BehaviorSpec: A Declarative Contract for Governing AI Agent Behavior

Rick Buonincontri
Solsta Inc.
rick@solsta.io

Abstract

AI agents combine models, prompts, and tools into a composite system with a behavioral capability surface that is rarely declared in a single, reviewable artifact. As these systems move into regulated, customer-facing, and business-critical environments, the absence of a single, canonical mechanism for declaring, reviewing, and binding agent capability at deployment boundaries creates material governance, compliance, and operational risk. As a result, undeclared capability expansion, environment drift, and limited reproducibility frequently occur at promotion boundaries.

This paper introduces BehaviorSpec, a declarative governance model for managing an agent's behavioral capability surface across staging and production environments. BehaviorSpec requires two artifacts: (1) a mandatory `behavior.intent`, which defines declared purpose, scope, tool permissions, model policy, constraints, and promotion requirements, and (2) a mandatory `behavior.lock`, generated at promotion time, which binds the approved intent to immutable artifact identities such as model versions and tool releases. Together, these artifacts enforce a promotion invariant: no agent may be deployed into a controlled environment unless its declared behavioral intent has been reviewed, approved, and cryptographically bound to the exact runtime artifacts.

Inspired by declarative infrastructure practices such as Kubernetes manifests, Terraform configuration and state separation, and dependency lockfiles, BehaviorSpec extends infrastructure governance discipline to agent behavior. Enforcement is embedded in CI/CD or equivalent promotion control planes and introduces no required runtime coupling. The result is a minimal, architecture-neutral governance primitive for making agent behavioral capability explicit, reviewable, auditable, and reproducible at scale.

BehaviorSpec is not a configuration pattern or framework feature. It defines a promotion invariant independent of runtime execution frameworks, cloud providers, and orchestration systems. The schema scope is deliberately open to accommodate variation in agent complexity and risk classification.

Keywords: AI, agents, governance, compliance, production, control plane, agent deployment, declarative intent, behavioral artifact, promotion invariant

1. Introduction

AI agents increasingly operate in production environments where they interpret natural language, invoke tools, retrieve data, and generate outputs that influence users, downstream services, or external systems. As these systems transition from experimental prototypes to production-critical infrastructure, their behavioral capability surface becomes operationally

significant. Seemingly minor changes to prompts, model routing, tool permissions, or configuration may alter effective capability, expand privilege, or modify risk exposure.

Traditional software governance assumes that deployed artifacts are deterministic binaries whose behavior is defined by version-controlled source code and configuration. Agentic systems depart from this assumption. Their behavioral capability surface emerges from a composition of language models, prompts, tool integrations, orchestration logic, and environment-specific policy. This composition is often implicit and distributed across repositories, configuration files, and infrastructure settings. Review occurs indirectly through code diffs rather than through an explicit declaration of intended capability.

This creates a governance gap at promotion boundaries. When an agent is promoted from development to staging or production, there may be no single artifact that answers foundational questions:

- What is the agent permitted to do?
- What actions are explicitly out of scope?
- Which tools and model classes are allowed?
- What approvals are required before exposure to real users or data?

BehaviorSpec addresses this gap by introducing a declarative contract for governing the agent's behavioral capability surface. It requires that declared behavioral intent be formalized, reviewed, and versioned prior to promotion, and that the resulting deployment be bound to immutable artifact identities. BehaviorSpec does not attempt to regulate runtime inference semantics. Its role is to enforce discipline at promotion boundaries, where governance and risk control are most tractable.

As large language models remain stateless and bounded by token-window constraints, agent systems increasingly externalize memory and context into persistent repositories. This architectural evolution reinforces the need to distinguish between dynamic context management and declared behavioral capability. As systems become more modular and mount heterogeneous resources dynamically, the risk of unreviewed capability expansion increases. BehaviorSpec provides a formal governance layer to control that expansion.

2. Parallels to Established Practices

BehaviorSpec intentionally mirrors patterns that are already familiar in modern infrastructure and software delivery systems.

- Kubernetes manifests**.** A Kubernetes YAML file declares the desired state of a deployment. The cluster reconciles actual state against that declaration. Similarly, `behavior.intent` declares the desired behavioral capability surface of an agent prior to promotion.
- Terraform configuration and state. Terraform separates declarative infrastructure configuration from the resolved state file that captures actual resource bindings. `behavior.intent` plays the role of declarative configuration, and `behavior.lock` plays the role of a resolved binding that records concrete artifact identities.

- Package lockfiles. Dependency manifests describe allowed packages and lockfiles pin exact versions to guarantee reproducibility. `behavior.lock` serves an analogous function for model versions, tool releases, and configuration digests.

These analogies are illustrative rather than equivalent. Kubernetes governs runtime reconciliation of infrastructure state. Terraform governs infrastructure resource provisioning. BehaviorSpec governs promotion-time declaration and binding of behavioral capability. It operates at the transition of declared capability into controlled environments which is a different control boundary.

3. Problem Statement

An agent is a system that combines one or more language models with prompts, tool integrations, orchestration logic, and environment-specific configuration in order to perform tasks autonomously.

The behavioral capability surface of an agent is the complete and reviewable set of actions the agent is authorized and technically able to perform within a specific environment.

It is determined solely by:

- The declared functional scope of the agent
- The tools the agent is permitted to invoke and the access level of each tool
- The models the agent is allowed to use and the routing rules governing their selection
- The operational constraints imposed by the environment, such as data classification, memory policy, logging requirements, and timeout limits

The behavioral capability surface excludes implementation details that do not affect externally observable behavior.

In practice, this behavioral capability surface is often implicit. It is encoded across prompt text, model endpoints, configuration fragments, and infrastructure definitions. There is typically no required, unified declaration of declared behavioral intent at the moment of promotion into a controlled environment.

This absence produces three systemic risks:

1. Undeclared capability expansion. New tools, elevated privileges, or broader model classes may be introduced without explicit review of their implications.
2. Environment drift. Staging and production may differ in tool releases, model versions, or configuration without a clear binding to declared intent.
3. Irreproducibility. The precise behavioral configuration associated with a past deployment may not be reconstructed with confidence.

The core problem addressed in this paper is the lack of a formal mechanism that binds declared behavioral intent to promotion workflows and to immutable runtime artifacts.

4. Definitions

4.1 Behavioral Intent

A behavioral intent is a human-authored, declarative specification that describes the declared purpose, scope, tool permissions, model policy, constraints, and promotion requirements of an agent ("`behavior.intent`").

4.2 Behavioral Lock

A behavioral lock is a machine-generated artifact produced at promotion time that binds a reviewed behavioral intent to immutable artifact identities for models, tools, and configuration components ("`behavior.lock`").

4.3 Promotion Gate

A promotion gate is a policy-enforced boundary at the entry to a controlled environment such that an agent may not be deployed into that environment unless a validated behavioral intent exists, all declared approval requirements have been durably recorded, and a corresponding behavioral lock has been generated and bound to immutable artifact identities.

4.4 Controlled Environment

A controlled environment is any environment in which promotion requires explicit approval and is subject to audit or rollback semantics (e.g., staging, production). Development environments operating in isolation from live data and production systems are not controlled environments for the purposes of this specification.

4.5 BehaviorSpec

BehaviorSpec is the composition of a mandatory behavioral intent and a mandatory behavioral lock, together forming a promotable contract that governs an agent's behavioral capability surface in controlled environments.

The unit of governance is the BehaviorSpec pair. A `behavior.intent` file in isolation declares acceptable capability boundaries but does not constitute a promotable contract. Behavioral differences between deployments are resolved at the lock layer, where exact artifact identities are bound. Two deployments sharing an identical `behavior.intent` but carrying different `behavior.lock` artifacts represent distinct, independently reviewable capability states.

These definitions establish the minimal formal substrate required to treat behavioral capability as a versioned, reviewable, and reproducible artifact.

