Governance and Authoring:

The Legislative Process of Behavioral Law

Bachi (aka Bhash Ganti) Contact: bachipeachy@gmail.com

Abstract

This paper formalizes the governance mechanics of protocol-governed systems. Building on the structural taxonomy defined in Paper 3 [Bachi, 2026c], it specifies how behavioral law is proposed, validated, ratified, versioned, and amended.

We define the legislative lifecycle of artifacts, the separation between Authoring and Governance layers, schema enforcement mechanics, version immutability guarantees, and constitutional amendment procedures. Governance is treated not as documentation review but as a structural enforcement layer with machine-verifiable constraints.

This paper does not address runtime execution semantics (Paper 6), protocol artifact semantics (Paper 5), or computational universality (Paper 2). Its purpose is institutional: to formalize how behavioral law is created and made authoritative.

Categories: Software Engineering (cs.SE), Programming Languages (cs.PL), Software Architecture

Keywords: governance, authoring, artifact lifecycle, schema validation, version immutability, constitutional amendment, legislative process, behavioral law

---

1. Position in the Series

Paper 1 introduced the WHAT/HOW separation and established layers and concerns as foundational constructs [Bachi, 2026a]. Paper 2 demonstrated that constitutional constraint is compatible with universal computation [Bachi, 2026b]. Paper 3 formalized the eight layers and ten concerns as a structural taxonomy with explicit invariants [Bachi, 2026c].

This paper focuses exclusively on the **Authoring and Governance layers**, and on the transformation:

$$\text{Authoring} \rightarrow \text{Governance} \rightarrow \text{Protocol}$$

This paper answers a single question:

> *What makes a behavioral artifact authoritative?*

The answer involves three concepts: **lifecycle**, **validation**, and **authority**. An artifact becomes authoritative when it traverses a defined lifecycle, passes structural validation, and receives ratification from the appropriate authority.

---

2. The Constitutional Separation of Authoring and Governance

2.1 Distinction

Authoring and Governance are structurally separate layers for a constitutional reason: they serve different functions in the legislative process.

Authoring produces behavioral proposals. It is the layer where human intent is expressed in protocol-declarative form. Artifacts in the Authoring layer are drafts—they express what someone wishes the system to do, but they carry no authority.

Governance validates admissibility. It is the layer where proposals are examined against constitutional rules. Governance does not create behavior; it determines whether proposed behavior is admissible under the system's constitution.

If Authoring and Governance collapse into a single layer, governance becomes procedural rather than structural. The distinction is not organizational convenience; it is constitutional necessity.

Formally:

$$\text{Authoring}(a) = \text{express\_intent}(a)$$

$$\text{Governance}(a) = \text{validate\_admissibility}(a)$$

These functions compose:

$$\text{Ratified}(a) = \text{Governance}(\text{Authoring}(a))$$

But they must not be conflated.

2.2 Legislative Analogy

The relationship between Authoring, Governance, and Protocol mirrors the legislative process without rhetorical embellishment. Define:

Term	Definition
Proposal	An authored artifact instance in draft state
Ratification	Governance validation success
Law	A Protocol-layer artifact

A proposal becomes law through ratification. No proposal becomes law without passing through Governance. No law may be created directly in the Protocol layer.

This mapping is structural, not metaphorical. The system literally implements a legislative process where behavioral specifications are proposed, validated against constitutional rules, and promoted to authoritative status.

---

3. Artifact Lifecycle Model

3.1 Lifecycle Stages

Every artifact progresses through defined lifecycle stages. The lifecycle is not advisory; it is constitutionally enforced.

Let a be an artifact of concern C. At time t, artifact a occupies exactly one layer:

1. **Draft State** — Artifact resides in Authoring layer
2. **Validation State** — Artifact is under Governance examination
3. **Ratified State** — Artifact resides in Protocol layer

Formally:

$$\text{Layer}(a, t_0) = \text{Authoring}$$
$$\text{Layer}(a, t_1) = \text{Governance}$$
$$\text{Layer}(a, t_2) = \text{Protocol}$$

Where $t_0 < t_1 < t_2$.

Critical invariant: The concern of an artifact never changes during lifecycle progression:

$$\forall t : \text{Concern}(a, t) = C$$

A workflow draft (WF_) becomes a ratified workflow (WF_). It does not become an intent or a capability contract. Layer changes; concern remains invariant.

