INFRASTRUCTURE FOR THE AI ECONOMY

Axiograf:

A Protocol for Machine-Verifiable Enterprise Action in the Age of Autonomous Firms

Alexander Adani

March 2026

> *The central challenge of autonomous enterprise is not intelligence. It is legitimacy.*

DOI 10.5281/zenodo.19153625 | axiograf.com

ABSTRACT

This paper derives, from five primitive axioms concerning admissibility, a complete protocol architecture for machine-verifiable enterprise action. The derivation proceeds without empirical assumption, domain commitment, or temporal presupposition: from the logical structure of constraint alone.

The axiomatic foundation is Constraint-Induced Morphodynamics (CIMD). CIMD establishes that a system is constituted by its admissibility conditions; that constraints act solely by elimination, not by generation or preference; that non-invariant states cannot persist; and that stability is persistence under ordered re-evaluation, not under time. From these axioms a single derived statement follows necessarily: where admissibility conditions exist, non-invariant states cannot persist, admissible transitions therefore occur, and configurations that remain admissible across ordered evaluations constitute stable structure.

Axiograf is the operationalization of that derived statement as deployable enterprise infrastructure. Its architecture is not designed from observed practice — it is derived from the axioms. The three-layer structure of Canonical Fact Language, Adaptor Layer, and Constraint Layer follows necessarily from what the predicate C requires to be declared, evaluated, and verified. Constraints compile to stratified Datalog with bounded quantifiers — decidable, polynomial-time, and always explainable. A distributed stateless evaluator executes admissibility determinations at action time, producing cryptographically anchored Admissibility Records that serve as the complete evidentiary artifact of the protocol. A three-tier Constraint Registry with content-addressed versioning ensures every evaluation is permanently replayable. A cross-enterprise trust model enables independent verification without central authority and without prior agreement on policy content.

Together, CIMD and Axiograf constitute the missing admissibility substrate of the AI economy — the first infrastructure layer derived from first principles of admissibility rather than assembled from engineering convention.

1. INTRODUCTION

Something has shifted. Across every industry, enterprises are delegating consequential decisions — financial, logistical, commercial, operational — to autonomous AI agents executing in real time, at a scale and frequency no human organization has ever operated at before.

This is not a future scenario. It is happening now. And it is accelerating.

The world has invested enormous resources in making these agents smarter, faster, and more capable. But a more fundamental question has gone largely unasked — and entirely unanswered:

How can an enterprise prove that an autonomous action was permissible at the exact moment it was taken?

In human organizations, this question is managed — imperfectly, but manageably — through policy documents, approval hierarchies, and retrospective audits. These mechanisms rest on three assumptions: that action frequency is low enough for human checkpoints; that human judgment can be inserted at critical junctures; and that compliance can be reviewed after the fact.

Autonomous enterprises violate all three assumptions simultaneously. Agents act in milliseconds. There are no human checkpoints. And when an action has already propagated through a supply chain, triggered a contract, or moved capital, "after the fact" is too late.

Without a machine-verifiable admissibility layer, large-scale AI operation exposes organizations to silent policy drift, unverifiable decisions, cross-enterprise disputes, regulatory action, and — most dangerously — systemic trust erosion across the entire AI economy.

This paper proposes that orderly AI economies require a minimal, formal admissibility substrate. It introduces Axiograf as a protocolized implementation of that substrate — and makes the case that this infrastructure is not optional. It is the missing foundation upon which every autonomous enterprise will eventually need to be built.

2. THE VOID AT THE CENTER OF ENTERPRISE AI

2.1 The Limits of Current Enterprise Controls

Existing enterprise control mechanisms were designed for a world of human-speed, human-mediated operations. Role-based access control, policy engines, workflow approvals, and compliance audits share a common architecture: they assume a human in the loop who can read context, exercise judgment, and be held accountable.

In autonomous environments, this architecture fails along four structural dimensions:

- Fragmented evaluation. Admissibility is assessed across disconnected systems — ERP, legal, finance, compliance — with no unified determination at the moment of action.
- Incomplete context. Policy engines evaluate against static rule sets, not the full operational context available at execution time.
- Weak provenance. Decision trails are lossy. What the agent knew, what constraints were active, and what alternatives were considered are rarely captured in verifiable form.
- No guaranteed replayability. In the event of a dispute, regulators and counterparties cannot reconstruct the exact conditions under which an action was taken.

These limitations were tolerable when humans remained in the loop. They become existential failure modes when agents act continuously at scale.

2.2 From Decision Intelligence to Decision Legitimacy

The AI infrastructure industry has overwhelmingly focused on decision intelligence: making agents that optimize better, predict more accurately, and act more efficiently. This is valuable work. But in regulated commercial environments, it addresses the wrong question first.

> *The prior question is not: What is the optimal action? The prior question is: Is this action admissible under the governing constraints that exist right now?*

An action that is optimal but non-admissible is operationally invalid. It creates liability. It erodes counterparty trust. It invites regulatory intervention. In a world of autonomous enterprise, non-admissible actions will be taken at machine speed, and their consequences will propagate faster than any human response can contain them.

This is the void that Axiograf is designed to fill.

2.3 Why Existing Systems Cannot Fill It

A natural question arises: do not existing policy-as-code systems already solve this? Systems such as Open Policy Agent, AWS Cedar, and runtime verification frameworks are sophisticated and widely deployed. The answer is that they solve a different, narrower problem — and the distinction is architectural, not incremental.

Existing policy systems evaluate rules against system-specific data structures within a single organizational boundary. They do not define a universal fact language that makes constraints portable across enterprises. They do not generate cryptographically anchored Admissibility Records that a counterparty can independently verify. They do not enforce stability across action trajectories. And they provide no mechanism for cross-enterprise trust that requires no prior agreement between parties.