Figure 1. Agent promotion lifecycle: environments, gates, and audit record

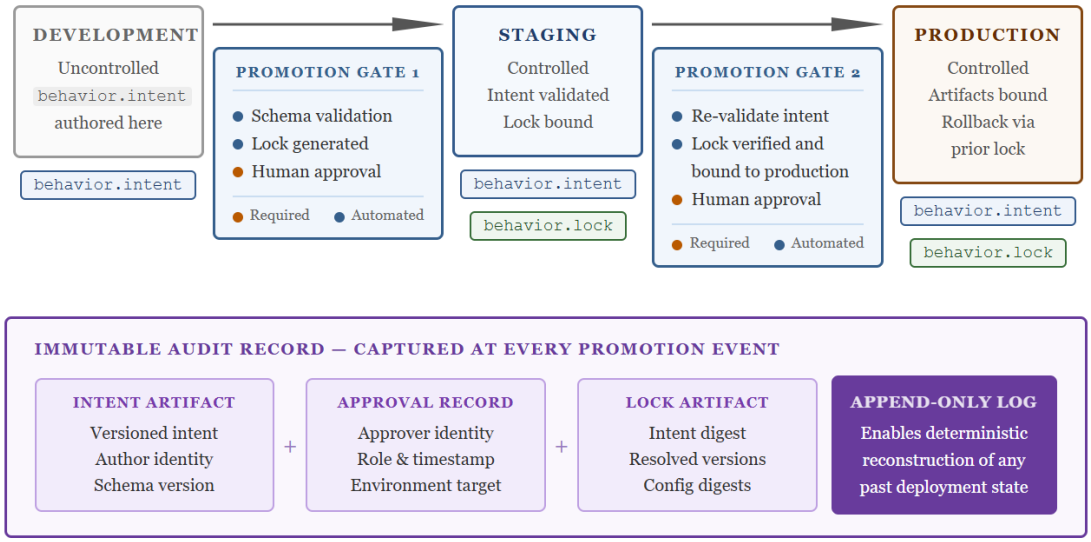


Figure 1. Agent promotion across development, staging, and production environments. Each controlled-environment boundary requires a promotion gate enforcing human approval and schema validation. At the staging gate a lock is generated and cryptographically bound to resolved artifact identities. At the production gate the staging lock is verified and appended with production-specific provenance. Every promotion event contributes an immutable record to the audit log, enabling forensic reconstruction of any prior deployment's behavioral capability surface.

5. Related Work

The engineering of agentic systems has generated work in three adjacent areas: structured context engineering, production-grade workflow design, and modular model deployment. These efforts improve performance, reliability, and operational efficiency. BehaviorSpec builds on these foundations but addresses a distinct concern: governance of declared behavioral intent at promotion boundaries.

5.1 Context Engineering

Structured approaches to prompt and context management treat context as an engineered artifact rather than informal text. Incremental update models aim to avoid degradation associated with wholesale rewriting and emphasize disciplined accumulation of knowledge.

BehaviorSpec shares the premise that context is an engineering surface but shifts the objective from optimization to governance. Rather than improving output quality through adaptive context, BehaviorSpec requires that declared behavioral intent be formalized and reviewed before promotion.

5.2 Production-grade Architectures

Production agent architectures emphasize determinism, modular decomposition, explicit tool interfaces, and separation between orchestration and execution. These practices reduce runtime ambiguity and improve debuggability. BehaviorSpec complements these practices by

introducing a promotion-time control surface. It governs what must be declared and approved before agents are permitted to execute in controlled environments.

Kartakis et al. introduce evaluation-gated deployment as a precondition for operational trust in agentic systems, advocating that promotion into production should require structured evaluation evidence rather than relying solely on process controls (Google, Nov 2025). BehaviorSpec is informed by this principle at the governance layer. The promotion invariant creates the structural conditions under which evaluation evidence could be recorded alongside approvals and bound to the corresponding `behavior.lock`. A principled framework for what constitutes sufficient evaluation evidence for agentic systems is an emerging area identified for future work.

5.3 Modular Model Deployment

Modular and heterogeneous model architectures recognize that agent subtasks often expose a narrow subset of model capability. This perspective reinforces the notion that the behavioral capability surface is structured and bounded.

BehaviorSpec formalizes this boundedness through explicit declaration of allowed tools and models within the behavioral intent specification.

6. Architectural Model

BehaviorSpec consists of two required layers:

1. `behavior.intent:` the declarative specification of declared behavioral intent
2. `behavior.lock:` the immutable binding of that intent to resolved artifact identities

The separation is intentional. The intent layer captures semantic and governance-relevant information subject to human review. The lock layer captures mechanical resolution at promotion time and ensures reproducibility.

Both artifacts are mandatory for deployment into controlled environments.

Figure 2. The BehaviorSpec contract: declared intent and immutable lock

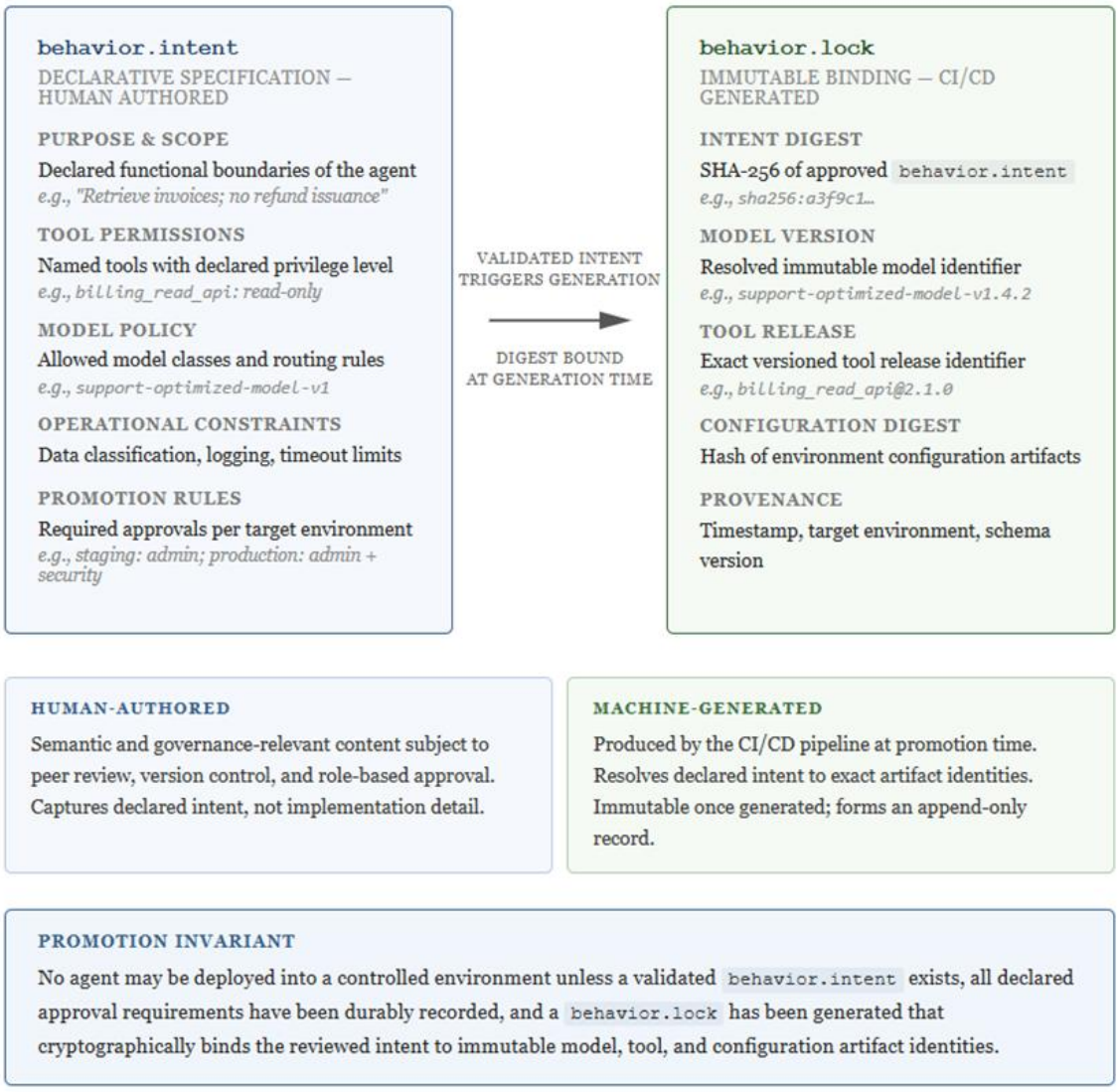


Figure 2. BehaviorSpec requires two artifacts at every controlled-environment promotion. The `behavior.intent` is human-authored and declares the agent's behavioral capability surface. The `behavior.lock` is generated by the CI/CD pipeline and binds the approved intent to resolved, immutable artifact identities. Together they enforce the promotion invariant.

