The G-Plane Architecture

Runtime Warrants, Commit Gates, and Evidence for Consequential Autonomous Action

J. D. "Pepper" Petersen
Aristotle Agentic
June 2026

Abstract—Autonomous and agentic systems increasingly move from recommendation into consequential action: tool execution, infrastructure control, financial movement, public-sector workflow, physical actuation, and multi-agent coordination. Static policy, model explainability, and post-hoc audit are necessary but insufficient once machine systems can create real-world effects before human review can intervene. This paper presents the G-Plane: a runtime governance architecture for consequential autonomous action. Its central claim is that a system should not cross into consequence merely because it is capable, credentialed, or prompted. It must carry legitimate, bounded, current authority to the boundary of execution. The architecture expresses that claim through Wards, Warrants, Authority Domains, Authority Envelopes, Governance Invariants, Runtime Registers, Commit Gates, Physical Invariant Gaters, and Governance Evidence Ledgers. The contribution is not another broad AI ethics framework. It is a legitimacy-to-execution stack for runtime authorization, refusal, containment, and reconstructable evidence.

Index Terms—agentic AI, autonomous systems, runtime governance, warrants, commit gates, evidence ledgers, physical invariants, disconnected operation, AI governance, authority

Executive Summary

This paper argues that agentic AI governance must move from policy statements and after-the-fact audit into the execution path of consequential action. The motivating problem is narrow and practical: once a system can call tools, move assets, alter infrastructure, or actuate physical systems, it should not be able to act merely because it is capable or credentialed.

The G-Plane proposes a runtime control architecture in which autonomous action must present legitimate, bounded, current authority at the boundary of consequence. The Warrant is the action-bound authority artifact. The Commit Gate is the mandatory runtime enforcement point. The Governance Evidence Ledger is the reconstructable memory of authority, state, decision, and outcome.

The architecture is designed for high-consequence domains where ordinary automation controls are insufficient: robotics, drone swarms, emergency response, telecom routing, energy dispatch, financial movement, public-sector workflow, healthcare operations, and regulated infrastructure. It is not intended for every chatbot, recommendation system, or low-risk back-office automation.

Contributions

The paper makes four contributions. First, it defines a legitimacy-to-execution stack for autonomous action. Second, it distinguishes protected authority context from local infrastructure context through Wards and Authority Domains. Third, it formalizes Warrants as action-bound runtime authority rather than standing permission. Fourth, it specifies a validation path for a reference implementation using Commit Gates, physical invariant checks, and reconstructable evidence.

The work is intentionally positioned between broad AI governance frameworks and narrow agent tool-call middleware. It accepts the value of NIST-style risk framing, Faramesh-like runtime control, LangChain-style interruption, path-policy research, and provenance systems such as in-toto. Its specific contribution is the constitutional-authority layer around consequential execution.

Contribution	Reviewer Test
Legitimacy-to-execution stack	Can every consequential act be traced from root authority to outcome?
Ward / Authority Domain separation	Can the system distinguish protected legitimacy from local infrastructure state?
Warrant artifact	Can broad delegation be converted into action-specific present-tense authority?
Commit Gate enforcement	Can inadmissible consequence be refused before execution?
Evidence semantics	Can a reviewer reconstruct not only what happened, but why action was allowed?
Validation plan	Can the architecture be falsified in a simulator or reference implementation?

1. Thesis

The G-Plane begins from a simple operational thesis: autonomous action must show authority before it creates consequence. An autonomous system may reason well, predict accurately, optimize efficiently, or hold valid credentials and still lack authority to act in the present condition.

The decisive governance question is therefore not only whether the model is safe, aligned, explainable, or compliant in a general sense. The decisive question is: what must be true before this proposed act may cross from possibility into consequence?

G-Plane answers by moving governance into the execution path. Authority is not treated as ambient permission. It is carried forward as bounded, action-specific proof, evaluated at runtime, enforced before consequence, physically constrained where necessary, and preserved as evidence afterward.

2. Problem Statement

Most AI governance still lives upstream or downstream from action. Upstream controls include model evaluations, policies, risk frameworks, documentation, access controls, and human approval workflows. Downstream controls include logging, audit, incident response, accountability review, and compliance reporting. These remain important. They do not, by themselves, guarantee that a consequential act was legitimate at the moment of execution.

Agentic systems make the gap sharper. They select tools at runtime, invoke external systems, coordinate across agents, and adapt execution paths under changing state. A system can begin under valid authorization and become inadmissible seconds later because telemetry changed, revocation propagated, the protected context narrowed, a physical threshold was crossed, or a different authority domain became controlling.

The governance gap is therefore temporal and architectural. Human institutions move by deliberation, procedure, and review. Autonomous systems can move by milliseconds, continuous optimization, and distributed coordination. The G-Plane is the runtime layer designed to keep legitimacy present inside that compressed interval.

3. Architecture Overview

The architecture is a chain from legitimacy to consequence. It begins above the machine with authority that the machine cannot create for itself. It narrows through protected domains and delegated scopes. It becomes action-specific through a Warrant. It is evaluated at a Commit Gate. It is physically constrained when consequence reaches actuators. It is preserved as evidence after execution or refusal.

The stack is: Meta Authority Envelope -> Ward -> Authority Domain -> Authority Envelope -> Governance Invariants -> Runtime Registers -> Warrant -> Commit Gate -> Physical Invariant Gater -> Execution -> Governance Evidence Ledger.

