaves others explicitly open.

Addressed within the reference. Unrecorded proposals and held decisions (durable effect-time refusal records are post-selection work); authority taken from the request rather than a trusted source; reuse of a stale decision at effect time; substitution of payload, target, adapter or decision; resolver records returned for the wrong object; stale or future-dated observations; retry on uncertain outcomes; and lost record writes among cooperating local processes.

Not addressed. The C0 check-to-commit race; cross-host ordering; production resolver authentication, identity, key custody and revocation propagation; clock authority beyond the configured trust boundary; compromised hosts, dependencies or processes; deletion or rollback of the record store by its owner; model hallucination; bad policy definitions, which the gate and workflow will apply faithfully; semantic understanding of payloads beyond the declared fields; business-intent duplicates on the default path; and compliance automation.

The correct posture is layered. The Control Plane supplies decisions, revalidation, lifecycle records and export utilities; deployments must add application authorization, infrastructure and credential security, destination-side idempotency and commit semantics, model-risk controls, policy review and operational monitoring. A production executor must declare its idempotency, observation, partial-delivery, commit or reservation, credential, revocation-age and retry semantics before stronger claims can be made.

26 Implementation Boundary

The public reference implementation exposes formats, schemas, validators, reference behavior, examples, tests and business-facing documentation. The stack’s open-source boundary places hosted services, enterprise storage, identity integration, approval workflows, compliance mappings, replay viewers, dashboards, managed policy workflows and customer-specific deployment support in a separate commercial or product layer. The principle is to open the record formats and compete on implementation, governance and operations.

Public reference layer	Product or deployment layer
Bounded decision and binding records	Authenticated institutional resolvers
Effect-time revalidation semantics	Production identity and credential custody
Attempt, observation, reconciliation records	Destination-specific adapters and finality
Local locked record store	Enterprise storage, retention, encryption
Optional C1 claim contract	Deployment-wide ordering and isolation
Toolkit records, traces, schemas	Replay viewers and dashboards
Redacted and HMAC-signed exports	Production key management
acp CLI and examples	Managed policy workflows and support

27 Example Use Cases

27.1 Refund with Withdrawn Approval

A customer-service agent proposes a synthetic refund. The decision is authorized and bound to the grant revision and approval. Before dispatch, the approval is withdrawn. Effect-time revalidation finds the changed approval and refuses; the destination records no effect, and the earlier decision is not reusable.

27.2 Refund with Lost Acknowledgement

A later authorized refund is dispatched as effect E1 and the acknowledgement is lost. The attempt is recorded as unknown. Reconciliation of E1 either accepts a fresh applied observation, resolving delivery without a second dispatch, or records point-in-time absence with `retry_eligible=false`.

27.3 Duplicate Business Intent

Two separately authorized proposals for the same underlying refund carry different effect identities. On the default path both can execute; this is the stack’s characterized limitation. The optional refund-intent execution profile addresses the local case and is not enabled by default.

27.4 Customer-Service Run Record

Using the toolkit, an agent reads a CRM record and proposes an email. The read is allowed; the send is blocked because outbound authority is missing. The trace distinguishes a missing-authority block from a blocked tool, and the run is validated, redacted, signed and exported.

27.5 Downstream Review

The bounded records from a refund run are imported by Replay Bundle, which preserves identities and observation states, and are packaged by Governance Evidence Pack for review without re-running the refund.

28 Evaluation Criteria

The usefulness of Control Plane should be evaluated against operational criteria rather than benchmark scores. A deployment should ask:

1. Is every proposal bound to an exact declared action, target and payload?
2. Is authority resolved from a trusted source independent of the request?
3. Are resolver responses checked against the requested object and freshness limits?
4. Are decision-critical inputs revalidated immediately before dispatch?
5. Is the residual check-to-commit race understood for the intended destination?
6. Are held decisions and refusals visible with reasons?
7. Are attempts, acknowledgements and observations kept as separate facts?
8. Do unknown outcomes stay unknown, and is retry kept separate from observed absence?
9. Are record writes durable for the actual filesystem and process model?
10. Is business-intent duplication addressed where it matters?
11. Can retained records be imported by downstream reconstruction without rewriting?
12. Which checks were actually executed in the intended environment, at which exact revisions?

29 Roadmap

The public roadmap should remain implementation-focused and avoid overclaiming. Release closeout has frozen feature intake for the current bounded preview, and newer component revisions are not automatically selected. Work that remains open in the repository documents includes:

- qualifying post-selection source (tenant-bound 0.2, durable refusal and resolution-input evidence, the unknown-outcome and clock addendum) for any future hub selection;
- defining durable storage, lifecycle and runtime adoption for the Decision Input Commitment record, with Replay and Evidence Pack consumer support and cross-language canonicalization evaluation;
- a destination-specific retry-eligibility mechanism with idempotency, finality or confirmed termination semantics;
- production resolver authentication, clock authority and revocation propagation, which remain deployment responsibilities.

Remote commit protocols, distributed coordination, external settlement finality, fleet-wide stop and general rollback are outside this bounded preview. Dashboards, hosted policy services and enterprise integrations remain product-layer concerns, separate from the public reference.

30 Conclusion

AI-agent governance cannot be limited to final-output review, and it cannot rely on an approval recorded at some earlier moment. As agents gain access to tools and systems of record, governance must move to the moment of effect and remain honest about what is known afterwards. Cognous Control Plane provides a public reference for that shift. It binds a proposal to an exact Manifest declaration, resolves authority from a trusted source rather than the request, records a deterministic decision binding, revalidates the critical inputs before dispatch, keeps attempts, observations and reconciliation separate, refuses to turn uncertainty into retry permission, and persists its records transactionally on one host. It retains the original run-record toolkit for simple, inspectable experiments.

The contribution is deliberately narrow and its limits are explicit: the default C0 path has a residual check-to-commit race; the stronger C1 ordering is optional and same-host; no C2/C3, external settlement or exactly-once claim is made; and business intent is not deduplicated by default. Within those limits, it makes agent actions decidable, checkable and reconstructable at the layer where agents touch enterprise systems.

References

- Cognous. *Cognous Control Plane*. Public reference implementation, 2026. <https://github.com/cogno-us/cognous-control-plane>
- Cognous. *Cognous Open Control Stack*: component lock, release status and recovery semantics, 2026. <https://github.com/cogno-us/cognous-open-control-stack>
- Cognous. *Cognous Action Manifest*, *Cognous Execution Runtime*, *Cognous Replay Bundle* and *Cognous Governance Evidence Pack*. Public reference implementations, 2026. <https://github.com/cogno-us>
- National Institute of Standards and Technology. *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. NIST AI 100-1, 2023.
- OWASP GenAI Security Project. *State of Agentic AI Security and Governance*, version 2.01, June 2026.
- John M. Willis. *Runtime Governance Body of Knowledge for Artificial Intelligence and Other Autonomous Systems: Glossary*. Discussion draft, 19 July 2026.
- Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.