Axiograf is not a better policy engine. It is the admissibility substrate that policy engines, compliance systems, and autonomous agents all need to operate on — the layer below them, not a replacement for them. The forcing function for adoption is not philosophical preference. It is the convergence of three concrete pressures: the EU AI Act's requirements for documented oversight of high-risk AI systems; the SOC 2 Type II expectation that automated decisions be auditable and reproducible; and the emerging reality that enterprises whose autonomous agents can demonstrate admissibility will win contracts, lower insurance premiums, and satisfy regulators faster than those that cannot. Admissibility proof is becoming a commercial advantage before it becomes a legal requirement. The enterprises that build on this substrate now will not scramble to retrofit it later.

3. CONSTRAINT-INDUCED MORPHODYNAMICS: A FIRST-PRINCIPLES AXIOMATIZATION

Axiograf originates from first principles. It is not assembled from engineering convention or derived from observed compliance practice. It is derived from a minimal axiomatization of admissibility itself — five axioms that define what a constrained system is, what constraints do, what they necessitate, and what stability means. Every component of the Axiograf protocol follows from these axioms. Nothing is assumed beyond them.

The axiomatization is maximally pure: definition then necessity then consequence. No domain commitments. No temporal assumptions.

The Irreducible Form

Before the axioms are stated, the theory can be expressed in its irreducible form. Everything in CIMD — and every component of Axiograf — derives from three elements and one rule:

$$\Omega, \quad C, \quad S(t+1) \in C(S(t))$$

Where: Ω is the possibility space of all logically conceivable states; C is the constraint operator that maps Ω to P(Ω), the set of admissible states; $S(t+1) \in C(S(t))$ is the rule that every successor state must be a member of the admissible set defined by applying C to the current state.

> *Note on the constraint operator: C maps Ω to P(Ω) — the power set of admissible subsets — not to a specific next state. C returns which states are admissible; the agent or actor chooses within that admissible set. This is a foundational distinction from generative or optimization models: constraints do not produce outcomes. They define the space within which outcomes must fall. The rule $S(t+1) \in C(S(t))$ is a membership check, not a function evaluation. This is what Axiom 2 formalizes.*

The axioms that follow derive the properties of this irreducible form. They are not additional assumptions — they are the unpacking of what Ω, C, and the membership rule logically entail.

Axiom 0 — Systemhood

A system exists if and only if there exists a non-empty set of admissibility conditions such that not all possible states are permitted. Formally, a system is defined where a constraint set C renders an admissible state space $\Omega_v \subset \Omega$.

> *Remark: This axiom does not assert the origin of C. It establishes the domain of applicability of the theory. A system is not a thing that has constraints applied to it — it is constituted by the existence of constraints. Where no constraint set renders any state impermissible, there is no system in the CIMD sense. This has a direct consequence for Axiograf: an enterprise without declared admissibility conditions is not a system the protocol can operate on. Axiograf does not impose constraints. It operationalizes constraints that must already exist for the enterprise to be a governed system at all.*

Axiom 1 — Possibility

For any system, there exists a possibility space Ω consisting of all logically conceivable states prior to admissibility evaluation. Ω is a prerequisite for exclusion and carries no empirical or causal commitment.

> *Remark: Ω makes no claim about what actions or states are physically possible, economically feasible, or operationally likely. It contains everything that is logically conceivable. The constraints do the domain work. This separation is what makes CIMD domain-agnostic: the same formal structure applies to financial actions, supply chain decisions, clinical authorizations, and any other enterprise domain, because Ω makes no domain assumption.*

Axiom 2 — Constraint

A constraint set C acts solely by eliminating inadmissible states from Ω, yielding Ω_v. Constraints do not generate states. They do not impose preference. They do not imply direction. They define admissibility only.

> *Remark: The three negatives are load-bearing. Constraints do not generate states — they cannot make new actions available, only render existing ones impermissible. They do not impose preference — there is no ranking, weighting, or utility within Ω_v. They do not imply direction — admissibility is not a gradient toward a goal. This is precisely what distinguishes CIMD from optimization frameworks, utility theory, and goal-directed planning. It is also why binary admissibility is not a limitation but a logical consequence: a predicate that does not impose preference cannot produce a gradient.*

Axiom 3 — Dynamics

If a state $\omega \in \Omega_v$ is not invariant under the full constraint set C, then ω cannot persist unchanged. Persistence therefore requires either transition to another admissible state, or elimination from Ω_v. Dynamics are defined as admissibility-preserving transitions necessitated by non-invariance.

> *Remark: Dynamics in CIMD are not caused by forces, incentives, or goals. They are necessitated by non-invariance under constraint. A state that cannot remain admissible must change. This is not a design choice of the protocol — it is a logical consequence of the axioms. The violation-response coupling in Axiograf — the rule that any non-admissible determination must block or redirect the action — is the operational expression of this axiom.*

Axiom 4 — Stability

A state or transition pattern is stable if it remains admissible under an ordered sequence of admissibility evaluations. The ordering of evaluations is logical, not temporal, and requires no assumption of discreteness or continuity. Stability is persistence under repeated admissibility.

> *Remark: The logical rather than temporal ordering of evaluations is the axiom that makes CIMD applicable to both real-time agent action sequences and to logical state progressions without modification. Time is never assumed. Succession refers to ordered re-evaluation of admissibility, not to duration or physical process. This is why bounded-horizon stability in Axiograf — evaluated over a declared finite sequence of N actions — is not a temporal claim but a logical one: it asks whether the declared sequence remains admissible under repeated evaluation.*

The Derived Statement

> ***Where admissibility conditions exist, non-invariant states cannot persist; admissible transitions therefore occur; and configurations that remain admissible across ordered evaluations constitute stable structure.***

This derived statement is the logical consequence of the five axioms taken together. It is not an empirical claim. It is not an engineering proposition. It is what follows necessarily from the definitions. Axiograf is the operationalization of this derived statement: the infrastructure that makes it possible for real enterprises, with real constraints, operating in real time, to instantiate the stable structure that the derived statement describes.