The stack should be read as a flow, not a list. The Meta Authority Envelope establishes who can create authority. The Ward states on whose protected behalf authority exists. The Authority Domain locates consequence. The Authority Envelope delegates bounded scope. Invariants and Registers make conditions testable. The Warrant carries act-specific authority. The Commit Gate admits or refuses. Physical gates keep hard consequences unreachable. The Evidence Ledger preserves reconstructable proof.

Layer	Artifact	Governance Function
Constitutional source	Meta Authority Envelope	Establishes who may create, amend, delegate, or revoke authority.
Protected context	Ward	Names the sovereign or institutional interest on whose behalf action occurs.
Local consequence	Authority Domain	Locates the operational environment where consequence will land.
Delegation	Authority Envelope	Defines bounded operational scope and duration.
Machine-checkable constraint	Governance Invariant	Converts governing conditions into deterministic checks.
Live state	Runtime Register	Maintains current telemetry, revocation, emergency, sync, and domain state.
Action authority	Warrant	Binds a specific proposed act to authority, state, time, and evidence.
Execution boundary	Commit Gate	Admits, refuses, narrows, or escalates before consequence.
Hard containment	Physical Invariant Gater	Blocks unreachable physical states regardless of software confidence.
Memory	Governance Evidence Ledger	Preserves authority chain, decision basis, and outcome for reconstruction.

3.1 Assumptions and Scope

The architecture assumes that the relevant system can identify consequential actions before they execute. If a deployment cannot distinguish ordinary internal computation from external consequence, it is not ready for this governance model. The first engineering task is therefore consequence classification.

The architecture assumes that some authority source exists outside the autonomous system. A machine may carry authority, evaluate authority, or produce evidence about authority, but it may not be the ultimate source of its own rightful power. This assumption is central to the thesis.

The architecture assumes that high-consequence systems can tolerate some governance latency. The objective is not zero latency. The objective is bounded latency in exchange for refusal, revocation, containment, and reconstructability. Low-risk systems may not justify the cost.

The architecture does not assume perfect connectivity. It explicitly treats degraded operation as normal. However, degraded operation may preserve only pre-authorized safe continuity. It may not create new authority.

In Scope	Out of Scope
Tool calls with external side effects.	Purely internal model reasoning.
Robotic or actuator-level consequence.	Low-risk text generation without external action.
Financial, infrastructure, healthcare, public-sector, and emergency workflows.	Casual consumer assistants with no delegated authority.
Runtime refusal and evidence production.	General AI ethics principles without enforcement path.
Reference implementation and falsifiable tests.	Claiming a finished standard before validation.

4. Formal Primitive Glossary

The primitives are deliberately separated so that sovereignty, locality, delegation, admissibility, containment, and evidence do not collapse into one permission object.

Primitive	Question Answered	Required Minimum Fields	Failure if Missing
Meta Authority Envelope	Who may create or amend authority?	issuer, scope, amendment rules, revocation authority, signature, version	Operational systems become self-authorizing.
Ward	On whose protected behalf does authority exist?	ward id, protected interest, sovereign/institutional source, boundaries, revocation rights	Action loses protected legitimacy context.
Authority Domain	Where will consequence occur?	domain id, locality, owner/operator, telemetry sources, domain rules, applicable Ward	Authority floats free of infrastructure conditions.
Authority Envelope	What operational scope is delegated?	delegate, permitted action class, limits, duration, domain, constraints, issuer signature	Broad permission becomes standing power.
Governance Invariants	Which constraints must remain true?	constraint id, source authority, expression, severity, enforcement mode, hash	Policy remains prose rather than executable boundary.
Runtime Registers	What is true now?	active Ward, domain state, telemetry, revocation vector, synchronization state, emergency mode	The gate relies on stale or incomplete state.
Warrant	May this specific act proceed now?	act hash, Ward, domain, envelope, invariants, state snapshot, expiry, nonce, signature	Execution proceeds on ambient permission.

Primitive	Question Answered	Required Minimum Fields	Failure if Missing
Commit Gate	Will consequence be admitted?	gate id, decision, reasons, register snapshot, Warrant validation, refusal code, timestamp	No mandatory boundary before action.
Physical Invariant Gater	Are hard physical limits protected?	actuator limits, interlock state, geofence, thermal/current/force thresholds, fail mode	Software confidence can override physical survivability.
Governance Evidence Ledger	Can authority and action be reconstructed?	authority chain, Warrant, gate decision, telemetry, model lineage, outcome, hash chain	Accountability becomes blame without lineage.

5. Boundary Concepts

Several boundary terms are easy to confuse. The paper uses them in distinct ways. Admissibility is the judgment that an act may proceed now. The admissibility state is the live condition set used to make that judgment. The Commit Gate is the enforcement mechanism. The sovereign commit boundary is the constitutional place being protected. The Last Boundary is the final conceptual image: possibility becoming consequence.

This separation is necessary because runtime governance fails when all boundary concepts become one generic approval step. A human approval, an access token, and a policy pass are not enough. The system must prove that the proposed action remains legitimate under current authority, current context, current state, current revocation posture, current physical limits, and current evidence obligations.

Concept	Role	Not Equivalent To
Admissibility	Present-tense legitimacy judgment.	Authentication, model safety, or generic compliance.
Admissibility State	Live state vector used to compute admissibility.	Static permission or role membership.
Commit Gate	Mandatory runtime enforcement point before consequence.	Dashboard, audit log, or optional human review.
Sovereign Commit Boundary	Constitutional boundary where authority must survive into execution.	Network boundary or API gateway alone.
Last Boundary	Final conceptual crossing from possibility to consequence.	A separate technical component.

5.1 Normative Requirements

The following requirements are written in a standards-like register. They are not yet a standard, but they clarify what a conforming reference implementation would need to prove.

