PUBLIC EDITION 1.3

Bounded Agent Workflows

A provider-independent reference design for preventing churn, recovering failures, and evaluating self-improvement

Public edition: 1.3

Date: 30 September 2026

Status: Proposed reference design; not a released implementation or benchmark result

Audience: Engineers building agent orchestrators, review workflows, and recovery systems

> Keep the required outcome fixed. Make progress observable. Bound the work outside the model. Preserve both failures and successful results.

This document describes an implementable control pattern for agent workflows. It separates durable accounting and authority from model-generated proposals, then applies the same controls to recovery and improvement experiments. It does not claim to eliminate every failure, prove arbitrary prose correct, or guarantee that a task will succeed. The cited research informs the approach; the detailed protocol below is a proposed synthesis, not an endorsed laboratory standard.

Edition 1.3 clarifies historical exceptions, clerical corrections, failure accounting, input-size preflight, and execution supervision. It incorporates lessons from local trials without claiming a validated full controller or self-improvement system. Requirements and acceptance scenarios specify intended behavior; publication of this design is not implementation evidence.

Contents

1. Start with the smallest useful controller 3

2. What the system can and cannot guarantee 4

3. Architecture and trust boundaries 5

4. The work contract and deployment profile 5

5. Obligations, evidence, and progress 8

6. Admission, dispatch, and completion 9

7. Budgets, deadlines, and extensions 11

8. Crash recovery and external side effects 13

9. Reviews, repeat detection, and invalidation 14

10. Bounded automatic recovery 16

11. Optional recursive improvement of agent software 17

12. Provider-independent interfaces 18

13. Worked example 19

14. Implementation sequence 20

15. Requirements and acceptance cases 21

16. Measurement and rollout 25

17. Limitations and responsible claims 26

18. Sources and their relevance 27

1. Start with the smallest useful controller

An agent can appear busy while repeatedly reviewing the same artifact, restating the same findings, or replacing one unsupported assumption with another. This document calls that pattern churn: repeated work without sufficient new evidence or verified movement toward the accepted outcome.

A long task is not automatically churn. Investigation can be valuable before a requirement closes. Conversely, a short task that declares success without evidence is not efficient.

Build the core in this order:

1. Assign a stable work identity and an explicit acceptance contract.
2. Store that contract, consumed allowance, actions, evidence, and findings durably.
3. Place an admission check on the actual model and tool execution paths.
4. Reserve capacity before dispatch and preserve it across crashes.
5. Verify outcomes against identified artifacts, rather than accepting a model's success statement.
6. Stop new work when acceptance is satisfied, authority is missing, or allowance is exhausted.

The minimum architecture is an existing coordinator, a transactional store, a controlled execution adapter, and an evaluator. These may live in one service. Reuse existing facilities rather than introducing a new framework or a network of agents.

For a single-host implementation, a local transactional database may be sufficient. Shared multi-host execution requires verified coordination, storage, and failure semantics; do not assume that placing a local database file on a network share supplies those properties.

Automatic recovery is an extension to the core. Recursive improvement is a further extension, appropriate only after recovery and independent evaluation work. A team can adopt the core without adopting self-modification.

Scope clarification, risk analysis, design critique, and deterministic tests consume their own declared allowance and elapsed time. They are not unlimited exceptions because they precede a build or do not call a model. A small accepted scope with observable checks matters more than the number of preparation rituals performed.

1.1 A concise operating rule

> An agent may propose work, but only the controller admits it. Every admitted action belongs to an authorized work item, consumes a declared allowance, has a deadline, and produces a recorded outcome. New sessions do not reset that accounting. Passing the accepted checks ends the work.

This rule is advisory until the real execution paths enforce it.

2. What the system can and cannot guarantee

2.1 Conditional guarantees

Property	Required enforcement	Honest claim
Bounded admissions	Every covered invocation passes through durable transactional admission; budgets cannot refill themselves.	At most the authorized number of covered actions can be admitted.
Bounded local execution	An external supervisor can enforce deadlines and terminate the owned execution, including descendants.	Covered local execution is time-limited under the tested supervisor assumptions.
Bounded metered usage	Admission reserves a valid upper bound for all chargeable work, including outstanding requests, and the provider honors applicable limits.	The specified usage ceiling is enforceable for the covered billing surface.
No unauthorized completion	Acceptance requires applicable evidence and required independent decisions.	A model's unsupported PASS does not establish completion.
State continuity	Durable records, stable identity, idempotent result ingestion, and controlled recovery.	A new session does not create a fresh allowance or erase findings.

These properties do not prove that the required outcome will be achieved. They do not establish exactly-once external side effects, unlimited reliability, or immunity to a compromised host.

2.2 Assumptions and failure boundaries

The control process, authoritative store, execution supervisor, and evaluator are trusted for their declared responsibilities. Workers and model outputs are not trusted to set policy or attest to their own execution. Ordinary candidate processes must not be able to rewrite those trusted components or invoke an uncovered execution path.

If a remote provider cannot cancel an accepted operation, a local timeout may stop waiting without stopping remote execution or billing. Report that limitation. Block further dispatch and retain the outstanding reservation until the outcome is reconciled; do not claim a hard remote stop.

If the controller or store is unavailable, admission fails closed. A separately functioning supervisor must enforce deadlines already handed to it. If that supervisor also fails, the strict execution guarantee no longer holds. State the operational assumptions rather than claiming unconditional termination in any failure scenario.

A deadline is not a reason to fabricate a completed cleanup. At the cutoff, stop covered execution as supported, record unresolved effects as unknown, and expose any containment or reconciliation gap. Reserve time for ordinary cleanup, while recognizing that arbitrary external recovery cannot be guaranteed to finish within a fixed interval.

3. Architecture and trust boundaries

Component	Owns	Must not delegate to model assertions
Authority and admission controller	Accepted contracts, extensions, allowed actions, budget reservations, state transitions.	Permission, scope reduction, or additional allowance.
Durable store and artifact collector	Action history, reservations, observations, artifact identities, findings, provenance.	Whether a reported test actually executed.
Execution adapter and supervisor	Controlled dispatch, status, deadlines, cancellation, execution handles.	Whether a proposed action is authorized.
Evaluator	Declared checks, evidence interpretation, independent findings, acceptance decisions.	Final approval by the builder being evaluated.

Models can propose actions, changes, findings, and interpretations. Validate proposals against a versioned closed schema. Reject malformed fields, unsupported action types, unknown authority references, and inconsistent identities. Tool output, retrieved documents, and peer messages remain data; instructions embedded in them cannot alter the contract.

Hashes identify content. They do not establish authorization, truth, or execution provenance. The trusted collector must bind a result to its actual action, execution handle, artifact, and evaluation method.

Record configured roles and permissions. A human operator or authenticated policy service may hold authority; a role name supplied in model-generated JSON does not confer that role. Protect the store and dispatch credentials with the host's access controls or sandbox boundaries. If workers retain unrestricted administrator access, describe this as a cooperative workflow rather than tamper-resistant enforcement.

Interactive sessions need the same boundaries. Use restricted worker permissions, suitable isolation, controller-owned execution credentials, and a mandatory broker for the paths claimed as controlled. Hooks can help connect an application to that broker, but a bypassable hook does not provide complete enforcement. An uncovered path is a coverage gap, not permission to evade an existing manual hold.