On the Origin of Constraints

CIMD does not posit a source of constraints. This is not an omission — it is a deliberate feature of the axiomatization. Constraints are not internal to the agent, external impositions from an environment, or emergent properties of a system. They are preconditions for system definition. A constraint set C must exist for there to be a system in the CIMD sense at all. The framework specifies the consequences of constraint existence, not its origins.

For Axiograf, this means the protocol is neutral on whether the constraints governing an autonomous enterprise are regulatory (imposed by law), contractual (agreed with counterparties), ethical (self-declared by the enterprise), or operational (derived from system capabilities). All that is required is that they be declarable and evaluable. The source is the enterprise's concern. The consequences are the protocol's domain.

Computational Characterization

Constraints in Axiograf compile to stratified Datalog with bounded quantifiers. This characterization is a direct consequence of Axiom 2: because constraints act solely by elimination and impose no preference or direction, the constraint language needs only the expressive power of a classification predicate over typed facts. Stratified Datalog is precisely such a language, with well-established properties that matter directly to enterprise deployability:

- Decidability. Every constraint evaluation terminates. There are no infinite loops, no undecidable halting conditions, no pathological cases arising from computational generality.
- Polynomial-time evaluation. Evaluation complexity is $O(k \cdot |F|)$ where k is the number of constraint rules and $|F|$ is the size of the fact set. Both are bounded by protocol design.
- No Turing-completeness. Constraints cannot express arbitrary computation. They reason only over typed facts declared in the Canonical Fact Language schema. This bound is enforced at constraint authoring time.
- Composability. Stratified Datalog constraints compose without loss of decidability or complexity properties.

This places Axiograf's constraint evaluation in a known, extensively studied computational class. Implementers have well-understood tooling. Auditors have a precise specification. Regulators get a constraint language that is, by construction, always explainable.

3A. FROM AXIOMS TO ARCHITECTURE: THE DERIVATION

The architecture of Axiograf is not a design. It is a derivation. Each component of the protocol answers a question that the axioms make unavoidable. The chain runs as follows.

Axiom 0 establishes that a system requires a constraint set C. C must therefore exist somewhere, be owned by someone, and be evaluable against proposed actions. This immediately requires a representation of C that is machine-readable, versioned, and cryptographically bound — the Constraint Object of the Axiograf protocol.

Axiom 1 establishes that possible states exist prior to admissibility evaluation. Those states must be representable in a form that C can reason over. This requires a canonical representation of proposed actions as typed facts — the Canonical Fact Language. The language must be universal: if different enterprises use different fact representations, the predicate C cannot be evaluated consistently across enterprise boundaries.

Axiom 2 establishes that C acts solely by elimination. The evaluation of C against a proposed action must therefore be a pure classification: admissible or not. This requires an evaluator that is itself free of preference, direction, and context — the distributed stateless evaluator, which receives only facts and predicate and returns only a binary determination.

Axiom 3 establishes that non-admissible states cannot persist. The protocol must therefore enforce this structurally — not through alerts or flags that humans may or may not act on, but through architectural prevention. The Admissibility Record is not a log; it is a gate. A non-admissible determination blocks the action before execution. This is the violation-response coupling as protocol enforcement.

Axiom 4 establishes that stability is persistence under ordered re-evaluation. The protocol must therefore be capable of evaluating not just individual actions but declared trajectories — sequences of N actions evaluated for sustained admissibility. This requires bounded-horizon stability evaluation, which in turn requires the agent to assemble a complete trajectory envelope before submission, and the evaluator to assess the full sequence as a single atomic computation.

Axiograf has no arbitrary design choices. Every component exists because an axiom requires it. Every interface is defined because the derivation chain demands it.

The three-layer architecture — Canonical Fact Language, Adaptor Layer, Constraint Layer — is the minimal engineering consequence of the derived statement. The Canonical Fact Language is what makes Ω representable to C. The Adaptor Layer is what translates enterprise reality into that representation. The Constraint Layer is where C lives as a versioned, evaluable, composable predicate. The derived statement says that admissible transitions occur and stable structures emerge where these conditions are met. The protocol is the infrastructure that instantiates those conditions in real enterprise systems.

4. THE THREE-LAYER ARCHITECTURE

The translation from CIMD's formal semantics to a deployable enterprise protocol requires a precise architectural answer to a deceptively simple question: what does the evaluator actually receive?

The answer defines the entire protocol. Axiograf's architecture is built on three separable, composable layers. Each layer has a defined role, a defined interface, and a defined governance model. The separation is not incidental — it is the source of the protocol's universality.

> **Layer 1** **The Canonical Fact Language** — *universal, protocol-defined, typed*

> **Layer 2** **The Adaptor Layer** — *enterprise-specific, implementer-defined, extensible*

> **Layer 3** **The Constraint Layer** — *policy-specific, declarant-defined, composable*

4.1 Layer 1: The Canonical Fact Language

The Canonical Fact Language is the universal substrate of the protocol. It is a typed, bounded schema that defines the complete vocabulary of facts that constraints may reference. It does not define what those facts mean in any particular domain. It defines what kinds of facts can exist: agent identity, resource type, quantity, jurisdiction, timestamp, prior action reference, counterparty identity, and so on.

The fact language is defined by the Axiograf protocol specification and evolves through a governed protocol process. Enterprises cannot customize it unilaterally. This is not a restriction — it is what makes cross-enterprise verifiability possible. Two enterprises can verify each other's admissibility determinations because they share the same fact language. The constraint logic may differ entirely. The vocabulary of facts it operates on does not.