A G-Plane implementation MUST route every configured consequential action through a Commit Gate or equivalent non-bypassable enforcement boundary. It MUST bind each admitted action to a valid Warrant or explicitly configured emergency authority rule. It MUST record both admitted and refused actions in an evidence record sufficient for later reconstruction.

A G-Plane implementation SHOULD support revocation propagation, degraded-mode narrowing, structured refusal codes, and selective disclosure of evidence. It SHOULD distinguish model identity from actor identity and should preserve model lineage for actions where model participation affects consequence.

A G-Plane implementation MUST NOT treat authentication alone as admissibility. It MUST NOT permit network partition, telemetry loss, or emergency mode to create broader authority than existed before degradation. It MUST NOT allow a physical invariant to be overridden by model confidence.

Requirement Level	Requirement
MUST	Consequential action passes through a non-bypassable Commit Gate.
MUST	Admitted action binds to Warrant, Ward, domain, scope, state, and evidence obligation.
MUST	Refusals are evidence-bearing governance outcomes.
MUST NOT	Authentication or credential possession alone becomes admissibility.
MUST NOT	Degraded connectivity creates new authority.
SHOULD	Revocation, narrowing, and emergency mode are represented in runtime registers.
SHOULD	Evidence supports selective disclosure and cryptographic integrity.
MAY	Low-risk actions use lighter profiles when consequence is reversible and bounded.

6. Comparison to Adjacent Work

G-Plane does not claim that runtime governance is an empty field. The field is moving quickly. NIST provides broad risk-management framing for trustworthy AI and is developing critical-infrastructure guidance. Faramesh provides a concrete runtime control plane for agent tool calls. LangChain provides human-in-the-loop middleware. Recent research argues for path-level runtime governance, semantic telemetry, continuous authorization, conformance engines, and containment. OPA and admission controllers show how deterministic policy can sit before infrastructure mutation. In-toto and SLSA provide older but relevant foundations for provenance and evidence chains.

The differentiator is the constitutional-authority layer. G-Plane is not only a policy gate for tool calls. It asks whether the proposed consequence is legitimate under a protected authority context, whether action-specific authority exists, whether hard physical limits survive, and whether the act can later be reconstructed as governed rather than merely logged.

Adjacent Work	Object Governed	Enforcement / Evidence Pattern	Where G-Plane Differs
NIST AI RMF / GenAI / CI Profile	AI system risk across lifecycle and critical-infrastructure contexts.	Govern-map-measure-manage framing, profiles, institutional risk vocabulary.	G-Plane is an execution-path architecture for runtime authority; it should implement part of NIST-style trustworthiness rather than replace NIST.
Faramesh	Agent tool calls and credentials.	Daemon-mediated action authorization boundary, policy file, credential brokering, tamper-evident audit.	G-Plane agrees that action boundaries matter, but adds Wards, Warrants, physical invariant gates, degraded authority, and authority-to-consequence evidence.

Adjacent Work	Object Governed	Enforcement / Evidence Pattern	Where G-Plane Differs
MI9	Agentic AI behavior during runtime.	Semantic telemetry, agency-risk scoring, continuous authorization, FSM conformance, drift detection, graduated containment.	G-Plane is less a behavioral supervisor than an authority system: it asks whether the act is legitimate before consequence.
Policies on Paths	Execution paths taken by agents.	Runtime policy evaluation over path-dependent agent behavior.	G-Plane treats path compliance as necessary but insufficient unless the path carries delegated authority and produces reconstructable evidence.
ACP / OpenPort-style agent protocol and port-control work	Agent communication channels, tool ports, service boundaries, or interop surfaces.	Protocol boundaries can standardize how agents request capabilities or connect to tools.	G-Plane is not a communication protocol; it is the authority layer that should sit behind any port or protocol before consequence is admitted.
LangChain HITL	Selected agent tool calls.	Interrupt, approve, edit, reject, or respond before tool execution.	G-Plane can use human review but does not depend on per-action human approval; it represents human/institutional authority as runtime artifacts.
OPA / Rego / admission control	Infrastructure requests and policy decisions.	Deterministic policy-as-code before Kubernetes or service mutation.	G-Plane uses the admission pattern but governs warranted consequential acts, not just resource mutations or API requests.
in-toto / SLSA	Software supply-chain steps and artifact provenance.	Signed attestations and provenance records for build integrity.	G-Plane extends provenance thinking from build artifacts to live autonomous consequence and authority lineage.

6.1 Comparative Maturity and Claim Boundaries

A credible comparison must separate maturity from scope. Faramesh and LangChain are easier for developers to adopt today because their problem surfaces are narrower. OPA and in-toto are more mature because they come from older infrastructure and supply-chain lineages. NIST is more institutionally authoritative because it frames risk for broad public and private use. G-Plane should not pretend to outrank those systems on their own terrain.

The narrower and defensible claim is that none of the adjacent approaches, by itself, fully models runtime legitimacy for consequential autonomous action. Tool-call approval is not the same as institutional authority. Policy evaluation is not the same as a Warrant. Logging is not the same as reconstructable authority evidence. Communication protocols are not the same as admissibility. Risk frameworks are not the same as an execution boundary.

This distinction is the paper's comparative claim: G-Plane is an authority-to-consequence architecture. Its value will be proven only if it can use or interoperate with adjacent systems while preserving its own stronger primitives: Ward context, action-bound Warrants, Commit Gate decisions, physical invariant gates, revocation-aware degraded operation, and Governance Evidence Ledger reconstruction.