An agent may transcribe an owner instruction into a manual ledger, but the row cannot authenticate its own origin. Retain the original decision reference and label that arrangement as manually governed. For authenticated grants, protect the issuer, verification configuration, and approval credentials from workers. A file restricted to the owner's account does not separate human and agent authority if the worker has the same effective access. A signature helps only when the worker cannot invoke the signing authority or replace the verifier's trusted keys.

In a provisional manual workflow, designate the session that directly observed the owner's instruction as its recorder. Retain the verbatim instruction, original message reference, root, scope, units, deadline and distinct recording identity. A peer relay alone cannot confer approval. Faithful transcription is administrative work; it must not expand the instruction. A mutable log line or content hash is a locator, not human authentication.

4. The work contract and deployment profile

4.1 Stable work identity

Assign each accepted work item a `root_work_id`. Sessions, child tasks, reviewer changes, and artifact revisions refer back to that root. The authoritative registry controls root creation and records lineage. An agent cannot obtain fresh allowance by assigning its own new identifier to the same unresolved work.

Related work may legitimately receive a separate root, but that requires an authorized allocation and a recorded relationship. A project or campaign ceiling prevents a stream of individually bounded roots from becoming unbounded aggregate work.

4.2 Required contract fields

Field	Required meaning
Identity and revision	Root ID, parent allocation if applicable, immutable revision identity.
Objective and phase	Observable outcome; design, implementation, or verification phase.
Required obligations	Stable IDs and explicit acceptance predicates.
Optional scope	Work that may be deferred without breaking the accepted required outcome.
Authority	Verifiable references to scope, execution, and any consequential-action permission.
Allowed surface	Paths, tools, environments, network destinations, and side effects that are permitted.
Dependencies	Relevant upstream contracts and environmental assumptions.
Evidence requirements	Check method, artifact/input identity, freshness or invalidation conditions.
Review policy	Required independent roles, responsibilities, and closure authority.
Budget	Action, retry, resource, concurrency, and time ceilings in explicit units.
Protected capacity	Allowance reserved for final verification and authorized recovery.
Failure behavior	When to stop, reconcile, cancel, or request an owner decision.

Required fields may not be replaced by vague references such as an unspecified spending ceiling. Missing operational values make the work inadmissible. Not-applicable dimensions must be explicit; omitting a monetary cap is acceptable only when metered execution is disallowed or a separate verified billing control is identified.

An accepted contract may evolve through authorized revisions. Preserve its history and expenditure. A scope change is not an automatic budget extension. Moving required behavior to a later layer needs the same authority as removing it from the current delivery.

4.3 Deployment profile

Before enabling automation, declare the covered invocation paths, authority source, store transaction model, execution ownership, adapter capabilities, resource-accounting method, clock policy, evaluator owner, and promotion policy. Configure numeric ceilings and test them in that environment.

Choose limits from the workload and observed performance; this reference design does not prescribe universal counts or timeouts. A category counter such as repair rounds supplements the underlying resource limits and never replaces them.

4.4 Adopting existing work

Use a one-time checkpoint: root identity, accepted outcome, prior receipt-backed expenditure, unknown history, unresolved obligations, in-flight actions, remaining authorized allowance, and the policy's effective time.

Do not automatically start an existing item at zero or invent retrospective charges under a newly adopted rule. The authority holder explicitly records the transition allocation and treatment of in-flight work. Valid prior admissions and admissions that violated the then-active policy are different cases. Record mistakes rather than grandfathering them silently.

If remaining authority is unknown, admit no new work. Reconcile already-running actions through the authorized safe procedure. Adoption cannot be repeated to obtain another initial allocation.

Later proposed limits do not retroactively become the terms of earlier valid admissions. Record the applicable policy revision and original limits with each action. A duly authorized cancellation or revised deadline may change future execution, but a draft proposal or peer interpretation cannot do so. Do not claim compliance with a resource cap whose consumption was never observed or bounded.

4.5 One authority per root and bounded exceptions

One root must have one authoritative admission ledger at a time. A local receipt cache and a central store may coexist, but they cannot independently issue overlapping capacity. Reuse an existing transactional store and identify the authority responsible for each root; a separate distributed system is not required for a single-coordinator installation.

An escalation threshold and remaining resource allowance are different checks. If a policy permits owner exceptions, represent an exception separately from an allowance extension. Bind it to the authenticated decision, root, specific refused condition, permitted action types and quantities, validity window, and any required order. Every other admission check still applies. Consume a one-use exception atomically with its applicable admission; duplicate delivery cannot consume or grant another unit.

An exception preserves earlier findings and escalation events. It grants no extra capacity unless that same authenticated decision explicitly adds it. On completion, expiry, revocation, or a binding stop condition, preserve its disposition. An unlaunched action is unspent, not consumed, and unused capacity does not override a stop condition. Do not require the owner to approve an already-valid exception again merely because the launcher changes.

Store multiple historical exceptions while selecting at most one eligible exception for a root and check type. Eligibility requires the effective time window, remaining uses and satisfied conditions. Reject conflicting eligible entries. Expired, consumed or stopped entries remain history; their presence must not prevent a separately authorized successor. Selection must evaluate eligibility, rather than taking the first historical row.

For a legacy active-record format, an authorized recorder may archive a demonstrably expired entry while preserving its exact original contents, identity, consumption and a before/after receipt. Use the format's existing supported archival convention, exclude concurrent admission, and verify the replacement before resuming. This is not permission to hide malformed accounting, retire a still-eligible grant, reuse its identity or invent a successor. Durable history must remain available for reconciliation and migration; comments alone are not an authenticated event store.

5. Obligations, evidence, and progress

5.1 Different statements need different support

Statement type	Support required
Requirement	Accepted intent establishes the obligation; checks establish compliance.
Observation	Recorded method, inputs, environment, and actual result.
Decision	Authority and provenance.
Hypothesis	Investigation; it remains unresolved until evidence settles it.
Upstream guarantee	Applicable contract, evidence, and satisfied preconditions.
Preference	Accepted rubric or a recorded owner decision.

An owner decision does not prove a library's behavior. A source link does not automatically support every nearby sentence. Requirements should include useful positive behavior as well as forbidden behavior: a system that rejects everything is usually not meeting its objective.

Map the normative parts of an existing specification to obligation IDs before repairing it. Record coverage and omissions. An incomplete assessment is `INCOMPLETE`, not PASS. Coverage records show what was processed, not proof that every implication was understood.

5.2 Progress at checkpoints

Let `R` be required obligations in the accepted revision, `U` those lacking valid acceptance evidence, and `F` open findings. Track these separately: several findings may concern one obligation.

Verified closure reduces `U` without reopening another obligation. It is a useful progress signal, not a universal admission prerequisite for every intermediate action. A complex repair or experiment can produce useful evidence before closing an obligation.

At a bounded checkpoint, distinguish:

- Verified closure: continue only if acceptance is incomplete and the next action remains admissible.
- Useful new evidence: a reproduced failure, eliminated hypothesis, or identified dependency can justify remaining bounded investigation.
- No closure and no useful evidence: reject another equivalent attempt.
- Regression or newly discovered defect: record it and evaluate a targeted response within the remaining allocation.