3.2 Lifecycle Invariants

The lifecycle model enforces three invariants:

I-G1 (Pre-Execution Validation) No artifact may enter Execution unless validated by Governance. There is no path from Authoring to Execution that bypasses Governance.

$$\text{Executable}(a) \Rightarrow \text{Validated}(a)$$

I-G2 (No Runtime Governance) Governance rules execute at authoring time and load time, not during runtime behavior. Once execution begins, no governance validation occurs. All artifacts are known-valid before execution starts.

$$\text{Executing}(t) \Rightarrow \neg\text{Validating}(t)$$

I-G3 (Irreversibility of Ratification) Once ratified, artifact content cannot be mutated in place. A ratified artifact is immutable. Behavioral change requires creation of a new artifact version.

$$\text{Ratified}(a_v) \Rightarrow \text{Immutable}(\text{Content}(a_v))$$

These invariants ensure that governance is structural, complete, and non-circumventable.

---

4. Schema Validation Mechanics

Governance operates through machine-verifiable validation. It is not a review process performed by humans examining documentation. It is an enforcement layer executed by validators.

4.1 Validation Components

Governance validation comprises five distinct mechanisms:

Component	Purpose
Structural schemas	Validate artifact shape and required fields
Vocabulary validation	Validate that identifiers use registered vocabulary

Component	Purpose
Prefix enforcement	Validate that artifact names use correct concern prefixes
Referential integrity	Validate that artifact references resolve to existing artifacts
Concern-layer consistency	Validate that artifacts appear in admissible layers for their concern

Each component is independently enforceable. An artifact must pass all five components to achieve ratification.

4.2 Formal Admissibility

Let V be the vocabulary of concerns (the set of registered concern prefixes). Let S be the set of structural schemas. Let $\mathcal{A}_{\text{valid}}$ be the set of valid artifacts. Let admissible_layers(C) return the layers where concern C may appear.

An artifact a is **admissible** if and only if:

$$\text{Prefix}(a) \in V$$

$$\text{SchemaValidate}(a, S) = \text{true}$$

$$\text{ReferencesResolve}(a) = \text{true}$$

$$\text{Layer}(a) \in \text{admissible\_layers}(\text{Concern}(a))$$

This definition connects directly to the Layer-Concern invariants established in Paper 3 [Bachi, 2026c].

4.3 Validation as Gate Function

Validation is a gate function. It returns a binary result: admissible or not admissible.

$$\text{Validate} : \text{Artifact} \to \{\text{admissible}, \text{rejected}\}$$

There is no partial admissibility. An artifact that fails any validation component is rejected. Governance does not negotiate or compromise. It enforces.

When an artifact is rejected, the validation system provides:

1. The specific validation component that failed
2. The rule that was violated
3. The location within the artifact where violation occurred

This feedback enables correction in the Authoring layer. The artifact may then be resubmitted for validation.

---

5. Versioning as Constitutional Guarantee

Versioning in protocol-governed systems is not packaging metadata. It is behavioral identity.

5.1 Version Immutability

For artifact a at version v, denoted a_v:

$$\text{Hash}(a_v) \text{ is immutable}$$

Any behavioral change requires a new version $v+1$. Version v remains unchanged, addressable, and available.

This is not a recommendation; it is a constitutional guarantee enforced by the Governance layer.

5.2 Versioning Invariants

I-G4 (Version Immutability) Artifact content at version v cannot be modified. Once a_v is ratified, its content is fixed for all time.

$$\text{Ratified}(a_v, t_0) \Rightarrow \forall t > t_0 : \text{Content}(a_v, t) = \text{Content}(a_v, t_0)$$

I-G5 (Version Addressability) All versions remain independently addressable. Version v and version $v + 1$ both exist and can be referenced.

$$\text{Exists}(a_v) \wedge \text{Exists}(a_{v+1}) \Rightarrow \text{Addressable}(a_v) \wedge \text{Addressable}(a_{v+1})$$

I-G6 (Coexistence) Multiple versions may coexist without override. Creating version $v + 1$ does not invalidate or remove version v.

$$\text{Create}(a_{v+1}) \Rightarrow \text{Exists}(a_v)$$

These invariants enable:

- **Behavioral auditability:** The system can demonstrate what behavior was authoritative at any point in time.
- **Rollback capability:** Previous versions remain available for explicit selection.
- **Parallel deployment:** Different runtime bindings may reference different versions of the same artifact.
- **Change attribution:** Each version records when it was ratified and by what authority.