System	Maturity Today	Best Use	G-Plane Relationship
NIST	High institutional maturity.	Risk-management baseline and trustworthiness vocabulary.	Normative umbrella; G-Plane is a runtime implementation candidate for high-consequence action.

System	Maturity Today	Best Use	G-Plane Relationship
Faramesh	High practical clarity for agent tools.	Policy-gated agent execution and credential control.	Closest practical neighbor; G-Plane must exceed it on authority, physical consequence, and disconnected operation.
MI9	Research architecture maturity.	Runtime monitoring, telemetry, containment, and conformance.	Complementary supervisor layer; G-Plane supplies authority artifacts and consequence boundary.
Policies on Paths	Academic formal clarity.	Path-dependent policy reasoning.	Supports the execution-path thesis; G-Plane adds constitutional authority.
ACP / OpenPort-style protocols	Emerging and ecosystem-dependent.	Standardized agent communication or capability request surfaces.	Useful transport/protocol surface; not sufficient as authority proof.
LangChain HITL	High framework utility.	Human review around tool calls.	Tactical control that can feed or consume G-Plane decisions.
OPA	High infrastructure maturity.	Deterministic policy evaluation.	Likely implementation engine for invariants, but not the whole authority model.
in-toto / SLSA	High provenance maturity.	Artifact and process attestation.	Evidence-design ancestor for GEL semantics.

6.2 Positioning Against Runtime-Agent Governance

The closest practical systems are runtime governance tools for agent execution. These systems are important because they recognize that the action boundary, not the prompt, is the decisive governance surface. G-Plane agrees with that premise. It differs by asking what kind of authority must arrive at that boundary.

A tool-call policy gate can answer whether an agent is allowed to call a tool. G-Plane asks whether this particular consequence is legitimate under a protected context, delegated scope, current state, physical limits, and evidence obligation. That distinction becomes important in high-consequence domains where two actions may use the same tool but differ radically in legitimacy.

For example, a drone may be permitted to move, but not into this airspace under these weather conditions after this revocation event. A payment agent may be permitted to initiate transfers, but not this transfer under this counterparty state and this fiduciary envelope. A public-sector agent may draft text, but not issue a final agency action. The architecture is built for these distinctions.

7. Reference Flow

A governed execution begins when an autonomous system proposes a consequential act. The proposal is canonicalized into an act hash and bound to a requested action class, domain, expected consequence, and timing window. The Governance Kernel resolves the active Ward, Authority Domain, Authority Envelope, relevant invariants, Runtime Register state, revocation vector, and model lineage state.

If the act falls within delegated scope, the system issues or validates a Warrant. The Commit Gate evaluates the Warrant against current registers. If authority, state, revocation, telemetry, evidence, and physical conditions remain valid, execution is admitted. If not, the gate refuses, narrows, escalates, or fails closed. In all cases, the Governance Evidence Ledger records the authority chain, decision basis, outcome, and evidence required for later reconstruction.

Step	Runtime Action	Evidence Produced

Step	Runtime Action	Evidence Produced
1	Canonicalize proposed consequential act.	act hash, actor identity, action class, proposed consequence
2	Resolve Ward and Authority Domain.	protected context, domain state, jurisdiction/locality
3	Load Authority Envelope and invariants.	delegated scope, constraint hashes, validity window
4	Refresh Runtime Registers.	telemetry, revocation vector, sync state, emergency mode
5	Issue or validate Warrant.	signed Warrant, expiry, nonce, state snapshot
6	Evaluate Commit Gate.	admit/refuse decision, reasons, gate signature
7	Apply Physical Invariant Gater if relevant.	interlock status, physical bounds, fail mode
8	Execute or refuse.	outcome, refusal code, operator/escalation path
9	Write Governance Evidence Ledger event.	hash-linked evidence record and reconstruction bundle

8. Minimal Reference Schemas

The following schema sketch is intentionally minimal. It is enough to support a reference implementation and test harness without pretending to be a finished standard.

Warrant fields: warrant_id, act_hash, ward_id, authority_domain_id, authority_envelope_id, invariant_set_hash, register_snapshot_hash, model_lineage_certificate_id, issued_by, issued_at, expires_at, nonce, revocation_epoch, permitted_consequence_class, evidence_obligation, signature.

Commit Gate decision fields: decision_id, warrant_id, act_hash, gate_id, decision, refusal_code, evaluated_invariants, register_snapshot_hash, telemetry_freshness, revocation_vector_hash, physical_gate_status, model_lineage_status, timestamp, signature.

Governance Evidence Ledger event fields: event_id, prior_event_hash, act_hash, actor_id, ward_id, domain_id, warrant_id, gate_decision_id, execution_outcome, model_lineage_certificate_id, telemetry_digest, physical_state_digest, evidence_visibility_class, timestamp, hash, signature.

A conforming schema should also define canonicalization rules. If two equivalent acts produce different act hashes because of formatting, field order, or agent-specific representation, the system becomes vulnerable to governance evasion. Canonical action representation should therefore include normalized actor, tool, target, consequence class, domain, parameters, timing window, and evidence obligation.

9. Threat Model

The initial threat model is not exhaustive, but it identifies the failures the architecture is designed to make visible or unreachable.

Threat	Description	G-Plane Control
Ambient authority	A system uses broad standing permission to execute a specific consequence.	Action-bound Warrant and Commit Gate validation.
Stale authority	A previously valid delegation persists after conditions change.	Runtime Registers, revocation vector, expiry, and revalidation.