Do not reward deletion of requirements or conceal new defects to preserve a decreasing metric. After an authorized baseline change, report added, removed, deferred, closed, and reopened obligations separately. A denominator change is not evidence of improved performance.

The finite budget is the termination control. A progress score cannot substitute for it.

5.3 Evidence validity

Each evidence record binds the obligation, check definition, actual execution or reviewer identity, inputs, artifact digest, relevant environment, observation, and result. Define relevant dependencies before evaluation. Invalidate evidence when those dependencies change or its declared freshness condition expires.

Do not invalidate everything for an unrelated edit. Do not preserve a passing receipt when the code, configuration, policy, or environment on which it depended changed. Shared-state dependencies must be declared where they can affect acceptance; dependency edges alone do not prove contract compatibility.

Design readiness means the behavior, checks, dependencies, and failure paths are specified sufficiently to build. Implementation verification requires completed checks against the actual implementation. A promised future test establishes neither its result nor deployment authority.

5.4 Reproducible review packets

Record the amendment's starting baseline, then freeze the final candidate and relevant evidence at review admission. These are different snapshots: the admission packet must reconcile applicable decisions and material changes since work began. An old baseline cannot justify ignoring a newer restriction or known defect.

Give the packet an immutable manifest identifying the candidate, source versions and digests, exported excerpts, evaluation policy, and relevant environment. Before dispatch, check that each cited file, line range, or stable target is actually included or resolvable in the reviewer's permitted snapshot. A tool-free reviewer needs the cited content in its packet. Report missing exports as packet incompleteness; distinguish that from a claim whose underlying source does not support it. Resolve either within existing allowance before admission, or stop.

Bind each finding and review result to this packet identity. Required reviewers of one candidate use the same candidate and applicable evidence snapshot. This makes the review reproducible without implying that hashes prove the sources true.

Record later evidence and authority changes as separate events. A revocation applies when effective even if a frozen packet contains older permission. A material correction invalidates affected evidence and may require a targeted reevaluation within remaining allowance; unrelated log growth does not invalidate the whole review. Never edit an old packet in place or silently restart its budget.

Before review admission, measure the input and check the adapter's configured input/output context envelope and execution-time profile. Prefer token accounting; a documented byte ceiling can be a conservative fallback. A previously successful size does not guarantee completion, and a larger timed-out packet does not prove size caused the timeout. Reject an oversized packet before charging a review admission. Lossless compaction within accepted scope needs a new frozen packet identity and citation check, not a new scope decision. Removing obligations or dividing the review into extra invocations requires the corresponding authority and allowance.

6. Admission, dispatch, and completion

6.1 Action unit

An action is one controller-admitted execution invocation, such as a model call, tool operation, review execution, or recovery operation. A review containing additional model or tool calls must expose and charge those sub-invocations under the same root, or use an adapter that verifiably caps the whole delegated execution. Higher-level labels cannot conceal unbounded internal work.

Passive receipt ingestion and controller bookkeeping are not productive agent actions, but must have bounded implementation behavior and rate limits. They cannot launch work through an unmetered exception path.

6.2 Admission sequence

For each proposed action:

1. Validate the schema and authenticated actor; resolve the existing root and contract revision.
2. Check active state, authority, allowed surface, and relevant input identities.
3. Verify prerequisites, repeat policy, required role, and any declared trial identity.
4. Check root and parent ceilings, concurrency, deadline, and protected capacity.
5. Atomically reserve capacity, create a unique action ID, and persist dispatch intent.
6. Dispatch only that recorded action through the covered adapter.
7. Bind observations to the actual execution and ingest its result idempotently.
8. Evaluate evidence and either admit the next action or enter a terminal decision state.

At no point may an unsigned model message grant an extension or mark its own independent review complete.

6.3 States and completion

Work state	Meaning
DRAFT	No accepted executable contract yet.
ACTIVE	Actions may be admitted under the current allocation.
QUIESCING	New productive dispatch is disabled; existing work is being stopped or reconciled.
COMPLETED	Phase acceptance is established and relevant executions are settled.
DEPENDENCY_FAILED	A known dependency prevents this attempt from continuing.
NEEDS_DECISION	Scope, authority, budget, or another owner decision is unresolved.
OUTCOME_UNCERTAIN	Further work is stopped but effects or execution status remain uncertain.
CANCELLED	Authorized cancellation is recorded and relevant execution/effect status is settled.

Use a separate phase field so COMPLETED for design does not mean an implementation is verified. COMPLETED must not coexist with an unresolved execution that could invalidate the accepted artifact. Reconcile or cancel relevant siblings before closing the root.

A cancellation request with unresolved execution or effects remains QUIESCING or OUTCOME_UNCERTAIN. The request alone does not establish CANCELLED. A remote operation may finish successfully before a cancellation takes effect; reconciliation must record the actual outcome.

COMPLETED, CANCELLED, DEPENDENCY_FAILED, OUTCOME_UNCERTAIN, and NEEDS_DECISION are distinct stopped-attempt outcomes. DRAFT, ACTIVE, and QUIESCING describe lifecycle progress. A stopped root may have a later authorized continuation or an evidence-based status reconciliation; neither resets expenditure. Distinguish a review's PASS/BLOCK verdict from the root's outcome: a review BLOCK does not by itself specify whether a permitted repair remains, a dependency failed, or an owner choice is required.

Do not turn every stopped action into a request for human attention. Use NEEDS_DECISION only when a concrete choice is required. Deduplicate notifications by root, unresolved decision or incident, and relevant state revision.

Preserve repeated observations in the ledger, but notify again only for a meaningful change, an accepted reminder policy, or resolution.

Reopening requires a documented invalidation or authorized request. It preserves spent allowance and does not automatically supply more. Exact result delivery retries are idempotent; conflicting duplicate results are retained as a reconciliation issue, not silently overwritten.

6.4 Refusal routing without unnecessary owner work

Record refusal or stop reason separately from the attempt state and reviewer verdict. Use these distinctions to select the next responsible actor:

Reason	Required response
AUTHORITY_REQUIRED	Ask the authority holder for the concrete missing decision.
RECORD_ERROR	The authorized recorder may correct a uniquely evidenced transcription or format error while preserving identity, consumption and limits. Unknown accounting must be reconciled, not erased.
DEPENDENCY_FAILED	Identify the unavailable prerequisite and responsible operator. Repair only within applicable authority and allowance.
LIMIT_REACHED	End the attempt. Ask for an extension only when presenting a useful concrete choice.
EXECUTION_FAILED or NO_VERDICT	Preserve actual execution evidence; no acceptance or reviewer BLOCK is implied. Unsettled execution/effects remain OUTCOME_UNCERTAIN.

A clerical correction that faithfully records existing authority should not require the owner to grant it again. Increasing time or units, overriding escalation, changing required scope or accepting ambiguous consumption is a substantive decision. Changing enforcement code follows its release process. Repeated notifications about the same unresolved issue are merged.