6.1 Design Principles

The design of `behavior.intent` adheres to five principles:

1. Declarative: It describes permitted capability, not implementation detail.
2. Human reviewable: It must be interpretable in standard diff workflows.
3. Minimally complete information: It must contain sufficient information to assess risk without embedding runtime configuration noise.
4. Deterministic validation: Its structure must be machine verifiable.
5. Environment specificity: Promotion requirements must be declared per environment.

6.2 `behavior.intent` Schema Overview

The `behavior.intent` schema defines the declared behavioral capability surface. It includes:

- Metadata (identifier, version, owner)
- Purpose statement
- Scope (in-scope and out-of-scope categories)
- Tool policies (tool identifiers and privilege levels)
- Model policies (allowed models and routing constraints)
- Operational constraints
- Promotion rules (required approvals and checks per environment)

The schema does not attempt to encode semantic meaning of tools or models. It governs declared interfaces and permitted capability classes. `behavior.intent` must declare permitted model classes and routing modes (e.g., single-model, heterogeneous, fallback). Introduction of additional model classes constitutes a material capability expansion.

6.2.1 Tool Contract Integrity

Each tool referenced in `behavior.intent` must correspond to a versioned interface contract whose schema is immutable or cryptographically versioned. `behavior.lock` binds to the exact contract identity and release identifier resolved at promotion time. Changes to tool schemas that alter invocation surface or output structure constitute material behavioral capability changes and require re-approval.

6.3 Schema Evolution and Compatibility

BehaviorSpec anticipates evolution of the `behavior.intent` schema over time. To preserve reproducibility and auditability, schema versioning must be explicit and machine verifiable.

Each `behavior.intent` must declare the schema version to which it conforms. Schema versions define structural expectations, required fields, and validation rules. Changes to the schema must be versioned in a manner that allows deterministic validation of historical intent artifacts.

`behavior.lock` binds not only to the content digest of `behavior.intent`, but also to the specific schema version under which that intent was validated. This ensures that a historical lock can be interpreted unambiguously, even if the schema evolves in subsequent editions.

Schema evolution requires explicit migration pathways. When introducing breaking structural changes, organizations must define upgrade or transformation procedures that convert prior `behavior.intent` artifacts into the new schema version. Promotion gates should reject intent artifacts whose declared schema version is unsupported in the target environment.

By making schema versioning explicit and binding it at lock generation time, BehaviorSpec preserves long-term interpretability and prevents ambiguity arising from silent schema drift.

6.4 Authoring Guidance for `behavior.intent`

YAML is proposed for `behavior.intent` because it:

- is widely understood across DevOps and platform engineering teams
- represents hierarchical structure clearly
- produces readable, reviewable diffs in pull requests
- is easily validated

However, YAML is not mandatory. The critical requirement is deterministic serialization so that `behavior.lock` can compute stable digests.

Figure 3. Minimal compliant behavior.intent — customer-support billing agent (Section 8.5)

behavior.intent — HUMAN-AUTHORED · YAML

```yaml
schema_version: "1.0"
id:             customer-support-billing-agent
version:        "2.1.0"
owner:          platform-engineering@example.com

purpose: Retrieve invoices and explain billing line items.

scope:
  in_scope:
    - Invoice retrieval and display
    - Payment status lookup
  out_of_scope:
    - Refund issuance or adjustment
    - PII export or transmission

tools:
  - id: billing_read_api
    version_constraint: ">=2.0.0"
    privilege: read
  - id: customer_lookup_api
    version_constraint: ">=1.4.0"
    privilege: read

models:
  allowed:
    - support-optimized-model-v1
  routing: fixed

constraints:
  data_classification: internal
  logging:             required
  memory_persistence:  none

promotion:
  staging:
    required_approvals:
      - role: environment-administrator
        minimum: 1
  production:
    required_approvals:
      - role: environment-administrator
        minimum: 1
      - role: security-reviewer
        minimum: 1
```

VERSION RANGE

`version_constraint` declares an acceptable range. Exact version binding is resolved by `behavior.lock` at promotion time. Routine patches require no intent update.

EXPLICIT BOUNDARIES

`out_of_scope` states prohibited action classes explicitly. Any capability expansion requires a declared scope change and re-approval before promotion.

ENVIRONMENT-SPECIFIC APPROVALS

The `promotion` block declares required roles per target environment. Production requires both administrator and security reviewer approval.

Figure 3. A minimal compliant `behavior.intent` *for the billing agent in Section 8.5. Exact artifact versions are deferred to the* `behavior.lock`*, generated at promotion time.*

The `behavior.intent` declares acceptable version ranges for tools rather than pinning exact versions. In the example above, billing_read_api is constrained to >=2.0 rather than a specific release. This is deliberate. The intent layer governs declared capability (what the agent is permitted to do and within what boundaries), and not the mechanical resolution of exact artifact identities. Exact version binding is the responsibility of the `behavior.lock`, which resolves declared constraints to specific, immutable release identifiers at promotion time. This separation mirrors the Terraform pattern described in Section 2, wherein a configuration declares the desired infrastructure state and acceptable resource parameters and the state file records the exact resources provisioned against that declaration. Similarly, `behavior.intent` declares the acceptable capability envelope while `behavior.lock` records the exact artifacts bound within that envelope. A `behavior.intent` that pinned exact tool versions would conflate these two concerns, coupling the governance declaration to a specific release and requiring intent updates for routine tool patch cycles that do not alter declared capability. Keeping version constraints in the intent layer and exact resolution in the lock layer preserves the reviewability of the intent artifact while ensuring reproducibility through the lock.

6.5 Practical Authoring Guidance for `behavior.intent`

The `behavior.intent` file defines the agent's behavioral capability surface. Its purpose is to declare what the agent is authorized and able to do within a given environment. Effective authoring therefore requires discipline and restraint.

- Keep it semantic. Do not embed implementation details or transient runtime configuration. The file should describe declared capability, not execution mechanics.
- Declare boundaries explicitly. The 'out of scope' section should clearly state prohibited action classes. Ambiguity at the boundary undermines review quality.
- Align declared scope with tool permissions. If functional scope expands, tool access and privilege levels must be updated explicitly.
- Treat privilege elevation as a material change. Introducing write or execute permissions, expanding data access, or allowing new model classes should trigger heightened review.
- Version with intent. Changes to the intent version should correspond to meaningful shifts in the behavioral capability surface, not cosmetic edits.
- Avoid environment leakage. Environment-specific operational constraints may vary, but the declared capability surface must remain explicit per environment rule set.

Organizations may benefit from standardized templates aligned to risk tiers, such as low-risk internal automation, medium-risk operational workflows, and high-risk customer-facing or regulated agents.

6.6 `behavior.lock` and Reproducibility

At promotion into any controlled environment, a `behavior.lock` is generated and required. The lock includes:

- A digest of the validated `behavior.intent`
- Resolved model version identifiers
- Tool release identifiers
- Relevant configuration digests
- Timestamp and target environment

The lock transforms a reviewed declaration into a reproducible deployment contract. Without a mandatory lock, reproducibility and artifact integrity cannot be guaranteed.