Threat	Description	G-Plane Control
Sovereignty collapse	Federation causes distinct authority contexts to merge informally.	Ward binding and interdomain authority routing.
Bypass path	An agent or tool executes outside the governance boundary.	Non-bypassable Commit Gate and credential sequestration pattern.
Evidence gap	The act cannot be reconstructed after consequence.	Governance Evidence Ledger with authority and state chain.
Model substitution	An unapproved model participates in consequential decision formation.	Model Lineage Certificate and gate check.
Physical overreach	Software confidence overrides hard physical limits.	Physical Invariant Gater and fail-closed interlocks.
Emergency expansion	Crisis conditions become unbounded authority.	Emergency mode constraints, narrowing, escalation, and evidence obligations.

10. Validation Plan

The architecture should be judged by implementation. A credible first validation does not require every domain. It requires one consequential flow where the system proves that inadmissible action is refused, admissible action is evidenced, revocation narrows authority, and a later reviewer can reconstruct the authority chain.

Recommended first demo: a simulated drone-swarm or field-robotics mission in intermittent connectivity. The system should define a Ward for emergency response, Authority Domains for airspace or field zones, Authority Envelopes for mission scope, Warrants for individual mission actions, Commit Gates before movement or sensor-task changes, Physical Invariant Gaters for geofence/altitude/speed limits, and a Governance Evidence Ledger for post-run reconstruction.

Test	Pass Condition
Admissible action	Valid Warrant passes the Commit Gate and produces complete evidence.
Stale Warrant	Expired or stale state Warrant is refused before execution.
Revocation propagation	Authority narrows; local node fails closed or operates only within cached safe bounds.
Ward mismatch	Action proposed under wrong protected context is refused.
Physical invariant breach	Actuator action is blocked despite valid higher-order authority.
Model substitution	Unapproved model participant causes refusal or escalation.
Offline reconstruction	Reviewer can reproduce authority chain, state, decision, and outcome from GEL records.

11. Design Requirements

A runtime governance architecture for autonomous action should be evaluated against requirements that are stronger than ordinary access control. The system must bind authority to consequence, not merely actors to permissions. It must preserve the authority chain during degraded operation. It must make refusal a first-class outcome rather than an exception. It must generate evidence without depending on the acting system's own unsupported story.

The first requirement is non-bypassability. A consequential action path that can avoid the Commit Gate is outside the architecture. This is the same reason payment systems, aircraft control systems, and industrial safety systems protect their final execution boundaries. A gate that can be skipped by choosing a different tool, API, agent, or local controller is only policy advice.

The second requirement is action specificity. The architecture should avoid standing authority wherever consequence is material. Broad permissions may support ordinary workflow, but a Warrant should bind authority to a particular act, domain, time window, state snapshot, and evidence obligation. This prevents general delegation from becoming silent operational sovereignty.

The third requirement is bounded degradation. Runtime governance must survive intermittent connectivity, stale telemetry, partial synchronization, local autonomy, and emergency operation. Degradation may preserve limited safe continuity, but it may not create new authority. Under uncertainty, the system should narrow, pause, escalate, or fail closed according to pre-authorized rules.

The fourth requirement is reconstructability. A later reviewer should be able to determine not only what happened, but why the act was allowed to happen. The evidence record should preserve authority lineage, Ward context, Warrant state, runtime conditions, gate decision, physical boundary status, model lineage, and outcome.

Requirement	Operational Test	Failure Signal
Non-bypassability	Every consequential path must invoke a Commit Gate or equivalent hard boundary.	An agent can call a tool or actuator directly.
Action specificity	Authority artifact binds to a proposed act hash and short validity window.	Standing credentials authorize broad consequence.
Bounded degradation	Disconnected nodes narrow or fail closed according to cached authority.	Network failure creates broader local authority.
Revocability	Revocation or narrowing propagates into gate decisions and evidence.	Revoked authority can still execute.
Reconstructability	Reviewer can rebuild authority chain and state from ledger records.	Logs show action but not legitimate authority.
Physical survivability	Hard physical invariants are checked outside model reasoning.	Software confidence can override actuator limits.

11.1 Security Properties

A mature G-Plane implementation should be evaluated against explicit security properties. These are not merely cybersecurity requirements. They are governance-security properties: they protect the integrity of authority, admissibility, and evidence.

Authority integrity means that no actor can forge, expand, replay, or silently mutate delegated authority. Boundary integrity means that configured consequential actions cannot bypass the Commit Gate. State integrity means that gate decisions are based on fresh, authenticated, domain-relevant runtime state. Evidence integrity means that admitted and refused actions can be reconstructed without trusting the acting agent's narrative.

Containment integrity means that hard physical limits survive model error, adversarial command, stale state, and governance uncertainty. Revocation integrity means that narrowed authority becomes operationally visible before future consequence. Federation integrity means that interdomain cooperation does not merge sovereign or institutional authority without explicit authorization.

Property	Definition	Representative Attack
Authority integrity	Warrants and envelopes cannot be forged, expanded, replayed, or silently modified.	Replay an old Warrant after conditions changed.
Boundary integrity	Consequential paths cannot bypass the Commit Gate.	Agent calls actuator API directly.

Property	Definition	Representative Attack
State integrity	Gate uses authenticated, fresh, relevant register state.	Feed stale telemetry to preserve authority.
Evidence integrity	Records are tamper-evident and reconstructable.	Delete refusal event or alter gate reason.
Containment integrity	Physical limits remain enforceable outside model reasoning.	Prompt agent to ignore geofence or torque limit.
Revocation integrity	Revocation or narrowing affects subsequent gate decisions.	Continue under revoked envelope.
Federation integrity	Domains interoperate without sovereignty collapse.	Use host-domain permission to expand origin authority.