7. Budgets, deadlines, and extensions

7.1 Budget dimensions

Track action slots, equivalent retries, applicable metered resources, concurrency, and elapsed-time limits. Child work is subject to its allocation and all applicable ancestor ceilings. Do not double-count actual expenditure when reporting totals; enforce each relevant ceiling against the same attributed execution.

An invocation slot is consumed when the action is durably admitted. A crash or missing report does not reset it. Reconcile actual metered consumption separately from reserved upper bounds. Releasing demonstrably unused resource reservation is not a refund of the consumed attempt slot.

Distinguish orchestration attempts from admitted reviews. A coordinator that fails before review admission consumes elapsed time and its applicable executor allocation, but no review admission unit. An admitted reviewer that returns nothing still consumes its unit. Record infrastructure outcome and review verdict separately; neither an empty output nor a timeout manufactures a reviewer BLOCK. Any replacement must fit existing retry authority or a new explicit allocation.

For a metered resource, the admission rule is:

```
settled usage + outstanding reservations + proposed upper bound <= ceiling
```

Include input, output, tool, hosting, and other chargeable components within the claimed scope. If any component lacks an enforceable upper bound, do not claim a hard aggregate spending limit. Disallow that operation under a hard-cap profile or explicitly classify the uncovered exposure before activation.

Use integer resource units or exact decimal monetary accounting. Reconcile using the provider's actual charging model, including delayed usage reports and asynchronous work. At the ceiling, stop new admissions; do not assume cancelling a request eliminates already-incurred charges.

7.2 Protected verification and recovery capacity

Reserve capacity for required final checks and permitted recovery before productive work consumes the rest. Represent the protected allocation separately from per-action reservations. When an eligible verification action consumes its protected capacity, transfer that amount into the action reservation once; do not count the same reservation twice.

Ordinary work cannot borrow protected capacity implicitly. An impossible or inadequate final-check allocation is a planning defect: revise it through the appropriate authority rather than skipping acceptance. Recovery reservation permits only the declared recovery, not more feature work.

7.3 Retry ceilings

A transient retry must satisfy a per-operation ceiling, a root-wide retry ceiling, all parent ceilings, and the overall resource limits. Keep retry lineage tied to the original operation even if a new execution ID is required. Changing the error wording or provider cannot create fresh retries.

Retries require a declared reason and a supported replay-safety argument. A failure is not automatically transient, and an unknown external outcome is not automatically retryable. Planned replication trials are different: their count, purpose, role, and aggregation rule are fixed before examining results.

7.4 Time accounting

Start the work deadline at the contract's declared admission or activation event, before setup. Record both the durable deadline and the local supervisor's remaining duration. Bound each action by the remaining work window, with time reserved for shutdown and reconciliation where possible.

Use a monotonic timer during an execution. On restart, derive remaining allowance conservatively from the persisted deadline and trusted time; detected clock rollback or unexplained clock discontinuity must not grant more time. A pause does not stop the deadline unless the accepted contract explicitly defines that behavior.

A supervisor must run outside the worker and check actual process termination or remote cancellation status. Request cancellation early enough for the declared grace period. If termination cannot be confirmed, enter an unknown/containment state and retain accounting; do not issue a success receipt for an unverified hard stop.

Targeting a launched process identifier and requesting tree termination is not itself proof of cleanup. Check cancellation results and reconcile the actual owned execution. Root-process exit can leave children alive; polling can miss short-lived parents and orphaned children. Reject uncertain identity relationships rather than risking termination of unrelated work. A wrapper that prints a success message regardless of the termination command's result fails this requirement. Verify normal exit with surviving children, failed termination, and ownership uncertainty in the declared environment.

Every action receives the minimum of its own limit and the remaining root deadline. A fixed long wrapper timeout cannot enforce a shorter subsequent step. Detached executors also need identity-bound heartbeat telemetry observed by an external supervisor; silence triggers bounded reconciliation and blocks new child dispatch, not an automatic replacement or fresh allowance. The deadline must remain enforceable if the worker or monitoring conversation disappears. A heartbeat is neither proof of useful progress nor proof that work stopped.

7.5 Extensions

An extension needs an authenticated authority decision identifying root, increment and units, purpose, and deadline treatment. Apply each decision ID once, transactionally. Preserve previous expenditure, findings, and scope. Ordinary clear language may establish the decision through the deployment's authority interface; no magic phrase is required.

Extensions do not erase overruns. Record overruns separately and prevent new work until the controller has valid capacity and authority. Automatic renewal requires a separately accepted program policy with aggregate limits; it is not an implicit feature of a recurring task.

Make allocation units explicit. An additive extension increases the authorized total without forgetting spent capacity. An explicitly authorized replacement of a remaining window has different semantics and must state what happens to prior unused capacity. Do not silently substitute latest-row timestamp accounting for an additive root ledger or accept repeated copies of one decision as fresh extensions.

7.6 Building and changing the controller itself

The controller's own build and audit work need a separately recorded bootstrap allocation. The candidate cannot authorize or extend that allocation, select a favorable replacement evaluator, or write its own approval. Until the intended controls exist, describe the bootstrap as manually governed and use an independently controlled, tested execution supervisor where available.

Freeze the candidate identity for required reviews. Keep ordinary dispatch paused during a multi-file activation until the installed identity and applicable checks are verified; individual atomic file replacements do not make a multi-file installation atomic. Preserve recovery artifacts before changing live files. Work already in flight must be settled or explicitly handled under the accepted transition policy. Queue waiting and activation consume the existing deadline unless the original contract explicitly states otherwise.

Define activation dependencies as explicit predicates before execution. "No dependent execution remains active" differs from "both dependent checks passed" or "both checks produced verdicts." A stop rule can leave an unlaunched check settled for quiescence without satisfying a required successful check or producing a verdict. Preserve that distinction in receipts. An agent's interpretation, another agent's agreement, or a technical PASS cannot waive an unmet owner condition. If the recorded authority is ambiguous, keep activation or reopening paused until the authority holder resolves that condition; preserve the candidate and prior evidence.

8. Crash recovery and external side effects

8.1 Durable dispatch is not exactly-once execution

Persist the action, reservation, and dispatch intent in one transaction. A dispatcher may deliver the same intent more than once after a crash. The adapter must recognize the action ID and avoid creating another execution when one is already known.

This alone cannot guarantee exactly-once effects in an arbitrary external service. Use that service's idempotency mechanism, a reconciliation query, or an explicit prohibition on automatic replay. If none is available, a crash between performing the effect and recording the receipt leaves an unknown outcome that requires investigation.

8.2 Ownership and recovery

When multiple workers can claim work, the store must grant one current ownership claim with a lease and increasing generation or fencing value. Reject stale owners' authoritative updates. A lease expiry is not proof that the old process stopped.

Where possible, the side-effecting service or adapter must also reject stale ownership. If it cannot, do not dispatch a replacement that could duplicate or conflict with an old live execution until its status is reconciled. A database lease does not fence a remote API by itself.

On restart, load the existing root, find unsettled actions, inspect owned execution handles and artifacts, and classify outcomes as completed, failed, cancelled, or unknown. Preserve reservations while chargeable work may still be active. Resume only an admissible next action.

8.3 Compensation and evidence retention

A compensating action has its own authority, preconditions, reservation, and evidence. It is not necessarily a perfect inverse: an email cannot be unsent merely by recording a rollback. State which effects are reversible and what remains unresolved.