The fact language is intentionally not Turing-complete. Constraints may only reference fact types declared in the schema. This bound is what guarantees that the evaluator's admissibility determination is always decidable in finite time. Expressiveness is achieved through richness of the fact schema, not through computational generality.

Table 1. Canonical Fact Language — Core Schema Specimen (v0.1)

Ten core fields sufficient to evaluate admissibility across payment, vendor, and regulatory constraint domains. The full schema extends this nucleus; all extensions are governed additions, backward-compatible, and version-stamped.

Field Name	Type	Constraints	Description
agent_id	UUID	required, immutable	Cryptographic identity of the acting agent
action_type	Enum	required, versioned	Declared category of the proposed action
resource_type	Enum	required	Class of resource the action operates on
quantity	Decimal	required, non-negative	Magnitude of the action (units defined by resource_type)
jurisdiction	ISO 3166	required	Legal jurisdiction governing the action
counterparty_id	UUID	optional	Identity of the receiving enterprise or agent
timestamp_utc	RFC 3339	required, monotonic	Moment of action proposal, tamper-evident
prior_action_ref	UUID	optional	Reference to preceding action in trajectory
horizon_n	Integer	required, ≥ 1	Declared stability horizon for trajectory evaluation
schema_version	SemVer	required	Canonical Fact Language version used to construct this envelope

The schema above is not illustrative. It is the nucleus of the first protocol version — ten fields sufficient to evaluate admissibility across payment authorization, vendor onboarding, and cross-border regulatory constraint domains. The schema is versioned using semantic versioning. A constraint authored against schema v0.1 is permanently bound to that version. When the schema evolves, existing constraints remain valid against their declared version. New constraints may adopt new schema versions. Backward compatibility is a governance requirement, not an implementation convenience.

4.2 Layer 2: The Adaptor Layer

The Adaptor Layer is where Axiograf touches existing enterprise systems. An adaptor is a translation component that converts the output of an enterprise system — an ERP record, a payment authorization, a contract event, a sensor reading — into a valid Action Envelope expressed in the Canonical Fact Language.

Adaptors are implemented by enterprises, system integrators, and third-party developers. They are not part of the core protocol. The protocol defines the interface: an adaptor must produce a well-typed Action Envelope conformant with the declared schema version. How it does so — which enterprise systems it reads, which transformations it applies, which context it assembles — is the concern of the implementer, not the protocol.

This design is deliberate. It means the protocol's adoption surface is the adaptor ecosystem, not the core specification. An enterprise that wants to bring a new system into the Axiograf framework does not need to modify the protocol. It needs to build or acquire an adaptor. The constraint logic remains unchanged. The fact language remains unchanged. Only the translation layer is new. This is precisely how TCP/IP achieved universality: the protocol defined the packet; the network interface card was the adaptor; neither needed to know about the other's implementation.

Adaptor development is a non-trivial engineering investment. Unlike network interface cards, enterprise adaptors require deep domain knowledge of the source system — its data models, its event semantics, its edge cases. Early adoption will concentrate where adaptor complexity is lowest and the admissibility problem is most acute: payment authorization, vendor onboarding, and cross-border regulatory compliance. As the adaptor ecosystem matures, coverage will extend to legacy ERP systems, clinical data platforms, and other domains where the integration surface is larger but the legitimacy stakes are equally high.

A precise distinction must be drawn between the admissibility of an action and the truth of the facts the action is based on. The Adaptor Layer is responsible for producing a well-typed, timestamped fact set from the enterprise system's output at the moment of envelope construction. If a fact is disputed — the price the adaptor reported was incorrect, the inventory level was stale — the dispute concerns the adaptor's output, not the evaluator's determination. The Admissibility Record proves what facts were submitted and what determination was made against those facts. It does not certify the independent truth of the facts themselves. This is architecturally correct: the protocol establishes legitimacy of process, not omniscience of data. Disputes about underlying facts are resolved through the enterprise's existing data governance mechanisms; Axiograf provides the evidentiary record of what the agent knew and what determination was made on that basis.

4.3 Layer 3: The Constraint Layer

Constraints are logical predicates written against the Canonical Fact Language and compiled to stratified Datalog. Because all constraints share the same fact substrate, they are composable: a constraint governing financial admissibility can be evaluated alongside a constraint governing regulatory compliance without either knowing the other exists. The evaluator applies all declared constraints; any non-admissible determination blocks the action.

When multiple constraints govern a single action, their combined effect is expressed precisely:

$$C = \bigcap C_i \quad (i = 1 \ldots n)$$

The valid state space is the intersection of all active constraints. Each C_i is a separately registered, separately versioned Constraint Object. An action is admissible only if it falls within every constraint's admissible set simultaneously. This is the formal expression of the strict-with-escalation rule: no majority weighting, no preference ranking — full intersection only.

Domain specialization in Axiograf lives in the Constraint Objects, not in evaluator infrastructure. The evaluator is domain-blind by design. What varies by domain is the constraint predicate it receives. Constraint Objects may be authored for specific domains — financial admissibility, regulatory compliance, operational capacity, security policy. A single evaluator instance applies all declared Constraint Objects to a given Action Envelope. This preserves the evaluator's policy-blindness while fully supporting the domain-structured constraint reality of real enterprises.

Constraints are versioned, cryptographically hashed, and bound to an effective time interval. A constraint that was active at the moment of an action is permanently associated with the Admissibility Record generated for that action. This is what makes historical replay possible: given the Action

Envelope and the constraint version hash, any party can re-execute the evaluation and verify the result.

Constraints may be authored by enterprises, by industry bodies establishing sector-wide standards, or by regulatory authorities publishing machine-readable compliance requirements. Because they are expressed in the Canonical Fact Language, a constraint authored by a regulator is directly evaluable by any enterprise whose adaptors produce conformant Action Envelopes. Regulatory compliance becomes a first-class protocol operation, not an interpretive exercise.