When an agent is promoted from staging to production, the production `behavior.lock` does not re-resolve artifact identities. The model versions, tool release identifiers, and configuration digests recorded in the staging lock are carried forward unchanged. The production gate verifies the cryptographically bound artifact identities in the staging lock, then appends to that lock to bind it to production by recording the production-specific provenance. This ensures that the artifact configuration validated in staging is identical to the configuration deployed in production. Introducing new artifact identities at the production gate would undermine environment parity and is not permitted under the promotion invariant.

6.7 Rollback Semantics

Rollback in BehaviorSpec is defined as redeployment of a previously generated `behavior.lock` into a controlled environment. No mutation of `behavior.intent` may affect a controlled environment unless a new `behavior.lock` is generated through the authoritative promotion pathway. Lock artifacts are immutable and form an append-only historical record of approved behavioral capability states, enabling deterministic rollback and forensic reconstruction.

6.8 Promotion Invariant

BehaviorSpec defines a promotion invariant for controlled environments such as staging and production.

An agent may be deployed into a controlled environment only when all of the following conditions are satisfied:

1. The `behavior.intent` file has passed structural and schema validation.
2. The declared promotion rules for the target environment have been satisfied.
3. All required approvals for that environment have been durably recorded.
4. A `behavior.lock` has been generated from the approved intent and binds that intent to immutable model, tool, and configuration artifact identities.

Under these conditions, the following guarantee holds:

For every agent deployed into a controlled environment, there exists a reviewed `behavior.intent` and a corresponding `behavior.lock` such that the runtime artifacts deployed into that environment match the exact artifact identities recorded in the lock at promotion time. This invariant is a process guarantee, not a behavioral guarantee. It ensures that what was declared was reviewed, that what was reviewed was approved, and that what was approved was bound to the exact artifacts deployed. It does not assert that the declared intent accurately described the system's actual behavior, nor that the bound artifacts behaved in production as their declarations anticipated. The distinction between artifact binding and behavioral compliance is structural and intentional and is addressed further in Section 8.3.

The name "BehaviorSpec" reflects an aspiration as much as a guarantee. Behavior in agentic systems cannot be fully specified through artifacts. It emerges from the interaction of models, prompts, tools, and runtime context in ways that no static declaration can completely anticipate. The immutable binding of the artifacts that define the behavioral capability envelope is the closest tractable approximation to behavioral control at the promotion boundary. BehaviorSpec does not eliminate the gap between declared capability and actual behavior, but it closes the gap between undeclared capability and the production environment. This is the gap that governance can most directly address.

That process compliance at the promotion boundary is nonetheless consequential. A governance failure that produces a bad outcome is more easily correctable when a canonical record of declared intent exists than when accountability is distributed across informal controls. The promotion invariant creates the structured record against which runtime monitoring, evaluation evidence, and audit can assess whether declared intent and actual behavior diverged over time. It establishes the accountability boundary upon which behavioral compliance efforts can build.

6.9 Evaluation Evidence

A promotion gate that requires only a well-formed artifact and recorded approvals is necessary but not sufficient. Without evidence that the deployed system's actual behavior is consistent with its declared intent, the promotion invariant is reduced to a procedural check that the artifact existed and someone signed off. That is a meaningful control, but it is incomplete. Structured processes often create the appearance of oversight without substantive validation is a known failure mode in conventional change management. A promotion gate without evidence is at risk of replicating this.

Evaluation evidence is the mechanism by which declared intent is tested against actual behavior before a promotion boundary is crossed. It is what distinguishes a reviewed declaration from a validated one.

For controlled environments, and particularly for production, promotion policy should require evaluation evidence commensurate with the agent's risk classification. At minimum, evaluation evidence should address three questions:

1. Capability boundary adherence. Does the agent's observed behavior remain within the declared scope? Are out-of-scope action classes actually excluded from observed outputs?

2. Tool and privilege exercise. Do the tools the agent invokes in practice correspond to the tools declared in `behavior.intent`? Are privilege levels exercised consistent with declared access levels?

3. Constraint compliance. Do operational constraints declared in `behavior.intent`, such as data classification boundaries, memory policy, and output restrictions, hold under representative task conditions?

These questions do not require exhaustive formal verification. They require structured evaluation runs against representative task distributions, with results recorded as promotion artifacts and bound alongside the `behavior.lock`.

The form of evaluation evidence will vary with agent complexity and risk tier. A low-risk internal automation agent may satisfy evaluation requirements through a defined set of functional tests with recorded outputs. A high-risk customer-facing or regulated agent may require adversarial probing, red-teaming against declared out-of-scope action classes, and independent review of evaluation results before production promotion is permitted.

What is not acceptable, for agents operating in controlled environments, is the absence of any evaluation evidence as a promotion requirement. A `behavior.intent` that has never been tested against actual system behavior is a declaration of intent, not a validated contract. The promotion invariant binds artifacts to declarations. Evaluation evidence binds declarations to observed reality. Both are required for the promotion gate to function as substantive governance rather than structured record-keeping.

Defining a principled framework for what constitutes sufficient evaluation evidence across agent risk tiers is a non-trivial problem. The framework's design depends on agent architecture, task distribution, and organizational context, and is identified as a priority area for future work. Organizations should define minimum evaluation requirements proportionate to agent risk classification and enforce them as promotion gate conditions alongside artifact validation and approval recording. But in no case should an organization treat evaluation evidence as optional when crossing a controlled environment boundary.

7. Layered Governance Model

Behavioral governance for agentic systems operates across three distinct layers. Conflating these layers obscures accountability boundaries and weakens architectural clarity. BehaviorSpec is intentionally positioned at the promotion layer and should be understood within this broader context.

Layer 1: Infrastructure – Execution Substrate Integrity

The infrastructure layer provides foundational guarantees upon which all higher controls depend. It governs artifact integrity, identity and credential management, environment isolation, and deployment controls. It answers whether the artifact was built from approved source, deployed to the intended environment, and whether the integrity of what is running can be verified. These controls ensure that what was approved is what was deployed. They do not define what behavioral capabilities are acceptable in the first place.

Layer 2: Promotion – Behavioral Contract Governance

The promotion layer governs declared behavioral capability across environments. This is the layer where BehaviorSpec operates. It defines permitted model and tool classes, declared privilege scope, accessible data domains, and the behavioral deltas that require review. It answers whether the capability surface expanded, whether privilege escalation was introduced, and whether the change was reviewed and approved. BehaviorSpec does not inspect individual runtime actions or mediate tool calls. Its role is to constrain and formalize the behavioral envelope before the system executes in a controlled environment. If the runtime layer governs actions, the promotion layer governs the action surface.