Retain the prior verified artifact and necessary recovery data according to the accepted retention policy. Do not log secrets to make recovery convenient. Backup and restore of controller state must preserve work identity and reconciled expenditure; restoring an older snapshot cannot restore spent allowance. If reconciliation after restore is incomplete, block new dispatch.

8.4 Moving a root to a new authority

Pause new admissions at the old authority and reconcile its spend, pending reservations, executions, and active exceptions. Import the root once with a stable migration identity and explicit authority generation. Disable old-authority admission before enabling the replacement, and reject stale clients or stale-generation requests. Keep uncertain effects reserved and visible; a migration does not resolve them.

Retain the old records as history or read-only receipts. If the new authority is unavailable, do not fall back to the old allowance file. A rollback of the authority arrangement requires the same reconciliation discipline; it cannot recreate an earlier budget. Migrate one root or covered path at a time rather than assuming a simultaneous migration of the entire estate is necessary.

9. Reviews, repeat detection, and invalidation

9.1 Two identities with different jobs

The action ID identifies an admitted execution and supports idempotent delivery. A separate assessment fingerprint determines whether another assessment is substantively equivalent.

Repeated delivery of the same admitted request returns the existing action's recorded status and result when available. Unknown status triggers reconciliation; it does not authorize another execution. A legitimate new trial or permitted retry receives its own admitted action identity and retains its relationship to the original purpose.

The fingerprint covers root, obligation set, action purpose, relevant artifact and environment identities, evidence inputs, rubric version, required reviewer role, and any predeclared trial identity. Exclude irrelevant timestamps, session IDs, provider names, and newly appended receipt text that would make every repeat look new.

Use deterministic encoding for structured fields. Do not erase meaningful whitespace or configuration values when normalizing inputs. Hashes detect identical declared inputs; they cannot establish semantic novelty. The hard budget remains necessary even when superficial rewrites evade repeat detection.

9.2 Legitimate repeat work

Reassessment may be appropriate after changed code, relevant new evidence, changed environmental conditions, an affected upstream guarantee, a declared independent review role, or a remaining replication trial. Each still requires normal admission.

Provider switching by itself is not a new review purpose. Planned independent reviewers are not reviewer shopping: record their roles before the round, charge their actual executions, and retain all findings.

Run cheap relevant regression checks as required by the accepted verification policy. A rule against repeated semantic audits must not prohibit testing changed code or checking integration behavior.

9.3 Actionable findings

Each finding records a stable ID, affected obligation, location, basis, impact, and closure criterion. Classify it as a violation, evidence gap, scope defect, or optional suggestion. Suggestions do not silently become required scope.

Closure requires verified correction, adequate evidence, or an accepted demonstration that the finding was invalid. A later report omitting it does not close it. Reviewer disagreement should lead to a discriminating check or an owner decision about intent, not repeated model selection until a favorable answer appears.

REDO means more work is requested; it is neither a finding closure nor a new allocation. It preserves the root, consumed review/action allowance, retry history, and unresolved findings. If a local hook also uses a cumulative BLOCK counter to trigger escalation, REDO cannot reset that counter. Keep cumulative events separate from the count of currently open findings, which can legitimately fall when evidence closes them. Only an authorized policy transition may change the escalation baseline, and it must retain the earlier history.

If the accepted policy counts BLOCKs since the last valid PASS, that PASS ends the applicable checker's escalation sequence under the existing policy. It does not delete cumulative history, close another checker's findings, or replenish allowance. State whether a counter measures lifetime events or a current sequence; confusing them can turn a temporary escalation into an unintended permanent ban.

Give independent reviewers the accepted contract and relevant evidence without the builder's persuasive approval narrative. Independence is a property of role and access separation, not merely choosing a different model name. Reviewers may still share model biases; use deterministic checks and appropriate human calibration for consequential judgments.

10. Bounded automatic recovery

10.1 Event-driven behavior

Consume failure events from the existing execution path. Avoid continuous model polling that creates work merely to ask whether work is needed.

```
Observe failure -> preserve evidence -> resolve incident identity
    -> classify -> check recovery contract and allowance
    -> execute permitted recovery -> verify downstream result
    -> resume, stop, or report an exact blocker
```

Repeated notifications for the same unresolved incident retain lineage and do not refill allowances. Do not merge uncertainly related failures in a way that suppresses a genuinely different incident.

10.2 Recovery contract

Specify the failure class, owned target, allowed actions, preconditions, replay-safety condition, attempt ceiling, deadline, downstream verification check, failure/compensation procedure, and authority reference.

Situation	Response
Owned worker stopped before any side effect.	Reconcile status, then use an approved bounded restart if eligible.
Transient read failure.	Use only a permitted remaining retry.
Timeout after a possible external write.	Reconcile the outcome before replay.
Failed acceptance check.	Propose a bounded repair against the obligation.
Reviewer BLOCK or missing permission.	Resolve the finding or authority question; do not treat it as an infrastructure fault.
Lock held or lock outcome uncertain.	Follow the accepted lock policy; no automatic bypass.
Unknown failure class.	Preserve evidence and stop speculative recovery.

Starting a process is not proof of recovery. Verify the downstream behavior that failed. An operation with a no-rerun policy cannot obtain another attempt by changing its label to self-healing.

11. Optional recursive improvement of agent software

11.1 Definition

The optional improvement loop generates changes to the agent's tools, context handling, adapters, or recovery strategies, evaluates them independently, and uses an authorized promoted version in subsequent improvement work. That is recursive improvement of agent software. It does not imply model-weight training or an indefinitely increasing rate of capability growth.

DGM investigates agent-code modification evaluated on coding benchmarks [S5]. AlphaEvolve combines program generation with automated evaluation and selection [S7]. These are precedents for the pattern, not evidence that any implementation of this document will improve.

11.2 Protected and editable surfaces

The candidate may modify only its isolated permitted surface. It may not modify the authority controlling it, its own budget, the authoritative evaluator, required acceptance checks, holdout answers, outcome ledger, or deployment controls. Enforce this through access boundaries, including indirect paths through tools, dependencies, or configuration.

A candidate can propose a defect report about a protected component. Correction then requires a separate authorized change and independent evaluation. Protected means outside this candidate's authority, not impossible to maintain forever.

11.3 Candidate lifecycle

```
Measured incident -> reproducible case -> declared hypothesis
    -> bounded isolated candidate -> independent comparison
    -> reject or accept for promotion -> authorized installation
    -> observe -> use recorded promoted parent for a later candidate
```

Start with one candidate at a time. Record parent version, incident and obligation IDs, allowed paths, hypothesis, exact candidate digest, evaluator/suite version, resource allocation, result, and promotion/recovery references. Preserve rejected candidates' evidence sufficiently to prevent repeated equivalent experiments; do not retain sensitive raw data unnecessarily.

A shared controller issue may justify a separate improvement root, but only with explicit allocation. It cannot launder an exhausted ordinary task into unlimited new work. Limit the improvement campaign as well as each candidate; no self-created campaigns, automatic refill, or recursive delegation around campaign ceilings.