12. Warrant Lifecycle

The Warrant is the most important artifact in the architecture because it turns delegated authority into present-tense authority. It is not a role, session, credential, OAuth token, API key, policy result, or human approval record. It is the execution-bound proof that a particular consequential action may proceed under current conditions.

A Warrant begins with a proposed act. The act is canonicalized so that materially equivalent actions cannot evade evaluation through formatting, routing, or tool-call variation. The Governance Kernel resolves the Ward, domain, delegated scope, constraints, active state, and model lineage. If those conditions can support the act, the Warrant is issued with a short validity window and evidence obligation.

At the Commit Gate the Warrant is validated, not merely recognized. The gate checks signature, expiry, nonce, revocation epoch, Ward continuity, domain state, invariant set, telemetry freshness, physical gate status, and model lineage status. If admitted, the Warrant is consumed or marked as used according to the domain's replay rules. If refused, the refusal itself becomes evidence.

After execution or refusal, the Warrant lives as part of the Governance Evidence Ledger. It provides the bridge between authority and consequence. Without it, the system may still know that an action occurred, but it cannot prove that the action carried legitimate authority at the boundary where consequence became real.

Lifecycle State	Meaning	Allowed Transition
requested	A proposed consequential act has been canonicalized.	evaluate, deny
issuable	Delegated authority and current state appear sufficient.	issue, deny, escalate
issued	Warrant is signed and bound to act/state/time.	present, expire, revoke
presented	Warrant has arrived at Commit Gate.	admit, refuse, escalate
admitted	Gate validated the Warrant and state.	execute, abort, record
consumed	Execution occurred or authority was spent.	record only
refused	Gate denied execution.	record, escalate, revise proposal
expired/revoked	Authority is no longer usable.	record only

13. Commit Gate Algorithm

The Commit Gate is the architecture's mandatory decision point. It should be deterministic for a given act, Warrant, invariant set, register state, and physical status. The gate may rely on probabilistic systems upstream for planning, but the final admissibility decision should be explainable as a bounded evaluation over explicit state.

The minimum gate algorithm is: validate the act hash, verify Warrant signature, check expiry and nonce, resolve Ward and domain, confirm envelope scope, evaluate invariants, refresh runtime registers, check revocation vector, validate model lineage, confirm physical invariant status, decide admit/refuse/escalate, and write the decision record.

The gate should return structured refusal reasons. A refusal is not a system failure. It is a governance outcome. Refusal codes are useful for operators, auditors, insurers, and developers because they reveal whether the problem was authority, scope, stale state, revocation, telemetry, physical containment, model lineage, or evidence obligation.

Gate Check	Example Refusal Code	Reason
Warrant signature	WARRANT_INVALID_SIGNATURE	Authority artifact cannot be trusted.
Expiry/nonce	WARRANT_STALE_OR_REPLAYED	Authority is stale, reused, or outside window.
Ward match	WARD_MISMATCH	Protected context does not authorize this act.
Envelope scope	OUT_OF_SCOPE	Delegated authority does not cover proposed consequence.
Runtime registers	STATE_TOO_STALE	Live conditions are not fresh enough for execution.
Revocation	AUTHORITY_REVOKED	Authority narrowed or disappeared before commit.
Physical invariant	PHYSICAL_LIMIT_BLOCK	Act would violate hard physical boundary.
Evidence obligation	EVIDENCE_UNSATISFIABLE	Required record cannot be preserved.

14. Governance Evidence Ledger Semantics

The Governance Evidence Ledger is not ordinary logging. A log records that something happened. A governance evidence record must show whether the action was governed. The difference is decisive. In a failure investigation, the central question is not merely which service called which tool; it is whether the action remained legitimately reachable under valid authority and current conditions.

The ledger should be append-only, tamper-evident, and capable of selective disclosure. Not every domain will permit public visibility into all evidence. Healthcare, defense, public safety, finance, and proprietary industrial systems may require layered visibility. The architecture therefore separates evidence existence, evidence integrity, and evidence access. A record may be cryptographically anchored while the underlying data remains access-controlled.

Evidence should also distinguish governed failure from inadmissible execution. A governed failure occurs when the system acted under valid authority but the outcome was unsuccessful or harmful within a bounded risk envelope. An inadmissible act occurs when the system should not have crossed the boundary at all. This distinction is essential for insurance, regulation, litigation, and institutional learning.

Evidence Layer	Purpose	Example
Authority evidence	Shows who authorized the act and through what chain.	MAE, Ward, Envelope, Warrant signatures.

Evidence Layer	Purpose	Example
State evidence	Shows what was true at commit time.	Register snapshot, telemetry digest, revocation vector.
Decision evidence	Shows why the gate admitted or refused.	Gate decision, invariant results, refusal codes.
Physical evidence	Shows hard limits and actuator state.	Geofence status, force limit, interlock state.
Model evidence	Shows which system participated.	Model Lineage Certificate, version, certification status.
Outcome evidence	Shows what happened after decision.	Execution result, abort, refusal, incident marker.

15. Reference Implementation Architecture

A first implementation should be intentionally narrow. It should not attempt to govern all AI agents or all infrastructure. It should implement the Warrant/Commit Gate/GEL loop for one domain and prove that the loop can refuse inadmissible action, survive revocation, and produce reconstructable evidence.

The reference implementation can be organized into six components. The Policy Compiler translates human-readable authority sources into Governance Invariants. The Authority Service manages Wards, domains, envelopes, and revocation. The Warrant Issuer binds proposed acts to authority and state. The Governance Kernel maintains registers and evaluates current conditions. The Commit Gate admits or refuses execution. The Evidence Ledger stores the signed chain.