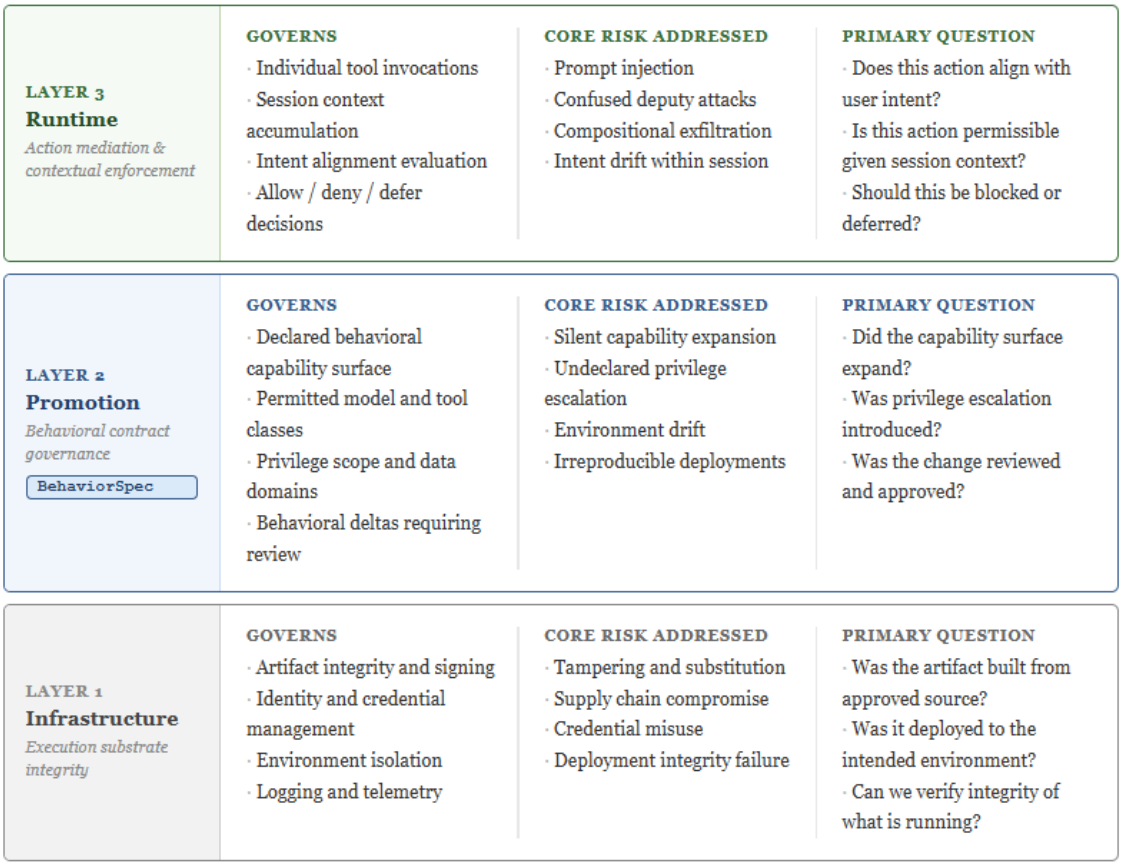
Layer 3: Runtime – Action Mediation and Contextual Enforcement

The runtime layer governs individual actions during execution. It intercepts tool invocations, accumulates session context, evaluates actions against policy and intent alignment, and enforces allow, deny, or defer decisions. It answers whether a specific action aligns with user intent and is permissible given session context. Runtime controls mitigate prompt injection, confused deputy attacks, compositional exfiltration, and intent drift. However, runtime governance assumes the existence of a capability surface. It does not determine whether a new model class, tool class, or privilege scope should have been introduced into production in the first place.

Separation of Responsibilities

Each layer governs a distinct class of risk. Infrastructure controls cannot detect behavioral expansion. Runtime controls cannot retroactively determine whether a capability was appropriately approved. Promotion controls cannot prevent runtime misuse within an approved envelope. No single layer is sufficient, but together they form a coherent governance stack. Figure 4 illustrates the relationship between layers and the risks each addresses.

Figure 4. Layered governance model for agentic systems



LAYER	GOVERNS	WHAT IT CANNOT DO ALONE
Infrastructure	Artifact integrity and identity correctness	Cannot detect behavioral capability expansion
Promotion	Declared behavioral capability surface	Cannot prevent runtime misuse within an approved envelope
Runtime	Individual action execution and intent alignment	Cannot determine whether a capability was appropriately approved

Figure 4. Behavioral governance for agentic systems operates across three distinct layers. The infrastructure layer ensures artifact and deployment integrity. The promotion layer — where `BehaviorSpec` operates — governs declared behavioral capability at environment boundaries. The runtime layer mediates individual actions during execution. No single layer is sufficient; together they form a coherent governance stack.

Implementation Across Layers

This paper provides a concrete implementation of the promotion layer through BehaviorSpec. The infrastructure and runtime layers are described functionally because their governance problems are distinct and are being addressed by parallel bodies of work.

At the infrastructure layer, supply-chain security frameworks provide structured approaches to artifact provenance, build integrity, and deployment controls. These frameworks define the enforcement substrate that promotion-layer guarantees depend on. BehaviorSpec's enforcement preconditions, artifact immutability, promotion pathway exclusivity, and restricted production write access, are problems that infrastructure-layer controls are designed to solve.

At the runtime layer, emerging work in autonomous action management addresses the interception, evaluation, and mediation of individual agent actions during execution. For example, Errico's recent Autonomous Action Runtime Management (AARM) represents one concrete specification of runtime governance at the action boundary (arXiv: 2602.09433, Feb 2026).

BehaviorSpec is designed to compose with both layers. It does not compete with infrastructure hardening or runtime enforcement. It occupies the promotion boundary between them, where declared behavioral capability is authorized before it is permitted to exist in controlled environments, and where the record of that authorization is created before runtime controls are asked to enforce it.

A coherent governance stack requires equivalent rigor at all three layers. The promotion layer cannot provide meaningful guarantees if the infrastructure beneath it is uncontrolled, and runtime controls cannot compensate for capability that was never reviewed at promotion. The three layers are mutually dependent. BehaviorSpec's contribution is to make the promotion layer explicit, formal, and composable with the controls that surround it.

8. Trust Model and Boundaries

BehaviorSpec governs declared behavioral capability at promotion boundaries. It formalizes and binds declared intent before deployment into controlled environments. It does not verify runtime semantic correctness, evaluate model outputs, or prevent adversarial input. Its guarantees apply strictly to the promotion pathway and the integrity of declared behavioral artifacts.

BehaviorSpec does not prescribe execution topology, workflow decomposition, or orchestration semantics. Deterministic or agentic execution strategies must operate within the behavioral capability surface declared in `behavior.intent` and bound in `behavior.lock`.

8.1 Enforcement Preconditions

The promotion invariant defined by BehaviorSpec depends on strict enforcement of deployment discipline. Its guarantees hold only if the following preconditions are satisfied:

1. Artifact immutability**.** Model versions, tool releases, and configuration artifacts referenced in `behavior.lock` must be immutable or cryptographically versioned such that their contents cannot change without altering their identifiers.
2. Promotion pathway exclusivity. Deployment into controlled environments must occur solely through a single authoritative promotion pathway that generates and records `behavior.lock` artifacts. Side-channel deployments, manual artifact substitution, or direct runtime mutation invalidate the invariant.
3. Restricted production write access. Direct write access to production runtime artifacts must be administratively constrained. Runtime artifacts must not be modifiable outside the promotion mechanism that binds intent to lock.

Satisfying these preconditions is a distinct engineering and organizational problem from the BehaviorSpec governance model. Establishing artifact immutability, enforcing promotion pathway exclusivity, and restricting production write access requires infrastructure controls and organizational policy that operate beneath the promotion layer.

BehaviorSpec specifies what must be enforced at the promotion boundary. It does not specify how the environment is hardened to make that enforcement durable. These preconditions represent an implementation problem that commercial platforms, managed deployment infrastructure, and enterprise tooling are well positioned to address, in the same way that the operational requirements of runtime governance specifications are expected to be realized through commercial and open-source implementations.

This mirrors the scope boundary the paper maintains elsewhere. BehaviorSpec does not govern runtime action mediation, because that is a separate layer with its own controls. Similarly, it does not govern the infrastructure hardening that makes promotion invariants enforceable, because that too is a separate layer. The governance stack is coherent only when all three layers operate together.

In fully managed deployments, the promotion system may enforce artifact immutability and promotion pathway exclusivity directly. In self-managed or shared responsibility deployments, organizations must ensure artifact immutability and restrict direct production write access in order for the promotion invariant to hold.

8.2 Assumptions

Within the enforcement constraints above, BehaviorSpec assumes:

- Immutable or versioned tool releases.
- Stable or snapshot-bound model identifiers.
- A single authoritative promotion pathway.
- No artifact substitution after lock generation.

These assumptions are structural to the governance guarantees provided. If tools, models, or deployment artifacts are mutable without version changes, reproducibility cannot be assured.