---

6. Amendment Model

Governance must allow change without instability. Behavioral law evolves, but evolution must be governed.

6.1 Amendment Types

Two amendment types are constitutionally permitted:

Additive Amendment A new artifact version is introduced without invalidating prior versions. The prior version remains ratified, addressable, and available. Systems referencing the prior version continue to function unchanged.

$$\text{AddVersion}(a, v + 1) \Rightarrow \text{Valid}(a_v) \wedge \text{Valid}(a_{v+1})$$

Deprecation Amendment A prior version is marked deprecated. Deprecation is a metadata annotation; it does not mutate or remove the artifact. Deprecated artifacts remain addressable and may continue to be referenced by existing runtime bindings.

$$\text{Deprecate}(a_v) \Rightarrow \text{Exists}(a_v) \wedge \text{Addressable}(a_v) \wedge \text{DeprecationFlag}(a_v)$$

6.2 Prohibited Amendment Patterns

The following amendment patterns are constitutionally prohibited:

Pattern	Description	Why Prohibited
Silent override	Replacing artifact content without version increment	Violates I-G4
Retroactive mutation	Modifying content of ratified artifact	Violates I-G4
Implicit replacement	New version automatically supersedes prior	Violates I-G6
Untracked deletion	Removing artifact without deprecation record	Destroys audit trail

6.3 Amendment Principle

The governing principle of amendment is:

> **Behavioral law evolves by addition, not mutation.**

New behavior is introduced through new versions. Old behavior is never silently changed. Systems that reference old versions continue to receive old behavior. The choice to adopt new behavior is explicit and traced.

This principle enables governance at institutional timescales. Organizations can adopt new protocol versions through deliberate governance processes without fear that underlying artifacts will change beneath them.

---

7. Authority Model

Governance requires explicit authority assignments. The system must know who may propose artifacts and who may ratify them.

7.1 Authority Sets

Let A_L denote the authority set for layer L. Authority sets are defined per-layer:

Layer	Authority Set	Function
Authoring	$A_{\text{Authoring}}$	May create and modify drafts
Governance	$A_{\text{Governance}}$	May ratify or reject artifacts
Protocol	A_{Protocol}	May reference ratified artifacts

Only members of $A_{\text{Authoring}}$ may create drafts. Only members of $A_{\text{Governance}}$ may ratify.

Authority does not leak downward. Execution authorities cannot modify Protocol artifacts. Protocol artifacts cannot be modified after ratification.

7.2 Authority Invariants

I-G7 (Authority Containment) Authority to modify artifacts is confined to Authoring and Governance layers. Once an artifact enters the Protocol layer, modification authority terminates.

$$\text{Layer}(a) = \text{Protocol} \Rightarrow \neg\text{ModifyAuthority}(a)$$

I-G8 (No Execution-Level Law Creation) Execution may not create or alter ratified artifacts. The execution engine enforces law; it does not make law.

$$\text{Layer}(a) = \text{Execution} \Rightarrow \neg\text{CreateAuthority}(\text{Protocol\_Artifact})$$

These invariants ensure that behavioral law originates in human governance processes and cannot be altered by computational execution. Execution is constitutionally subordinate to Protocol.

---

8. Governance Failure Modes

Governance can fail. Understanding failure modes enables their prevention through architectural design.

8.1 Structural Failure Cases

Failure Mode	Description	Prevention Mechanism
Schema Drift	Schema definitions change incompatibly	Schema versioning with compatibility rules
Vocabulary Inflation	Uncontrolled addition of new concerns	Constitutional amendment requirement for new prefixes
Prefix Ambiguity	Multiple concerns share prefix patterns	I-C1 (Prefix Uniqueness) from Paper 3
Cross-Layer Leakage	Artifacts appear in inappropriate layers	Concern-layer consistency validation
Runtime Rule Injection	Governance rules evaluated during execution	I-G2 (No Runtime Governance)

8.2 Prevention Architecture

Each failure mode has a corresponding prevention mechanism built into the Governance layer:

Schema Drift Prevention Schemas are versioned artifacts subject to the same governance process as behavioral artifacts. Schema changes require: - New schema version creation - Impact analysis on existing artifacts - Migration path specification - Explicit adoption by downstream artifacts

Vocabulary Inflation Prevention The concern vocabulary is constitutionally bounded [Bachi, 2026c]. Adding a new concern requires: - Unique prefix assignment - Execution semantics specification - Trace attribution rules - Governance amendment ratification

This constitutional expense prevents casual vocabulary expansion.