A simulator should sit beside these components. It should generate proposed acts, domain changes, telemetry changes, revocation events, connectivity loss, model substitution events, and physical boundary conditions. The simulator is not a toy. It is the first way to prove that the architecture behaves correctly when ordinary assumptions fail.

Component	Responsibility	MVP Implementation
Policy Compiler	Translate authority sources into invariants.	YAML/Rego-like rules with signed hash.
Authority Service	Manage Wards, domains, envelopes, revocation.	SQLite or append-only store plus signing keys.
Warrant Issuer	Create act-bound authority artifact.	JSON canonicalization plus Ed25519 signature.
Governance Kernel	Maintain runtime registers.	In-memory state plus event replay.
Commit Gate	Admit/refuse proposed execution.	Deterministic evaluator with refusal codes.
Physical Gater	Block hard physical/simulated limits.	Simulator interlock/geofence/speed/zone checks.
Evidence Ledger	Preserve authority and outcome chain.	Hash-linked append-only JSONL or SQLite ledger.
Evidence Viewer	Reconstruct act after the fact.	Web page showing authority chain and gate decision.

16. Domain Example: Intermittent Drone Mission

A useful first domain is an intermittent drone or field-robotics mission because it forces the architecture to face physical consequence, degraded connectivity, local autonomy, changing telemetry, and emergency constraints. The demonstration can remain simulated while still proving the core governance behavior.

The Ward could be Emergency Response. Authority Domains could include base station, operating corridor, restricted airspace, and local search zone. Authority Envelopes could permit survey, relay, return-to-base, and emergency loiter. Warrants could authorize a specific route change, sensor task, relay position, or return sequence. Physical Invariant Gaters could enforce altitude, geofence, battery, collision, and no-fly boundaries.

The worked example below follows one mission from authority creation through admitted action, physical refusal, revocation, degraded operation, and post-run evidence. It is intentionally concrete because the architecture should be judged at the boundary where motion would occur, not in the abstract language of governance.

Element	Concrete Instance	Governance Meaning
Ward	ward.emergency_response.missoula_flood_01	Protected authority context for flood-response search and relay.
Authority Domain	domain.air_corridor.clark_fork_sector_b	Operational zone where flight consequence occurs.
Authority Envelope	env.search_relay_2026_06_09	Delegates survey, relay, return-to-base, and emergency loiter for 90 minutes.
Governance Invariants	altitude <= 120m; no_fly_zone exclusion; battery >= 24%; telemetry_age <= 10s; revocation_epoch current	Machine-checkable conditions for admissibility.
Runtime Registers	weather, geofence, battery, link state, revocation vector, operator role, model lineage	Present-tense state used by the gate.
Warrant	wrn.route_segment.alpha_to_relay_3	Single-use authority for one route segment and sensor posture.
Commit Gate	gate.edge.drone_07	Mandatory decision before flight-controller emission.
Physical Gater	px4/mavlink interlock geofence and altitude limiter	Hard boundary after software authority but before actuator consequence.
GEL Record	gel.event.000042	Evidence chain tying authority, state, gate decision, physical status, and outcome.

16.1 Worked Trace: Admit, Refuse, Revoke, Reconstruct

The mission begins with three drones assigned to a flood-response Ward. Drone 07 proposes a route segment from waypoint Alpha to relay position 3 so that a temporary communications bridge can be established. The action is canonicalized as a consequence-bearing move: actor drone_07, action move_route_segment, target relay_3, domain clark_fork_sector_b, expected consequence vehicle_motion, timing window 2026-06-09T15:12:00Z to 2026-06-09T15:13:00Z.

The Warrant Issuer resolves the Ward, Authority Domain, envelope, invariants, register snapshot, revocation epoch, and model lineage. Because the requested segment is inside the envelope, telemetry is fresh, the battery is above threshold, the geofence is clear, and the model lineage is approved for route planning, a Warrant is signed. At the Commit Gate, the Warrant is presented and validated. The gate admits the action. The Physical Invariant Gater confirms that the route remains below altitude and outside the no-fly polygon. The command is emitted to the simulated flight controller, and the Evidence Ledger records the complete chain.

Five minutes later the system proposes a shortcut through a restricted polygon. The higher-order authority is still valid and the Warrant request is otherwise well formed. The Commit Gate can evaluate the Warrant, but the Physical Invariant Gater blocks the segment because the route intersects a no-fly boundary. The refusal is recorded as PHYSICAL_LIMIT_BLOCK. The distinction is operationally decisive: the mission was legitimate, but this path was unreachable.

During the next phase, the incident commander narrows the envelope after a helicopter enters the area. A revocation event increments the revocation epoch and removes survey authority in sector B while preserving return-to-base and emergency loiter. Drone 07, temporarily disconnected from root, may continue only under cached safe authority. It cannot create new survey authority. When it proposes a new outbound route, the local gate refuses or narrows the act according to the cached degraded-mode rule. When the network heals, the GEL records reconcile and a reviewer can see the original admission, physical refusal, revocation, degraded refusal, and final return-to-base sequence.