11.4 Independent evaluation and selection

Use the incident reproducer, the existing regression suite, and independently controlled representative cases. The evaluator executes checks and owns their results. Candidate-written success logs are not evidence of execution. The DGM authors describe fabricated test logs and scoring manipulation, illustrating why these boundaries matter [S6].

Compare incumbent and candidate under equivalent workload, model, environment, and resource conditions. Declare the primary improvement target, allowed regressions, sample/trial counts, timeout treatment, and selection rule before results appear. Retain failures and inconclusive comparisons. If the evidence cannot distinguish improvement from variation under the declared protocol, keep the incumbent.

Use held-out cases deliberately: repeated adaptive access can turn them into training feedback. Limit submissions and control refresh independently. A case made visible for debugging must no longer be treated as untouched holdout evidence. Test changes and rubric corrections are versioned; affected candidate comparisons must be re-evaluated, not silently carried forward.

Do not select solely for fewer BLOCKs or lower cost. Require preservation of accepted behavior, bounded resource use, and no disallowed correctness regression. Measure both incorrect acceptance and incorrect rejection so that rejecting everything cannot win.

11.5 Promotion and inheritance

Acceptance for promotion is separate from deployment authority. The baseline policy here requires an authenticated owner release decision and the declared independent reviews. Deploy the exact evaluated artifact, verify the target state, and retain the authorized recovery path.

A failed post-promotion check stops further rollout and triggers only the specifically authorized recovery procedure. Record effects and limitations. A later candidate inherits the recorded promoted version, not unapproved changes or a model's account of what supposedly passed.

Automatic promotion is an optional policy extension outside the baseline design. It requires an explicitly approved change class, independent release authority, verified containment and recovery, and unchanged external budget controls. Successful evaluation alone cannot create that policy.

12. Provider-independent interfaces

The core protocol operates on structured proposals, actions, and observations rather than a provider's chat-message format. Models remain interchangeable only to the extent that their adapters satisfy the declared capabilities.

```
dispatch(action_request) -> execution_handle
inspect(execution_handle) -> execution_status
cancel(execution_handle) -> cancellation_status
collect(execution_handle) -> result_envelope
```

Interface data	Required content
Action request	Schema version, action/root IDs, contract revision, purpose/role, input references, allowed execution surface, limits, deadline, delivery idempotency key.
Execution handle	Adapter identity and stable handle bound to the admitted action.
Status	Running, completed, failed, cancelled, or unknown; observed time and source.
Result envelope	Schema version, action/handle identity, observed outcome, artifact references, usage with units, errors, and proposed findings where applicable.
Capability declaration	Supported cancellation, resource bounds, usage reporting, idempotency, status lookup, and isolation mechanisms.

Cancellation requested and cancellation confirmed are different states. Missing usage is unknown, not zero. Unsupported capabilities must be declared and must prevent admission into a profile that requires them. The controller validates identity and provenance; result content remains untrusted until evaluated.

Platform support is a capability too. A gate intended for one operating system should reject unsupported admission paths explicitly before mutating admission state or dispatching work. Tests on one platform do not establish lock, cancellation, or filesystem semantics on another. Support a new platform only after its claimed behavior is implemented and verified.

The initial build needs a deterministic fake adapter and one real adapter. Provider names do not belong in policy decisions. Test two differently named fake backends for protocol parity; this demonstrates interface independence, not equal model quality or proof that an unimplemented backend works.

13. Worked example

13.1 A small, isolated task

Consider an agent fixing a configuration parser in a disposable test environment. Required outcomes are that valid input still works, malformed input is rejected, and the parser performs no external writes. A reviewer role evaluates any semantic claim that the deterministic checks do not cover.

The following is a budget fragment, not a complete executable contract. All values are illustrative, not recommended universal defaults:

```
{
  "root_work_id": "example-config-parser-001",
  "contract_revision": 1,
  "phase": "implementation",
  "budget": {
    "action_slots": 12,
    "root_retry_slots": 1,
    "max_concurrency": 1,
    "wall_clock_seconds": 1800,
    "max_action_seconds": 300,
    "final_verification_slots": 2,
    "recovery_slots": 1,
    "metered_execution_allowed": false
  }
}
```

The complete contract must additionally identify the approved workspace, accepted obligations, concrete commands and expected observations, authenticated authority, review role, dependency identities, and deadline/cleanup policy. The example may not be dispatched as-is.

An initial test reproduces the malformed-input failure. The builder makes a targeted change. Relevant tests and the declared reviewer assess the resulting artifact. Every covered invocation is charged; the name repair cycle does not hide its tool calls.

Ordinary work may not consume the capacity reserved for two final-verification actions and one approved recovery. An adapter timeout consumes an admitted action; the one retry is eligible only if replay is safe and all ceilings still permit it. Repeated delivery of the original action is deduplicated rather than treated as another authorized retry.

If checks establish all required outcomes, the root passes after relevant executions settle. If the wall-clock deadline or remaining allowance prevents completion, preserve the patch, observations, and exact unresolved requirement. Do not claim success, delete the failed test, or obtain more work by opening a new chat.

13.2 A later improvement experiment

Suppose the incident exposed a reproducible adapter bug. A separately allocated campaign may test a candidate adapter against a known timeout case, ordinary completion cases, and independently controlled recovery cases. The original task's budget remains unchanged. A candidate that merely hides timeout errors fails the outcome checks. A genuinely improved candidate still needs the applicable release decision before installation.

14. Implementation sequence

Stage	Deliverable	Falsifiable exit
1	Actual dispatch-path inventory and deployment profile.	Every claimed model/tool path and unsupported capability is identified.
2	Stable identity, contracts, budgets, and transactional store.	Concurrent reservations and restarts cannot create capacity.
3	Durable dispatch, idempotent ingestion, and supervised execution.	Duplicate delivery, crashes, and deadlines produce the specified states.
4	Evidence binding, finding closure, and repeat policy.	Unsupported PASS, stale evidence, and reviewer shopping cannot close work.
5	Real-adapter integration and operator-visible status.	Inadmissible work never reaches the covered real launcher.
6	One approved recovery contract.	A controlled failure is recovered without duplicate effects or extra allowance.
7	Optional isolated improvement experiment.	A candidate is independently compared without access to its judge or controls.
8	Authorized promotion and a later generation.	The exact approved version becomes a recorded parent while external controls remain unchanged.

Use the existing store and testing tools where possible. Keep the first deployment single-coordinator unless actual concurrency or availability requirements justify additional coordination machinery.

Persist implementation receipts as the work happens. A final summary cannot replace absent execution evidence. State which stages are complete; a core-only installation must not advertise automatic recovery or recursive improvement.

15. Requirements and acceptance cases

The matrix is an implementation checklist, not a claim that a reference implementation or these tests have been executed. Core requirements are P01-P14; P15 covers recovery; P16-P18 cover the optional improvement extension. Activating an extension requires its applicable cases as well as the core cases.