8.3 Risk Mitigation and Scope Boundary

BehaviorSpec mitigates governance-level risks associated with undeclared behavioral capability and promotion-time drift. It does not provide runtime security guarantees. The artifact binding guarantee does not address the accuracy of the declaration itself. An agent operator may author a `behavior.intent` that incompletely or inaccurately describes the system's actual capability surface. BehaviorSpec enforces that the declaration existed, was reviewed, and was bound to deployment artifacts. Whether the declaration was accurate is a question for the evaluation evidence provided at promotion time and for runtime monitoring in production. BehaviorSpec creates the structured record that makes those assessments tractable but it does not substitute for them.

The distinction is summarized below.

Concern	Mitigated by BehaviorSpec	Rationale
Undeclared capability expansion	Yes	Behavioral intent must be declared, reviewed, and approved prior to promotion.
Environment drift between staging and production	Yes	`behavior.lock` binds declared intent to immutable artifact identities at promotion time.
Irreproducible deployments	Yes	Lock artifacts form an immutable record of resolved model and tool versions.
Unauthorized behavioral changes via sanctioned deployment pathway	Yes	Promotion gates require validated intent and lock binding before deployment.
Prompt injection at runtime	No	BehaviorSpec does not evaluate or constrain runtime inputs.
Adversarial model output behavior	No	Model alignment and output safety are runtime and training concerns.
Divergence between declared intent and actual runtime behavior	No	Declaration accuracy is not verifiable at promotion time. Runtime monitoring, evaluation evidence, and periodic audit are the appropriate controls for assessing alignment between declared and actual behavior over time.
Compromise of tool implementation	No	Tool integrity depends on supply-chain and infrastructure security controls.

Concern	Mitigated by BehaviorSpec	Rationale
Model provider weight changes under unchanged identifier	No	Reproducibility depends on provider version stability assumptions.
Infrastructure-level compromise	No	Infrastructure security remains outside the governance layer.

BehaviorSpec should therefore be understood as a governance-layer control that formalizes declared behavioral capability at promotion boundaries. It complements, but does not replace, runtime safeguards, supply-chain protections, infrastructure hardening, or adversarial defense mechanisms.

This distinction has practical consequence when runtime controls fail or are bypassed. Guardrails, IAM policies, and cloud policy engines provide prevention at the invocation boundary but offer no structured recovery path when an incident occurs.

BehaviorSpec addresses the recovery dimension directly. Because lock artifacts are immutable and form an append-only record of approved behavioral capability states, rollback is deterministic. Any prior good state may be fully reconstructed from its corresponding `behavior.lock`, and redeployment to that state requires no forensic reconstruction of what was previously declared or approved. This makes BehaviorSpec a complement to prevention-oriented runtime controls rather than a substitute for them. Prevention without a structured recovery path is incomplete governance. BehaviorSpec closes that gap at the promotion boundary.

Reproducibility in BehaviorSpec refers to artifact and configuration determinism at promotion boundaries. It does not imply identical inference outputs across executions, which remain subject to model stochasticity.

8.4 Control Plane Boundary

BehaviorSpec operates at a distinct control boundary from cloud-native guardrails, identity systems, and runtime configuration mechanisms. The distinction is summarized below.

Control Layer	Governs	Primary Question Answered	When It Acts
IAM / Access Control	Principal access to tools and APIs	Who may invoke this resource?	Runtime
Cloud Guardrails / Policy Engines	Allowed API categories or resource usage	Is this invocation permitted under policy?	Runtime
Endpoint Configuration / Version Pinning	Model endpoint or configuration parameters	Which version is being executed?	Runtime
CI/CD Pipelines	Build and deployment process integrity	Has the artifact passed required process checks?	Promotion
BehaviorSpec	Declared behavioral capability surface	Is this capability part of the agent's approved intent?	Promotion

BehaviorSpec governs the promotion of declared behavioral capability into controlled environments. This is a distinct control point in the lifecycle that is not addressed by other layers:

1. IAM determines whether an identity may call a tool at runtime. BehaviorSpec determines whether that tool is part of the agent's approved behavioral capability surface before promotion.
2. Endpoint configuration may pin a model version at execution time. BehaviorSpec binds the declared model class to a specific immutable artifact identity at promotion time.
3. Cloud guardrails may block unsafe API calls dynamically. BehaviorSpec prevents undeclared capability from being promoted into a controlled environment in the first place.

These layers are complementary but not interchangeable. Infrastructure controls regulate access and execution at runtime. BehaviorSpec formalizes and binds the intended behavioral capability surface before that capability is permitted to exist in staging or production.

By operating at the promotion boundary rather than the invocation boundary, BehaviorSpec establishes a clear governance invariant:

> *No undeclared behavioral capability surface may enter staging or production through the authoritative deployment pathway, independent of cloud provider, orchestration system, or execution framework.*

This separation preserves compatibility with existing cloud-native controls while defining a distinct governance plane focused on declared intent and immutable artifact binding.

9. Compliance and Control Framework Alignment

BehaviorSpec is a governance-layer mechanism. While it does not replace enterprise security programs, it directly supports multiple control families within widely adopted assurance frameworks, including SOC 2 (Trust Services Criteria) and ISO/IEC 27001.

This section summarizes alignment at the control-family level. Detailed, line-by-line evidence mappings can be maintained as companion audit artifacts.

9.1 Alignment with SOC 2 Trust Services Criteria (Common Criteria)

BehaviorSpec primarily strengthens controls related to change management, access restriction, configuration integrity, and auditability.

BehaviorSpec produces structured evidence of process compliance at the promotion boundary. It does not assert that declared intent was accurate at the time of authorship, nor that bound artifacts behaved in production as their declarations described. Auditors do not expect those guarantees from a change management control. They expect evidence that behavioral changes were declared, reviewed, approved, and bound before deployment, and that when incidents occur, the organization can reconstruct what was authorized, identify what deviated, and demonstrate a defined response. BehaviorSpec directly supports all three of those expectations.

9.1.1 CC5 / CC8: Control Activities and Change Management

SOC 2 requires that system changes be authorized, tested, documented, and approved prior to implementation.

BehaviorSpec enforces:

- Explicit declaration of behavioral changes through versioned `behavior.intent`
- Role-based approval requirements embedded in promotion rules
- Deterministic generation of `behavior.lock` at promotion time
- Immutable binding between approved intent and deployed artifacts

Any expansion of behavioral capability, such as introducing new tools, elevating privileges, or changing model classes, requires modification of declared intent and re-approval before deployment. This directly supports formal change authorization, approval traceability, and release documentation expectations under CC5 and CC8.

BehaviorSpec defines structured human oversight as a required condition at the point of behavioral change, specifying role-based approval before promotion into controlled environments.

9.1.2 CC6: Logical Access Controls

SOC 2 requires that logical access to systems and configuration changes be restricted and attributable.

BehaviorSpec supports this control by:

- Restricting deployment into controlled environments to an authoritative promotion pathway
- Requiring durable recording of approvals tied to specific intent versions
- Preventing deployment of undeclared behavioral configurations

Promotion rights are role-scoped rather than artifact-scoped, and each promotion event binds declared intent to immutable artifacts. This structure reinforces logical access control around behavioral configuration changes.

9.1.3 CC4 / CC7: Monitoring and Incident Response

SOC 2 expects ongoing monitoring and the ability to investigate control deficiencies and incidents.

Because `behavior.lock` records exact artifact identities, BehaviorSpec enables:

- Reconstruction of the precise behavioral capability surface associated with a past deployment
- Deterministic comparison between declared intent and deployed artifacts
- Structured rollback and change history analysis

This strengthens auditability and incident reconstruction capabilities, particularly for behavioral or configuration-related incidents. BehaviorSpec creates durable, versioned records of declared behavioral capability and resolved artifact identity, supporting traceability obligations and post-incident analysis.

9.1.4 CC2: Quality of Information

SOC 2 emphasizes reliability and integrity of information used in control activities.

`behavior.intent` is:

- Schema-validated
- Version-controlled
- Human-reviewed
- Cryptographically bound to deployment artifacts

This improves reliability of configuration information used in deployment decisions and reduces reliance on implicit or dispersed configuration fragments.