Constraint authoring is a necessary ecosystem layer that sits above the protocol but is not part of it. Translating regulatory text — GDPR data subject rights, Dodd-Frank position limits, EU AI Act oversight requirements — into Datalog predicates requires both legal domain expertise and formal methods competence. The protocol defines what a valid Constraint Object must be; it does not define how to author one. This separation is architecturally correct but practically consequential: the ecosystem requires constraint authoring tooling, including compilers from structured regulatory text to Datalog predicates, visual constraint builders for compliance teams without formal methods backgrounds, and libraries of pre-authored constraints published by regulatory authorities and industry bodies. These are not protocol concerns. They are the necessary infrastructure above the protocol layer that will determine the breadth of adoption.

4.4 The Constraint Registry

Every Constraint Object in the Axiograf ecosystem has a canonical home: the Constraint Registry. The Registry is a content-addressed, immutable, three-tier repository in which every Constraint Object is permanently resolvable by its version hash. Any evaluator, any enterprise, any regulator can resolve a constraint version hash to its exact, unmodified definition at any point in time.

The Registry operates in three tiers, each with a distinct governance model that mirrors the namespace architecture of the Canonical Fact Language. The Core Registry, governed by the Axiograf Foundation, holds the canonical schema versions and any cross-industry base constraints. It evolves deliberately, with a high bar for inclusion and a governance cadence measured in years. Domain Registries, governed by sector bodies, hold domain-specific constraints: financial constraints registered by financial standards bodies, healthcare constraints by health interoperability authorities. Each domain registry federates through the Foundation's namespace registry. Enterprise Registries, governed by individual enterprises, hold private constraints that apply only within that enterprise's evaluation context. An enterprise may contribute a private constraint to a domain registry if it chooses; otherwise it remains local.

Constraint versioning is content-addressed. A constraint version hash is the hash of the Constraint Object's content — like a git commit hash. The same hash always returns the same Object. That Object is immutable after publication. If a constraint changes, a new hash is generated and a new Registry entry is created. Historical Admissibility Records permanently reference the hash that was active at the time of evaluation. The Registry never deletes. Replay is always possible.

4.5 Constraint Discovery: The Manifest

The evaluator does not search for the right Constraint Object. It is told which one to use. When an enterprise configures its evaluator deployment, it declares a Constraint Manifest — a cryptographically signed document mapping each action_type to the set of Constraint Object hashes that govern it. The Manifest itself is versioned and signed with the enterprise's private key, with the

corresponding public key registered in a known location so that counterparties can verify that the Manifest they are relying on was genuinely declared by the enterprise and not modified.

When an agent submits an Action Envelope, the envelope includes the action_type field. The evaluator looks up the Constraint Manifest for that action_type and resolves the declared constraint hashes against the Registry. Resolution is content-addressed: the hash is the address. The evaluator sends the hash to the Registry and receives exactly one immutable Constraint Object in return. The same hash always returns the same Object. No search. No inference. No ambiguity.

The complete resolution chain — Manifest version, constraint hashes, Registry response — is recorded in the Admissibility Record. This makes the governance decision that selected the constraints as auditable as the evaluation itself. A regulator or counterparty examining a historical Admissibility Record can verify not only what constraints were applied, but that those constraints were the ones the enterprise had formally declared for that action type at that moment.

5. THE DISTRIBUTED STATELESS EVALUATOR

The evaluator is the computational heart of the Axiograf protocol. Its design is governed by a single principle: it must be trustworthy without requiring trust in any party that operates it.

The protocol flow through the evaluator follows a precise sequence. Every action must traverse this sequence in full. No action reaches the execution layer without a confirmed Admissibility Record:

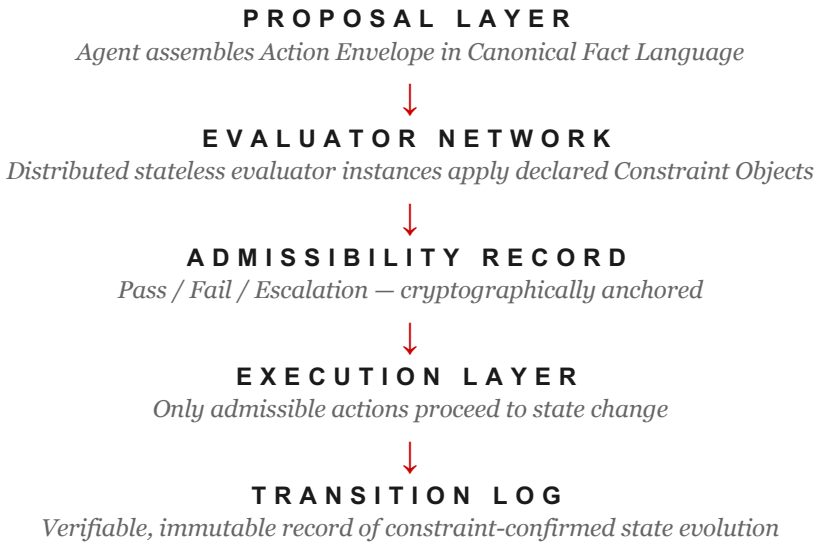

Figure 1. The Axiograf Protocol Flow — every action traverses all five layers in sequence.

This architecture transforms enterprise action from a two-step sequence — proposal then state change — into a four-step sequence: proposal, evaluation, admissibility confirmation, then state

change. The transformation is not cosmetic. It is the structural instantiation of Axiom 3: non-admissible states cannot persist, and the architecture enforces this before execution, not after.

This is achieved through a combination of statelessness, policy-blindness, determinism, and a performance envelope designed for enterprise-scale autonomous operation.