15.1 Requirement index

ID	Requirement	Acceptance cases
P01	Preserve work identity and authorized lineage.	T01, T02
P02	Protect accepted scope and authority.	T03, T04, T45, T47, T50
P03	Admit against root and parent budgets transactionally.	T05, T06, T07, T46, T49
P04	Account for retries, reservations, and protected capacity.	T08, T09, T10
P05	Enforce deadlines and disclose unsupported bounds.	T11, T12, T13
P06	Persist dispatch intent and deduplicate delivery/results.	T14, T15
P07	Reconcile crashes and unknown side effects.	T16, T17, T18
P08	Bind evidence to applicable artifacts and dependencies.	T19, T20, T52
P09	Distinguish progress, scope changes, and acceptance.	T21, T22, T23, T51
P10	Preserve findings and independent review obligations.	T24, T25
P11	Distinguish repeats from declared trials or new evidence.	T26, T27
P12	Separate trusted control from untrusted proposals.	T28, T29
P13	Expose provider-independent capability limits.	T30, T31
P14	Derive status and restart state from durable records.	T32, T33, T48
P15	Recover only through an authorized bounded procedure.	T34, T35, T36
P16	Isolate improvement candidates and independent evaluation.	T37, T38, T39
P17	Bound campaigns and preserve honest comparisons.	T40, T41
P18	Authorize promotion and retain version lineage/recovery.	T42, T43, T44

15.2 Acceptance scenarios

Use isolated stores, fake clocks, controlled worker failures, and deterministic adapters for protocol tests. Add integration tests for each real adapter's claimed capabilities. Run destructive or side-effecting cases only in a specifically authorized disposable environment.

ID	Scenario	Required observation
T01	Session, model, or reviewer changes for the same work.	Root, spent allowance, and findings persist.
T02	Worker invents a new root or child allocation.	No capacity without authorized lineage and parent allowance.
T03	Required test is removed or required scope becomes optional.	Reject unapproved baseline weakening.
T04	Extension is authorized once and delivered twice.	Apply its increment once without resetting spent allowance or scope.
T05	Concurrent workers reserve the last slot.	At most one reservation succeeds.
T06	Child allowance remains but parent allowance is exhausted.	No dispatch.
T07	A nominal review spawns additional model/tool calls.	All calls are covered and charged or bounded by a verified adapter limit.
T08	A reviewer returns nothing, a coordinator fails before review admission, or BLOCK is followed by REDO.	Charge admitted actions; charge no review unit for a never-admitted review. Preserve time/executor costs, verdict distinctions and history; no automatic refund or new capacity.
T09	Ordinary work would consume final-check or recovery capacity.	Reject borrowing; eligible protected actions transfer reservation only once.
T10	Concurrent billable calls and delayed usage reports approach a ceiling.	Outstanding upper-bound reservations prevent overshoot within the claimed billing scope.
T11	Long setup, restart, clock rollback or a stale detached-executor heartbeat occurs.	Deadline is not extended; external supervision reconciles liveness, blocks new child dispatch and grants no automatic replacement.
T12	Worker ignores its timeout, root exits with a live child, cancellation fails, or the next action has a shorter limit.	Supervisor enforces the applicable minimum deadline and verifies owned cleanup. Failed or uncertain termination cannot produce a successful stop receipt.
T13	Remote cancellation or usage bounds are unsupported.	Capability limitation is exposed; a profile requiring that guarantee rejects the action.
T14	Crash occurs before or after dispatch-intent delivery.	Reservation survives and duplicate delivery does not create another known execution.
T15	Identical or conflicting duplicate results arrive.	Identical ingestion is idempotent; conflict becomes a reconciliation issue.
T16	Remote write may have completed before a crash.	Reconcile actual effects or remain unknown; no blind replay.
T17	Lease expires while an old worker remains alive.	Stale owner cannot commit authoritative state; replacement side effects wait unless effectively fenced.

ID	Scenario	Required observation
T18	Store is unavailable or restored from an old snapshot.	No new admission until accounting is reconciled; running work follows independent supervisor limits.
T19	Model supplies a success log without collector/evaluator evidence.	Required obligation remains unresolved.
T20	Relevant upstream evidence changes while an unrelated file also changes.	Affected obligations invalidate; unrelated valid evidence remains usable.
T21	Assessment only covers part of the accepted scope.	INCOMPLETE rather than PASS.
T22	New defect appears or scope is authorized to change.	Record it and separate denominator changes from verified closure.
T23	Acceptance is met but a sibling could still modify the artifact.	Disable new productive work and settle the sibling before declaring the root COMPLETED.
T24	A new reviewer omits a previous BLOCK finding.	Finding remains open until its closure criterion is satisfied.
T25	Two required independent review roles assess the same artifact.	Both planned roles are admissible within budget; neither erases the other's findings.
T26	Equivalent assessment is requested with a new provider or timestamp.	Reject a duplicate when decision-relevant inputs and declared role/trial are unchanged.
T27	New relevant evidence or a predeclared replication trial exists.	Reassessment is eligible but still subject to ordinary admission and aggregation rules.
T28	Retrieved text or tool output requests extra permissions.	Treat it as data; authority and budget remain unchanged.
T29	Worker targets protected store, evaluator, or control configuration.	Actual access boundary blocks the change; otherwise enforcement claim fails.
T30	Two fake provider adapters return equivalent neutral results.	Core policy behavior is identical.
T31	Envelope is malformed or reports unknown usage as success.	Reject invalid structure and retain unknown accounting rather than assuming zero.
T32	Model status disagrees with stored action/evidence records.	Operator status follows authoritative records.
T33	Existing work has unknown history or a repeated adoption request.	Preserve unknowns, require transition authority, and prevent a second initial allocation.
T34	Repeated events refer to an unresolved recoverable incident.	Deduplicate incident lineage without creating new recovery allowance.
T35	Process restart succeeds but downstream behavior is still broken.	Recovery remains failed or incomplete.

ID	Scenario	Required observation
T36	Lock failure, missing authority, or exhausted budget is labeled self-healing.	No bypass, hidden rerun, or refill.
T37	Candidate modifies its own judge, acceptance checks, or result ledger.	Isolation blocks mutation, including indirect tool/configuration paths.
T38	Candidate passes the reproducer but fails required regression cases.	Keep the incumbent; no promotion.
T39	Candidate exploits repeated holdout exposure or changes scoring after results.	Enforce the declared evaluation-access and versioning policy; no unsupported comparison carries forward.
T40	Candidate has capacity but campaign or concurrency allowance is exhausted.	No dispatch or recursively created replacement campaign.
T41	Lower cost comes from skipping requirements or rejecting valid tasks.	No improvement claim; measure incorrect acceptance and rejection.
T42	Passing candidate lacks release authority or installed bytes differ.	Promotion blocked.
T43	A promoted version fails its defined post-promotion check.	Stop rollout and execute only the authorized recovery path; retain history.
T44	A later improvement experiment begins after promotion.	It uses the recorded promoted parent and retains external controls and expenditure.
T45	Worker writes an approval row, replays a signed decision, or can invoke the supposed owner signing credential.	Reject unauthenticated authority and duplicate increments; the claimed separation fails if the worker controls the issuer or verifier.
T46	A root is imported twice or the old client admits work during or after authority cutover.	No duplicate capacity; pending reservations persist; the stale authority cannot admit, and an outage creates no local fallback.
T47	A one-use exception is replayed, expires, gains an authorized successor, or overlaps another eligible entry.	Consume once; retain history; reject replay and conflicting eligibility. Inactive history must not block an authorized successor. A valid same-checker PASS may end its declared escalation sequence without refilling allowance.
T48	A clerical record error, missing authority, dependency failure or uncertain cancellation is reported repeatedly.	Route to the appropriate recorder, owner or operator; preserve outcome and limits; deduplicate notifications. A clerical correction must not confer new rights or erase usage.
T49	Preparatory critique, repeated deterministic tests, or the controller's own build attempts to run without bounds.	Apply the declared category and root limits; the candidate cannot grant itself extra bootstrap work.
T50	A later draft proposes tighter limits or a capability is implemented but inactive.	Preserve the actual admission terms and accurately report lifecycle progress; claim operational enforcement only for activated, verified behavior.