9.2 Alignment with ISO/IEC 27001

BehaviorSpec also supports several Annex A control domains in ISO/IEC 27001, particularly those addressing operational security and controlled change. As with SOC 2 alignment, BehaviorSpec's contribution to ISO 27001 control domains operates at the governance and artifact layer. It strengthens the evidence base for structured change management and access control over behavioral configuration. Runtime behavioral correctness and artifact integrity remain within the scope of complementary security and testing controls.

(1) A.12 — Operations Security

ISO 27001 requires disciplined change management and configuration control.

BehaviorSpec provides:

- Explicit declaration of behavioral configuration
- Structured promotion gates
- Immutable binding between approved declarations and deployed artifacts

This strengthens configuration management and controlled operational change.

(2) A.9 — Access Control

ISO 27001 requires that access to systems and configuration be restricted according to business requirements.

By restricting deployment to controlled promotion pathways and requiring role-based approvals before behavioral changes reach production, BehaviorSpec reinforces access control over behavioral configuration changes.

(3) A.14 — System Acquisition, Development, and Maintenance

ISO 27001 emphasizes secure development practices and controlled transitions to production.

BehaviorSpec embeds governance into the development lifecycle by requiring:

- Formalized behavioral intent before promotion
- Environment-specific approval requirements
- Artifact binding at deployment

This aligns agent deployment with structured system development lifecycle controls.

9.3 Alignment with EU AI Act (Deployer Obligations)

BehaviorSpec supports organizational obligations under the EU AI Act for deployers of high-risk AI systems by strengthening documentation, traceability, and change governance at the promotion boundary.

It contributes to:

- Controlled configuration management of AI systems
- Traceability of deployed versions and artifact identities
- Structured human approval before operational deployment
- Reproducible reconstruction of deployed behavioral capability

9.4 Control Boundary Clarification

BehaviorSpec enforces structured change authorization for behavioral capability at promotion boundaries.

BehaviorSpec does not govern runtime context assembly. It governs the declarative authorization of behavioral capability before that capability is permitted to exist in a controlled environment. Runtime context pipelines may dynamically select, compress, and inject context under token constraints, but those mechanisms operate within the boundaries defined by the promoted BehaviorSpec.

It does not assert control over:

- Model training correctness
- Runtime inference behavior
- Prompt injection resistance
- Infrastructure compromise

Those concerns remain within the scope of broader security architecture, supply-chain controls, and runtime safeguards.

BehaviorSpec should therefore be understood as a governance-layer enhancement that strengthens enterprise control posture around agent behavioral configuration, rather than as a comprehensive runtime security mechanism. The scope boundary with respect to declaration accuracy and runtime behavioral correctness is addressed in Section 8.3

9.5 A Simple Example

To illustrate operational behavior, consider a customer-support agent initially deployed with read-only billing access. The organization authors a `behavior.intent` scoped to a low-privilege, single-domain use case.

9.5.1 Initial Deployment

The `behavior.intent` declares:

- Scope: Retrieve invoices and explain line items
- Tools: billing_read_api (read)
- Models: support-optimized-model-v1
- Promotion rule: Single environment administrator approval for staging; administrator plus security approval for production

At production promotion, a `behavior.lock` is generated containing:

- Digest of the approved `behavior.intent`
- Resolved model version identifier
- Exact release version of billing_read_api

This ensures that the deployed agent's behavioral capability surface corresponds exactly to the reviewed declaration.

9.5.2 Capability Expansion Scenario

Suppose the organization decides the agent may issue refunds. The organization's policy is that changes to write privilege access to a billing system require elevated approvals.

This change requires:

1. Updating scope to include refund issuance
2. Adding billing_write_api with write privileges
3. Elevating approval requirements in the production promotion rule
4. Regeneration of `behavior.lock` at promotion

The expansion cannot reach production without modification of declared behavioral intent and re-binding to resolved artifacts. The promotion invariant forces visibility of privilege escalation and tool expansion. Prompt-level changes that broaden the agent's effective scope, such as expanding the categories of questions the agent will answer or relaxing constraints on its responses, also constitute material capability changes and require corresponding updates to the declared intent.

9.6 Policy Composition and Organizational Controls

BehaviorSpec governs per-agent declared behavioral intent. It does not replace organization-wide policy systems. In the billing example above, a global policy might prohibit write access to financial APIs entirely, in which case the capability expansion scenario in Section 9.5.2 would be rejected at the promotion gate regardless of whether the required approvals were obtained.

In practice, policy composition operates at two levels:

1. Global Policy Constraints. Organization-wide rules (e.g., disallowed tool classes, restricted model providers) may validate or constrain `behavior.intent` declarations during promotion.
2. Agent-Specific Declarations. Individual `behavior.intent` files declare permitted behavioral capability within those global boundaries.

Validation may therefore include both schema-level checks and policy-as-code evaluation. BehaviorSpec is intentionally composable: it provides a per-agent governance primitive that higher-level policy engines may inspect, validate, or restrict.

This separation preserves flexibility while enabling centralized risk control.

10. Existing Controls Are Insufficient

Modern software delivery environments already incorporate version control, CI/CD pipelines, identity systems, architectural review boards, and change-management workflows. Many production teams rigorously govern behavioral change through code review conventions, approval workflows embedded in CI configuration, and structured design reviews.

These controls collectively govern pieces of the capability surface, but no single control owns it, and that distributes accountability in ways that are difficult to audit, difficult to roll back, and difficult to defend under scrutiny. They do not create a canonical, versioned object that represents declared behavioral capability at the promotion boundary.

The following sections examine why each existing control category, taken individually, fails to close this gap. The claim is not that these controls are inadequate in what they do. It is that governing declared behavioral capability as a first-class object at the promotion boundary is not what they do. This is the gap BehaviorSpec addresses.

10.1 Context Engineering Systems Do Not Bind Behavior as a First-class Object

Context engineering frameworks improve storage, retrieval, compression, and injection of information into model sessions. They materially increase runtime coherence and reproducibility.

However, they do not:

- Declare the behavioral authority of the agent as a single, versioned object
- Require explicit approval for expansion of tool surface area
- Bind model class changes to promotion review semantics
- Create a durable artifact representing "what behavioral capability was approved"

They govern context assembly. They do not govern declared capability at promotion.

BehaviorSpec operates at the promotion boundary, not at runtime context construction.

This distinction is structural:

Layer	Responsibility
Context Engineering Infrastructure	Governs assembly and lifecycle of contextual artifacts
BehaviorSpec	Governs authorization and immutable binding of declared behavioral capability surface prior to deployment

BehaviorSpec complements context engineering infrastructure by establishing a promotion invariant that no undeclared behavioral capability surface may enter a controlled environment through the authoritative promotion pathway.

10.2 Version Control Does Not Declare Behavioral Scope

Source repositories track changes to prompts, orchestration code, configuration files, and tool integrations. Diffs reveal textual modification, but they do not require authors to declare the intended behavioral capability surface explicitly.

A prompt change that broadens authority, introduces new tool usage, or alters scope may appear as a minor textual edit. The repository records what changed, but not whether declared capability expanded or whether that expansion was reviewed.

10.3 CI/CD Pipelines Do Not Formalize Behavioral Capability by Default

Continuous integration and deployment systems validate builds, run tests, and automate environment transitions. In their default configuration, they govern process and artifact integrity, answering questions such as "does the code compile", "do tests pass", and "has the artifact been built from approved source". A pipeline can confirm all of these things without requiring a structured declaration of what the agent is permitted to do. Without a formal intent artifact, promotion gates operate on build artifacts rather than on declared behavioral capability.

This is not a limitation of CI/CD architecture. It is a consequence of what pipelines are asked to enforce. A pipeline step that requires a validated `behavior.intent`, checks approval records, and generates a `behavior.lock` at promotion time is still a CI/CD pipeline. BehaviorSpec is designed to be enforced exactly this way. It is a governance artifact the pipeline is extended to require.