5.1 Stateless and Policy-Blind

The evaluator receives exactly two inputs: an Action Envelope and a Constraint Object. It produces exactly one output: an Admissibility Record containing a pass, fail, or escalation determination. It retains no state between evaluations. It has no knowledge of what the constraint means, what enterprise authored it, or what operational context surrounds the action. It only determines whether the facts in the Envelope satisfy the predicate in the Constraint.

This policy-blindness is not a limitation. It is the source of the evaluator's trustworthiness. An evaluator that understood policy context could be manipulated through that context. An evaluator that only processes typed facts against a logical predicate cannot be. Its output is fully determined by its inputs. Given the same inputs, any evaluator instance — operated by any party, on any infrastructure — produces the same output.

5.2 Distributed Without Bottleneck

Because the evaluator is stateless and deterministic, it can be deployed as distributed infrastructure without consensus requirements. Multiple evaluator instances can process different actions simultaneously. No evaluator instance holds privileged state. No central evaluator is required. The protocol requires only that evaluators be conformant implementations of the Axiograf specification — a property that can be verified independently, by anyone, against the canonical reference implementation.

This eliminates the bottleneck concern that afflicts centralized compliance architectures. High-frequency autonomous agents can operate at machine speed because admissibility evaluation is a distributed, parallelizable computation, not a serialized approval queue.

5.3 Logical Statelessness: A Precise Definition

The statelessness claim warrants precise treatment because the protocol contains fields — notably prior_action_ref and horizon_n — that appear to imply trajectory reasoning and therefore implicit state. The clarification is architectural and consequential.

prior_action_ref is a fact in the envelope, not a pointer the evaluator resolves. The evaluator does not fetch, look up, or reconstruct prior actions. It receives the prior action reference as a typed UUID field and evaluates the constraint predicate against it as a value — exactly as it evaluates jurisdiction or quantity. If a constraint requires reasoning over a multi-action trajectory, the submitting agent assembles the complete trajectory as a fact set within the envelope before submission. The evaluator receives the full trajectory as a self-contained input and evaluates it in a single, atomic computation. No external calls are made. No prior evaluation results are consulted.

This is the critical design choice that preserves statelessness under trajectory reasoning: statefulness is the agent's responsibility, not the evaluator's. The agent tracks trajectory history, assembles the complete fact set for horizon N, and submits it as a single envelope. The evaluator's computation is

always a pure function of its two inputs. The same envelope submitted to any evaluator instance — now or in ten years — produces the same Admissibility Record.

A precise distinction must be drawn between logical statelessness and implementation purity. An evaluator implementation may cache immutable objects — constraint versions, schema definitions — to avoid redundant fetches. This is an implementation optimization, not a protocol violation. The evaluator's logical behavior remains stateless: cached objects are immutable, versioned, and identical across all instances. Caching an immutable constraint version does not introduce evaluation history into the computation. It accelerates access to a fixed input. The protocol specifies logical behavior; deployment architecture governs implementation optimization. These are separate concerns and must be kept so.

The presence of timestamp_utc in the Canonical Fact Language schema warrants explicit treatment. This field does not reintroduce temporal assumptions into the axiomatization. timestamp_utc is a fact in the envelope — a typed value that constraint predicates may reason over, exactly as they reason over jurisdiction or quantity. A constraint that requires actions in a given jurisdiction to carry a timestamp within a defined window uses timestamp_utc as a data input to the predicate, not as a sequencing mechanism for the evaluator. The evaluator's ordering of admissibility evaluations remains logical, not temporal. Axiom 4's logical-time stance is fully preserved: the timestamp is a fact the constraint reasons about; it is not the mechanism by which evaluations are ordered. This distinction is what allows the protocol to be simultaneously domain-agnostic at the axiom level and temporally aware at the constraint level.

5.4 Performance Envelope

Axiograf's architectural choices collectively define a performance envelope that is compatible with enterprise-scale autonomous operation. Because constraint evaluation compiles to stratified Datalog, evaluation time scales polynomially with the size of the fact set and the number of constraint rules — both of which are bounded by protocol design. The Action Envelope has a maximum size determined by the Canonical Fact Language schema; it is not an open-ended data structure. Cryptographic anchoring of the Admissibility Record adds fixed overhead per evaluation, independent of the complexity of the constraint or the size of the action. Because evaluators are stateless, horizontal scaling requires no coordination: additional evaluator instances can be added without synchronization overhead or shared state management.

The practical implication is that admissibility evaluation is not a latency bottleneck in autonomous enterprise systems. It is a fixed-cost operation that can be scaled horizontally to match agent throughput. Streaming and batching strategies are implementer concerns, not protocol constraints: the protocol defines what must be evaluated and what the result must contain; the deployment architecture determines how evaluations are scheduled and parallelized. Performance benchmarks for specific deployment configurations will emerge as reference implementations are built and tested against real enterprise workloads.

5.5 Conflict Resolution: Strict with Escalation

When an action is evaluated against multiple constraints simultaneously, conflicts may arise: one constraint returns admissible, another returns non-admissible for the same action. The protocol's resolution mechanism is strict with escalation.

Strict means: any non-admissible determination from any constraint blocks the action. There is no weighting, averaging, or majority rule. A single non-admissible determination is sufficient to prevent

execution. This conservatism is correct for an admissibility substrate: it is always safer to escalate a legitimate action than to execute a non-admissible one.

Escalation means: when constraints conflict — when one returns admissible and another returns non-admissible for the same action — the system generates an escalation record rather than an outright rejection. The escalation record captures the full evaluation context, the conflicting determinations, and the constraint versions involved. It is routed to a designated resolution authority, which may be a human decision-maker, a higher-authority constraint, or an automated triage system. The escalation triage system is a designable layer above the protocol, not part of the core specification. This keeps the protocol simple and deterministic while acknowledging the genuine policy complexity of real enterprises.