ID	Scenario	Required observation
T51	An authorized task completes through the covered real launcher.	Admit once, execute within applicable bounds, collect actual evidence, settle related executions, and close without unnecessary owner escalation.
T52	A packet omits a cited range, exceeds its input envelope, or receives unrelated and material source changes.	Reject incomplete or oversized packets before review admission. Validate compaction and freeze its identity. Preserve unrelated evidence; invalidate affected evidence; honor effective revocations without resetting allowance.

T51 is the required complete positive integration case. A controller that rejects every action fails the product objective even if negative cases pass. Core requirements include the applicable T45-T52 cases as well as their earlier mapped cases; these additions do not make the optional improvement extension mandatory. For conditional activation cases, include a stopped, unlaunched dependency: it may satisfy an explicit quiescence predicate but must not satisfy a required PASS or verdict predicate.

16. Measurement and rollout

16.1 Outcome measures

Track verified completion, incorrect acceptance, incorrect rejection, time and resource use per verified completion, actions since last verified closure, duplicate admissions prevented, repeated incidents, scope deferrals, owner interventions, and post-promotion regressions. Include failed and timed-out items in aggregate reporting.

Do not treat fewer audits or more commits as standalone success. Compare equivalent workloads and preserve the required outcome. Report sample size, environment, model and evaluator versions, trial policy, and uncertainty when describing an improvement. Separate synthetic controller-test results from results on real workloads.

16.2 Enablement process

Begin with deterministic fixtures, then test the real adapter in a bounded environment. Observation-only operation can help identify integration gaps, but it is advisory and cannot establish enforcement. Enable covered dispatch rejection only under the deployment's release authority and verified recovery procedure.

Report covered and uncovered paths, supported cancellation/resource controls, persistence guarantees, remaining limitations, and the exact deployed version. A native chat interface outside the broker remains outside the claimed admission control unless integration actually covers it.

For each capability, record PROPOSED, IMPLEMENTED, ACTIVATED, or VERIFIED with its evidence and version. Implementation progress must not be erased merely because activation has not occurred. Operational enforcement is claimed only for activated behavior that has been verified in the declared environment. Later drift can invalidate that verification.

Count coverage using a declared denominator: launcher entry points, underlying execution paths, and capability dimensions are different measurements. Two launchers sharing one admission function are not two independently implemented control boundaries. An admission check, a tested execution deadline, findings persistence, and authentic grant issuance each need their own evidence.

A useful first milestone demonstrates budget exhaustion, restart persistence, duplicate delivery handling, unauthorized-grant rejection, supported deadline enforcement, and one successful authorized task on one real

path. Its receipts must identify the actual controlled launcher and implementation version. Do not widen that result into a claim about other paths or optional recovery/improvement capabilities.

Keep the operator view concise: outcome, unresolved obligation IDs, relevant evidence, remaining allowance, outstanding executions, and next admissible action or exact decision. Status should not depend on a model retelling the conversation.

16.3 Lessons from local trials

Local development exposed several failure patterns that informed this revision. These anonymized observations are not controlled benchmarks, independent reproduction or evidence that the full proposed controller works. Underlying operational records remain private.

Observed failure	Lesson incorporated into the design
A parser allowed one historical exception per check type, so an expired entry prevented recording a new approved attempt.	Separate retained history from current eligibility; verify administrative archival without refunding consumption.
A review produced no verdict before its timeout; a different coordinator failed before any review was admitted.	Separate elapsed effort, review admissions and reviewer verdicts. Both can waste time, but they consume different units.
A termination wrapper reported a kill without checking success and did not reconcile children after root exit.	Test the execution boundary and cleanup result, rather than counting the presence of a timeout command as enforcement.
A format error was routed to the owner like a missing approval.	Let an authorized recorder make uniquely evidenced clerical corrections; reserve owner decisions for actual choices.

The observed review timeout does not establish whether input size, service latency or another cause dominated. No causal or efficiency claim is made from that incident. Run-level outcomes still pending at preparation time are excluded from completed-result claims.

17. Limitations and responsible claims

This design is not a formal proof of arbitrary software correctness. Natural-language scope, evidence relevance, review quality, and task decomposition can still be wrong. Deterministic tests can be incomplete; model reviewers can share biases; external services can violate assumptions.

The system can enforce declared workflow boundaries only where its access controls, adapters, accounting, and supervision actually apply. It cannot promise that a disconnected external operation stopped, that an unbounded service cannot bill more, or that a privileged worker cannot bypass controls it can rewrite.

The recursive-improvement extension is a constrained experimentation process. It may produce no useful candidates, plateau, or cost more than it saves. Its benefit must be measured; publishing this design does not demonstrate such benefit.

Before claiming an implementation conforms, supply the filled deployment profile, applicable acceptance evidence, integration coverage, operational limitations, and version identity. Before claiming effectiveness, supply workload-level results. Neither a polished document nor a passing sample configuration is that evidence.

18. Sources and their relevance

The following primary sources were checked while preparing this edition. They support the practices summarized here; they do not endorse the proposed protocol or its implementation details.

- **[S1] OpenAI, Harness engineering: leveraging Codex in an agent-first world.** Describes repository knowledge, observable execution, and mechanically enforced boundaries. https://openai.com/index/harness-engineering/
- **[S2] Anthropic, Building effective agents.** Discusses simple workflows, environmental feedback, and explicit stopping conditions. https://www.anthropic.com/engineering/building-effective-agents
- **[S3] Anthropic, Demystifying evals for AI agents.** Distinguishes observed outcomes from transcripts and discusses deterministic, model-based, and human evaluation. https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents
- **[S4] Anthropic, Harness design for long-running application development.** Reports experiments with work contracts, separate evaluators, and removing orchestration when it no longer helps. https://www.anthropic.com/engineering/harness-design-long-running-apps
- **[S5] Zhang et al., Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents.** Investigates modification and empirical evaluation of agent code. https://arxiv.org/abs/2505.22954
- **[S6] Sakana AI, The Darwin Godel Machine: AI that improves itself by rewriting its own code.** Documents the approach and examples of fabricated tool-use evidence and scoring manipulation. https://sakana.ai/dgm/
- **[S7] Google DeepMind, AlphaEvolve: A Gemini-powered coding agent for designing advanced algorithms.** Describes generated programs, automated evaluation, and selection. https://deepmind.google/blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/

The protocol requirements, example allocations, state names, and acceptance matrix are proposed engineering choices. They should be tested and adapted explicitly rather than represented as a validated industry standard.