The distinction matters for adoption. Organizations do not need to replace existing delivery infrastructure to implement BehaviorSpec. They need to extend their promotion gates to treat declared behavioral capability as a required artifact alongside build outputs and test results. The pipeline enforces the invariant. BehaviorSpec defines what the invariant is.

What current pipelines lack, without that extension, is any structural requirement that behavioral capability be declared, reviewed, and bound before promotion. A pipeline that does not require `behavior.intent` and `behavior.lock` will promote agents whose capability surface was never explicitly authorized. BehaviorSpec closes that gap by defining what the promotion gate must require, not by replacing the infrastructure that enforces it.

10.4 IAM Is Not Behavioral Authorization

Identity and access management systems restrict which principals may call specific APIs or tools. They answer the question of who may invoke a resource. They do not declare whether a given tool invocation falls within the intended behavioral capability surface of a particular agent. An agent may possess valid credentials for a tool without that capability being explicitly declared, reviewed, or approved as part of its intended function.

10.5 Infrastructure Pinning Is Not Capability Governance

Pinning a model version or endpoint configuration improves reproducibility of inference behavior. It does not specify the agent's intended purpose, permitted action classes, or

prohibited domains of operation. A model may remain constant while its effective behavioral capability surface expands through new tool integrations or scope changes.

10.6 Runtime Enforcement Does Not Govern Promotion

Runtime controls make execution safer. They do not decide what capabilities are allowed to ship.

Traditional observability systems help teams understand what happened. They are invaluable for investigation and response. But they operate after the system is already running, and after whatever capabilities were deployed are already present in the environment.

More sophisticated runtime enforcement systems go further. They can intercept actions before they execute and evaluate them against policy. They may block, modify, escalate, or defer specific actions in real time. That is meaningful protection at the action boundary.

Even so, these systems assume the capability surface already exists and determine whether a particular action should proceed. They do not determine whether a new model class, tool integration, privilege expansion, or domain access should have been introduced into production at all.

In every case, the capability has already crossed the promotion boundary. Runtime systems control how that capability is used. They do not control how it is introduced or expanded over time.

BehaviorSpec addresses that earlier decision point. Before deployment, what capabilities is this agent declaring? Were those capabilities reviewed? Were they explicitly approved? Are they immutably bound to the artifact being promoted?

Runtime enforcement governs execution. BehaviorSpec governs what is permitted to execute in the first place.

10.7 Governance Fragmentation Increases with Scale

The controls described in sections 10.1 through 10.6 are not inadequate in all contexts. In small systems, early-stage deployments, or single-agent environments with limited blast radius, existing review processes may provide sufficient oversight. BehaviorSpec is not a prerequisite for experimentation, and the paper does not argue that every agent requires formal promotion governance from day one.

The more useful frame is what happens as systems scale. Governance fragmentation that is manageable in a single-agent, single-team context becomes a material liability when the number of agents increases, when agents operate across multiple controlled environments, when platform ownership is distributed across teams, or when regulatory and audit obligations require structured evidence of behavioral change control.

At that point, the absence of a canonical behavioral capability artifact produces compounding problems. Accountability for what any given agent is authorized to do is distributed across whoever authored the relevant prompts, whoever configured the tool integrations, whoever approved the last deployment, and whoever maintains the infrastructure

pinning. When those are different people, or when that history spans months of incremental change, reconstructing the authorized capability surface at any prior point in time requires forensic effort rather than artifact lookup. Incident response slows. Audit preparation becomes expensive. Rollback decisions lack a clear reference point.

BehaviorSpec's value scales with the complexity of the environment it governs. A single `behavior.intent` and `behavior.lock` pair adds modest overhead in a simple deployment. Across a fleet of agents operating in regulated, customer-facing, or business-critical environments, the canonical artifact record becomes the difference between governance that is auditable by design and governance that is reconstructed under pressure.

11. Conclusions and Future Work

Agentic systems are crossing from experimental tooling into production-critical infrastructure. That transition exposes a governance gap that existing controls do not close. Version control, CI/CD pipelines, IAM systems, and runtime enforcement each govern a piece of the operational picture. None of them owns the declared behavioral capability surface as a versioned, reviewable, promotable artifact. The result is fragmented accountability at the boundary where it matters most.

BehaviorSpec closes that gap by applying a discipline that software engineering has relied on for decades: declare intent explicitly, bind it to resolved artifacts immutably, and enforce the binding at every promotion boundary. This is not a novel theoretical contribution. It is the deliberate application of known change management practice to a domain where that practice has not been formally instantiated. The contribution is the instantiation itself, and the recognition that the absence of it creates material operational, compliance, and governance risk as agentic systems scale.

The promotion invariant is the paper's core claim. When `behavior.intent` and `behavior.lock` are both mandatory for deployment into controlled environments, undeclared behavioral capability cannot enter those environments through the authoritative promotion pathway. That guarantee is bounded: it is an artifact binding guarantee, not a behavioral compliance guarantee. Declaration accuracy and behavioral compliance are questions for evaluation evidence and runtime monitoring. BehaviorSpec creates the structured record that makes those assessments tractable and that makes incident response, rollback, and audit reconstruction deterministic.

The schema is intentionally minimal. The right scope of a `behavior.intent` depends on agent complexity, risk classification, and organizational context. A normative schema definition is identified as a priority area for community contribution, and the author is actively engaging practitioners, platform teams, and enterprise AI teams working on production agentic systems to develop it through operational experience.

Future work includes empirical validation in large-scale deployments, formal verification of promotion invariants, deeper integration with policy-as-code systems, development of a principled framework for evaluation evidence requirements at promotion boundaries, and exploration of standardization pathways and cross-vendor adoption models.

BehaviorSpec is the specification layer of a broader production control problem. The enforcement preconditions described in Section 8.1 and the runtime governance layer described in Section 7 represent an implementation surface the authors are actively developing. Organizations interested in schema collaboration, reference implementation, or a managed enforcement substrate for BehaviorSpec are encouraged to contact the author directly.

The underlying premise is simple. Governance discipline that is well understood in conventional software delivery has not yet been carried forward into agentic systems. BehaviorSpec proposes a minimal, architecture-neutral mechanism for doing so. The cost of adopting it is low. The cost of not adopting it, as agents assume greater operational responsibility, is not.

12. References and Influences

BehaviorSpec draws conceptual influence from established practices in declarative infrastructure, secure software delivery, and production-grade AI system operationalization. The following work informed its design principles.

Declarative Infrastructure:

- Kubernetes Documentation. Concepts of declarative state management, environment promotion, and reconciliation
- Terraform Documentation. Infrastructure-as-code workflows, plan/apply discipline, and state binding
- Lockfile and Software Bill of Materials (SBOM) practices. Deterministic artifact resolution and dependency binding

Secure Software Delivery and Supply Chain:

- NIST Secure Software Development Framework (SSDF). Structured change management and artifact integrity controls
- SLSA (Supply-chain Levels for Software Artifacts)
  Provenance, build integrity, and immutable artifact guarantees

Production Agent Operationalization:

- Bandara et al. *A Practical Guide for Designing, Developing, and Deploying Production-Grade Agentic AI Workflows* (arXiv: 2512.08769, Dec 2025).
  Deterministic orchestration, structured tool invocation, and environment-aware deployment patterns
- Kartakis, et al. *Prototype to Production* (Google, Nov 2025).
  Evaluation-gated deployment, CI/CD enforcement, and operational trust in agentic systems

Context Adaptation and Agent Memory:

- Zhang, et al. *Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models.* (arXiv: 2510.04618, Oct 2025)

Context Engineering:

- Xu, et al. *Everything is Context: Agentic File System Abstraction for Context Engineering* (arXiv: 2512.05470, Dec 2025)

Runtime Governance:

- Errico, H., *Autonomous Action Runtime Management (AARM): A System Specification for Securing AI-Driven Actions at Runtime* (arXiv: 2602.09433, Feb 2026)