5.6 System Stability: A Formal Definition

CIMD provides a precise definition of what it means for an enterprise system to be stable. A stable state S^* is one that satisfies its own constraints:

$$S^* \in C(S^*)$$

A stable enterprise state is one that remains admissible under its own governing constraints. This is a constraint equilibrium: the system is not being driven toward a goal, it is not optimizing toward a maximum, it is persisting within a defined admissibility boundary. For Axiograf, this means that a system operating correctly — where every action passes through the evaluation sequence and every Admissibility Record is positive — is by definition in a stable state. Stability is not an emergent property to be measured after the fact. It is a verifiable property of the protocol's execution.

5.7 The Admissibility Record as Complete Evidentiary Artifact

The Admissibility Record constitutes the complete evidentiary artifact of the protocol: it binds the proposed action, the active constraint version, the evaluation result, the agent identity, and the evaluation timestamp under a single cryptographic signature, providing the chain of custody required for audit, dispute resolution, and regulatory replay without requiring a separate proof system.

This is the key design choice that makes the protocol legally and operationally sufficient without additional proof infrastructure. A Transition Proof in the theorem-proving sense would require a proof system, a verifier, and a formal language for expressing proofs — significant additional infrastructure that is neither necessary nor appropriate at the protocol layer. The cryptographic attestation the Admissibility Record already provides is stronger in practice for legal and regulatory purposes: courts and regulators understand chain of custody and cryptographic integrity. The Record gives them exactly that, completely, in a single artifact.

6. CROSS-ENTERPRISE TRUST WITHOUT CENTRAL AUTHORITY

The most architecturally significant property of Axiograf is its model for cross-enterprise trust. When two autonomous enterprises interact — executing contracts, exchanging payments, onboarding

vendors, settling trades — each party needs assurance that the other's autonomous actions were legitimate at the moment they occurred. Current systems have no mechanism for this. Legitimacy is assumed, not verified.

Axiograf enables a new kind of commercial relationship: one in which legitimacy is verified, not assumed — and verified without requiring trust in any counterparty.

The protocol achieves this through a model directly analogous to the one that solved the double-spend problem in distributed currency systems. The insight is the same: you do not need a central authority to establish trust if you have a shared protocol that each party can independently verify.

6.1 How Cross-Enterprise Verification Works

Enterprise A's autonomous agent proposes an action. The adaptor produces an Action Envelope in the Canonical Fact Language. The evaluator applies Enterprise A's declared constraints and generates a signed, cryptographically anchored Admissibility Record. This record does not assert that the action is admissible under Enterprise B's constraints. It asserts only that the action was evaluated by a conformant Axiograf evaluator against a specific, versioned constraint set, and that the result was recorded with cryptographic integrity.

Enterprise B, upon receiving this record as part of a commercial interaction, performs its own independent evaluation. It takes the same Action Envelope — expressed in the shared Canonical Fact Language — and evaluates it against its own constraints using its own evaluator instance. If Enterprise B's constraints are satisfied, and the cryptographic integrity of Enterprise A's record is verified, the interaction proceeds.

Neither enterprise trusts the other's policy judgment. Both trust the shared protocol. The Canonical Fact Language is the common substrate that makes independent verification possible. Without it, Enterprise B would be evaluating Enterprise A's action against its own constraints using Enterprise A's data structures — which it cannot trust. With it, both enterprises are operating on the same typed fact representation, produced by a conformant adaptor, evaluated by a conformant evaluator.

A note on the scope of the "no prior agreement" claim: what Axiograf eliminates is prior agreement on policy content — neither party needs to know or accept the other's constraint set. What the protocol does require is structural alignment: both parties must agree on which namespace and domain registry governs a given action type. This is an agreement about protocol version and namespace scope, not about the substantive rules each enterprise applies. It is analogous to agreeing to use the same version of TCP/IP — a structural prerequisite, not a policy constraint. The protocol's claim is precisely and only this: no prior agreement on policy content is required for cross-enterprise legitimacy verification.

6.2 Protocol-Level Integrity, Not Counterparty Trust

The cryptographic anchoring of Admissibility Records serves a specific function: it certifies that an evaluation was performed by a legitimate Axiograf evaluator at a specific point in time against a specific constraint version. It is not a certification that the action was wise, optimal, or aligned with

the counterparty's interests. It is a certification that the action was evaluated through a legitimate admissibility process.

Regulators operate at a third level. Given the Action Envelope, the constraint version hash, and the Admissibility Record, a regulatory body can independently re-execute any historical evaluation exactly as it occurred. The constraint version that was active at the moment of the action is permanently bound to the record. Regulatory replay is not an interpretation exercise. It is a deterministic recomputation.

7. SECURITY AND INTEGRITY

A protocol for enterprise admissibility is only as valuable as its resistance to manipulation. Axiograf's security model addresses four primary threat vectors.

- Constraint tampering. Version hashing and cryptographic signature chains prevent silent modification of constraint definitions. Any change to a Constraint Object produces a new version hash. Historical Admissibility Records remain permanently bound to the constraint version under which they were generated. A tampered constraint cannot be retroactively applied to historical evaluations.
- Provenance forgery. Cryptographic attestations bind the Action Envelope, the evaluator identity, and the constraint version to the Admissibility Record. The Provenance Graph cannot be retroactively constructed or modified without invalidating its cryptographic integrity.
- Replay attacks. Temporal binding of constraint versions to Admissibility Records prevents retroactive admissibility manipulation. An action cannot be declared admissible under a constraint version that did not exist at the time the action was taken.
- Evaluator compromise. Because the evaluator is stateless and deterministic, a compromised evaluator instance can be detected by re-running any evaluation against a known-good instance. The outputs must be identical for any conformant implementation. Divergence is evidence of tampering.