Prefix Ambiguity Prevention Prefix uniqueness (I-C1) is enforced at artifact creation. The validation system rejects any artifact whose prefix: - Does not match a registered concern - Matches a registered concern but violates that concern's schema

Cross-Layer Leakage Prevention The Layer-Concern matrix [Bachi, 2026c] defines which concerns may appear in which layers. Validation enforces this matrix. An artifact that attempts to appear in an inadmissible layer is rejected.

Runtime Rule Injection Prevention The architectural separation between load-time validation and runtime execution is structural. The execution engine does not contain governance code. Validation occurs before execution begins; no validation path exists during execution.

---

9. Machine-Enforceable Governance

Governance in protocol-governed systems is not a human review process. It is machine-verifiable enforcement.

9.1 Enforcement Mechanisms

Governance enforcement operates through:

Mechanism	Function
Schema validators	Verify artifact structure against registered schemas
Prefix checkers	Verify artifact names use valid concern prefixes
Referential integrity validators	Verify all artifact references resolve
Layer routing rules	Verify artifacts appear in admissible layers
Hash validators	Verify ratified artifact content matches recorded hash

Each mechanism is deterministic. Given identical inputs, enforcement produces identical results. There is no human judgment in the enforcement path.

9.2 Enforcement Architecture

The governance enforcement pipeline processes artifacts in sequence:

```
Authored Artifact
    ↓
Prefix Validation
    ↓
Schema Validation
    ↓
Referential Integrity Check
    ↓
Layer-Concern Consistency Check
    ↓
Ratification (hash recording)
    ↓
Protocol Layer Promotion
```

Each stage is a gate. Failure at any stage terminates the pipeline with rejection. Success at all stages results in ratification.

9.3 Governance as Architecture

This enforcement model makes governance **architectural** rather than procedural:

- **Procedural governance:** Humans review artifacts and make judgment calls. Consistency depends on human attention. Enforcement gaps appear under time pressure.
- **Architectural governance:** Machines enforce rules deterministically. Consistency is guaranteed by implementation. Enforcement cannot be bypassed because no bypass path exists.

Protocol-governed systems implement architectural governance. Human judgment occurs in the Authoring layer when creating artifacts. Human authority occurs in defining governance rules. But enforcement is mechanical and complete.

---

10. Separation from Execution

This paper defines governance mechanics. It explicitly does not define execution mechanics.

10.1 Scope Boundaries

This paper does not address:

Topic	Addressed In
Runtime routing	Paper 6
DAG compilation	Paper 6
Trace semantics	Paper 6
Transform execution	Paper 7
Side-effect execution	Paper 7
Protocol artifact semantics	Paper 5

10.2 The Governance-Execution Boundary

The boundary between Governance and Execution is constitutional:

$$\text{Governance} : \text{Artifact} \to \text{Admissibility}$$

$$\text{Execution} : \text{Artifact} \times \text{Input} \to \text{Trace}$$

Governance determines whether an artifact may exist in the Protocol layer. Execution determines what happens when a ratified artifact is invoked.

These functions are independent. Governance does not depend on execution semantics. Execution does not invoke governance rules. Their separation is structural and deliberate.

---

11. Structural Consequences

The governance model yields five structural consequences:

11.1 Deterministic Admissibility

Artifact admissibility is deterministic. Given an artifact a and governance rules G:

$$\text{Admissible}(a, G) = \text{deterministic\_function}(a, G)$$

There is no ambiguity about whether an artifact is admissible. The validation system produces the same result every time it examines the same artifact.

11.2 Versioned Behavioral History

The version immutability guarantees (I-G4 through I-G6) create a complete behavioral history. At any point in time, the system can answer:

- What artifact versions were ratified?
- When was each version ratified?
- What authority ratified each version?
- Which versions are currently active?
- Which versions have been deprecated?

This history is structural, not reconstructed from logs. It exists as first-class governance data.

11.3 Explicit Amendment Trail

Every behavioral change is explicit. There is no silent mutation. The amendment trail shows:

- Each version that ever existed
- The relationship between versions
- Deprecation status and timing
- The authority that approved each change

11.4 Authority Traceability

Authority is explicit and recorded. For any ratified artifact, the system can identify:

- Who created the draft
- Who ratified the artifact
- What governance rules were applied
- When ratification occurred

Authority traceability enables compliance verification without manual audit.

11.5 Bounded Vocabulary Expansion

Vocabulary expansion is constitutionally expensive. New concerns, new layers, and new prefixes require explicit governance amendment. This expense is deliberate.

Unbounded vocabulary expansion leads to: - Ambiguous artifact classification - Ungoverned behavioral types - Schema proliferation - Enforcement gaps

Bounded expansion maintains: - Clear artifact classification - Complete behavioral coverage - Manageable schema set - Comprehensive enforcement

The constitutional expense of expansion is a feature, not a limitation.

---

12. Limitations

This paper:

- **Does not formalize operational semantics.** The execution behavior of artifacts is not specified. That topic belongs to Paper 6.
- **Does not address enforcement performance.** The computational cost of governance validation is not analyzed. Performance engineering is implementation-specific.
- **Does not quantify governance cost.** The economic overhead of governance is not measured. Economic analysis belongs to Paper 9.
- **Does not restate universality claims.** The computational completeness of protocol-governed systems was established in Paper 2 [Bachi, 2026b]. This paper does not revisit that argument.

Its scope is legislative mechanics only: how behavioral law is proposed, validated, ratified, versioned, and amended.

---

13. Conclusion

Protocol-governed systems require governance to be architectural rather than procedural.

By separating Authoring from Governance, the system creates a legislative process with distinct proposal and ratification phases. By enforcing schema validation mechanically, the system eliminates governance

gaps that arise from human attention limits. By requiring version immutability, the system ensures that behavioral law cannot be silently changed.

The eight invariants introduced in this paper (I-G1 through I-G8) constitute the constitutional law of governance:

Invariant	Statement
I-G1	No artifact may enter Execution unless validated by Governance
I-G2	Governance rules execute at authoring time and load time, not during runtime
I-G3	Once ratified, artifact content cannot be mutated in place
I-G4	Artifact content at version v cannot be modified
I-G5	All versions remain independently addressable
I-G6	Multiple versions may coexist without override
I-G7	Authority to modify artifacts is confined to Authoring and Governance layers
I-G8	Execution may not create or alter ratified artifacts

These invariants are not design guidelines. They are constitutional constraints enforced by the architecture.

The core insight is this:

> **Execution enforces law. Governance makes law admissible. Their separation is structural and necessary.**

When governance is procedural, it depends on human diligence and degrades under pressure. When governance is architectural, it is implemented in the system itself and cannot be bypassed. Protocol-governed systems choose architectural governance because behavioral law is too important to depend on human attention alone.

License and Use

Author Information

Bachi (aka Bhash Ganti) Contact: bachipeachy@gmail.com

Conflict of Interest: The author is developing commercial implementations of the described architecture.

References

Bachi aka Bhash Ganti (2026a). Protocol-Governed Systems: An architectural foundation for the AI era. *Zenodo Working Paper. DOI: https://doi.org/10.5281/zenodo.18715516*

Bachi aka Bhash Ganti (2026b). Protocol-Governed Systems: A constitutional realization of Turing-complete systems. *SZenodo Working Paper. DOI: https://doi.org/10.5281/zenodo.18718409*

Bachi aka Bhash Ganti (2026c). The Layer-Concern Constitutional Model: A formal structural taxonomy for protocol-governed systems. *SZenodo Working Paper. DOI: https://zenodo.org/doi/10.5281/zenodo.18719589*

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley.

Dijkstra, E.W. (1974). On the role of scientific thought. EWD447.

Lamport, L. (2002). *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley.

Meyer, B. (1988). *Object-Oriented Software Construction*. Prentice Hall.

Object Management Group (2011). Business process model and notation (BPMN) version 2.0. OMG Standard.

Parnas, D.L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058.

Saltzer, J.H., and Schroeder, M.D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308.

Spivey, J.M. (1989). *The Z Notation: A Reference Manual*. Prentice Hall.