8. IMPLEMENTATION PATH

Axiograf is designed for incremental adoption. Enterprises do not need to transform their entire operational stack to begin capturing the protocol's value. The implementation path follows a natural progression from internal governance to cross-enterprise infrastructure.

- Stage 1 — Internal admissibility logging. Enterprises implement Axiograf for internal audit and governance. Adaptors are built for high-risk systems. Admissibility Records and Provenance Graphs are generated and retained for key autonomous actions. This is not a replication of existing audit trails: existing audit trails are retrospective, lossy, and non-replayable. Axiograf Stage 1 Records are prospective — generated at the moment of action,

before execution — complete, and deterministically replayable by any party with the constraint version hash. The evidentiary quality is categorically different.

- Stage 2 — Cross-system verification. Admissibility is evaluated across internal systems and business units using the shared Canonical Fact Language. Constraint conflicts are surfaced and resolved through the escalation triage layer.
- Stage 3 — Cross-enterprise exchange. Counterparties exchange and independently verify Admissibility Records as part of commercial interactions. The Canonical Fact Language enables verification without prior policy agreement.
- Stage 4 — Regulator-grade reporting. The full protocol stack supports regulatory reporting and dispute resolution, with constraint-version-accurate replay capability available to any authorized party.

Early adoption will concentrate in high-risk, high-frequency domains: autonomous payments and settlement, AI-driven vendor onboarding and contracting, regulated commerce in financial services and healthcare, and cross-border supply chain operations.

> *Every enterprise deploying autonomous agents today is incurring admissibility debt — a growing liability that will need to be settled, one way or another, as regulation catches up with capability. The enterprises that build on this substrate now will not scramble to retrofit it later.*

9. GOVERNANCE AND THE OPEN PROTOCOL PRINCIPLE

Axiograf is designed to be universal infrastructure. Universal infrastructure succeeds only when it belongs, in a meaningful sense, to everyone who depends on it.

The protocols that became the rails of the modern economy — TCP/IP, SMTP, HTTP — share a common governance property: their initial conditions were set by those who understood the problem most precisely, and then opened to the world. No subsequent fork could displace them because they were right at the foundation level, and because being right at the foundation level is the only form of primacy that compounds rather than erodes.

Axiograf is therefore proposed as an open protocol, governed by an independent foundation with open membership and transparent specification processes. The Axiograf Foundation will maintain the canonical specification, govern the evolution of the Canonical Fact Language, certify conformant implementations, and steward the protocol's development in response to technical change, regulatory evolution, and the accumulated experience of adopting enterprises and industries.

The Canonical Fact Language is the most governance-sensitive component of the protocol. Its evolution determines what can and cannot be expressed in constraints. Changes to it affect every constraint and every adaptor in the ecosystem. The Foundation's primary governance responsibility is to evolve the fact language with the deliberateness its centrality demands — expanding it to cover new action domains while maintaining backward compatibility and formal tractability.

Schema governance is historically contentious. The evolution of shared vocabularies — XML Schema, ontology alignment, financial data standards — has repeatedly produced fragmentation, competitive namespace proliferation, and governance capture by dominant parties. The Axiograf Foundation's structural response to this risk is the namespace extension architecture: the core schema evolves slowly and conservatively under Foundation governance; domain-specific extensions are governed by sector bodies under their own cadences; enterprise-specific extensions remain private unless contributed upstream. This architecture does not eliminate governance tension — no architecture can — but it localizes it. Disputes about financial constraint semantics are resolved in the financial domain registry, not in Foundation governance. The Foundation's governance surface remains minimal and therefore more defensible against capture.

Open governance does not mean absence of authorship. The initial conditions of a protocol — its foundational design decisions, its core semantics, its minimum viable implementation — shape everything that follows. Those who establish those initial conditions in a spirit of genuine service to the ecosystem earn a form of primacy that no fork can fully displace. The goal of Axiograf is not to capture value from the AI economy. It is to ensure that the AI economy has the legitimate foundation it needs to function.

10. CONCLUSION

This paper began with five axioms. It ends with a protocol. The path between them is not a design process — it is a derivation. Nothing in Axiograf was chosen because it seemed useful, or borrowed from adjacent practice, or adopted by convention. Every component exists because an axiom makes it necessary.

Axiom 0 established that a system is constituted by its admissibility conditions. Axiom 1 provided the possibility space those conditions operate on. Axiom 2 defined what constraints do and, equally precisely, what they do not do. Axiom 3 made dynamics a logical necessity of non-invariance. Axiom 4 defined stability as persistence under ordered re-evaluation, without temporal assumption. From these five axioms, the derived statement follows: where admissibility conditions exist, non-invariant states cannot persist, admissible transitions occur, and configurations that remain admissible across ordered evaluations constitute stable structure.

Axiograf operationalizes that derived statement. The Canonical Fact Language makes Ω representable to C. The Adaptor Layer translates enterprise reality into that representation. The Constraint Layer holds C as a versioned, evaluable, composable predicate compiled to stratified Datalog. The distributed stateless evaluator performs the pure classification Axiom 2 requires. The Admissibility Record enforces the non-persistence that Axiom 3 demands. Bounded-horizon stability evaluation instantiates Axiom 4 as a tractable protocol operation. The cross-enterprise trust model allows the derived statement to hold across organizational boundaries without central authority.

The central challenge of autonomous enterprise is not intelligence. The agents are becoming capable. The challenge that has gone unaddressed is legitimacy: the capacity of autonomous systems to demonstrate, at the moment of action, that what they did was permissible under the constraints that

governed them. That capacity does not emerge from engineering sophistication. It requires a formal foundation. CIMD is that foundation. Axiograf is its operational form.

The AI economy will either be built on verifiable legitimacy derived from first principles, or it will be rebuilt — expensively, disruptively, after considerable damage — when the absence of that foundation becomes undeniable. Axiograf exists so that the first path is available.