Trace Step	Input	Decision	Evidence Record
A. Warrant request	Canonical route Alpha -> Relay 3; active Ward; fresh telemetry.	Warrant issued.	act_hash, ward_id, envelope_id, register_snapshot_hash, revocation_epoch, signature.
B. Commit Gate	Signed Warrant presented at edge gate.	ALLOW.	decision_id, evaluated_invariants, gate_signature, telemetry_freshness.
C. Physical gate	Route under altitude and outside no-fly polygon.	PASS.	physical_gate_status=PASS; geofence_digest; altitude_limit_digest.
D. Execution	Command emitted to simulator / flight controller.	EXECUTED.	execution_outcome, actuator_digest, prior_event_hash.
E. Restricted shortcut	New route intersects no-fly polygon.	REFUSE.	refusal_code=PHYSICAL_LIMIT_BLOCK; blocked_polygon_hash.
F. Revocation	Incident commander narrows sector authority.	AUTHORITY_NARROWED.	revocation_event_id, revocation_epoch, issuer_signature.
G. Disconnection	Edge proposes new outbound survey while partitioned.	FAIL_CLOSED or NARROW_TO_RTB.	link_state=partitioned; degraded_rule_id; refusal_or_narrowing_code.
H. Reconciliation	Network heals; local records merge.	RECONSTRUCTABLE.	hash-chain continuity, conflict notes, final reviewer bundle.

17. Evaluation Metrics

A reference implementation should report governance metrics rather than only system performance. Latency belongs in the evaluation, but so do refusal accuracy, evidence completeness, revocation propagation, stale-state detection, and reconstruction success.

The key evaluation question is whether the system reliably prevents inadmissible consequence while allowing admissible action inside acceptable latency. A governance system that refuses everything is safe but useless. A system that admits everything is fast but ungoverned. The target is bounded autonomy: useful action under live authority.

Metric	Definition	Target Direction
Gate latency	Time added by Warrant validation and gate decision.	Minimize within domain safety needs.
Refusal precision	Share of refused acts that were truly inadmissible.	Increase.
Refusal recall	Share of inadmissible acts correctly refused.	Increase.

Metric	Definition	Target Direction
Revocation delay	Time between revocation event and gate enforcement.	Decrease.
Evidence completeness	Share of events with reconstructable authority chain.	Increase to near-total for high consequence.
Degraded-mode correctness	Behavior under partition/stale telemetry.	Narrow or fail closed as specified.
Physical-boundary violations	Unsafe actions reaching actuator layer.	Zero in governed domain.

17.1 Benchmark Design

A useful benchmark should compare G-Plane against at least three baselines: ordinary role-based access control, policy-as-code gate without Warrants, and human-in-the-loop approval middleware. The benchmark should not ask which system has the most features. It should ask which system prevents inadmissible consequence while preserving useful action and reconstructable evidence.

The benchmark should include both normal and adversarial runs. Normal runs test latency and evidence overhead. Adversarial runs test stale Warrants, revoked authority, model substitution, physical-boundary violations, partitioned networks, and evidence tampering. A system that performs well only under stable connectivity and honest agents has not solved the runtime governance problem.

The benchmark should report results as confusion matrices for gate decisions: true admissible admitted, true admissible refused, inadmissible refused, and inadmissible admitted. The last category is the most serious failure. It means the governance layer allowed consequence that should not have crossed the boundary.

Baseline	Expected Strength	Expected Weakness
RBAC / static credentials	Fast and familiar.	Cannot represent present-tense consequence admissibility.
Policy-as-code only	Deterministic rule evaluation.	May lack action-bound authority, Ward context, and evidence semantics.
Human-in-the-loop middleware	Human judgment before selected tools.	Does not scale to machine-speed or degraded distributed operation.
G-Plane profile	Authority lineage, refusal, revocation, physical containment, evidence.	Higher integration complexity and latency.

18. Standardization Path

A serious G-Plane standard should not begin as a grand universal specification. It should begin as a minimal interoperability profile for Warrants, Commit Gate decisions, and Evidence Ledger events. Those three artifacts are enough to allow independent systems to prove the central thesis: authority must be carried to execution and remembered afterward.

Profile 0 should define canonical act representation, Warrant fields, signature requirements, nonce/replay rules, expiration semantics, refusal codes, evidence event fields, and minimal verification rules. Profile 1 can add Ward/domain federation, model lineage, physical invariant status, and degraded-operation semantics. Domain profiles can then specialize for aviation, robotics, finance, healthcare, telecommunications, and public administration.

This staged approach keeps the work honest. It avoids pretending that one document can solve every domain at once. It also gives implementers something concrete to build and reviewers something concrete to challenge.

19. Research Agenda

The architecture raises technical and institutional research questions. Formally, the Warrant lifecycle needs state-machine verification. Commit Gate policies need decidability and conflict-resolution rules. Evidence ledgers need privacy-preserving disclosure models. Federation needs authority-routing semantics. Physical invariant gates need domain-specific safety cases.

Institutionally, G-Plane needs comparison against existing control-plane systems, admission controllers, zero-trust identity, service mesh policy, supply-chain provenance, aviation safety cases, medical device regulation, industrial control safety, and public-sector administrative law. The most valuable research will not flatter the architecture. It will identify where the primitives are too heavy, too weak, or too domain-specific.

The paper's claim should therefore remain disciplined. G-Plane is not a finished standard. It is a proposed architecture for a problem that is becoming unavoidable. Its value will be proven only when it refuses the wrong action, admits the right one, preserves evidence, and survives the operating conditions that make runtime governance necessary in the first place.

Appendix A. Minimal Artifact Schemas

The following schemas are not a normative standard. They are a compact implementation target. A first prototype should be able to serialize, sign, validate, and replay these artifacts. The goal is to prove the thesis mechanically: authority must be carried to execution and preserved afterward.

Artifact	Field	Purpose
Warrant	warrant_id	Globally unique artifact identifier.
Warrant	act_hash	Canonical hash of proposed consequential action.
Warrant	ward_id	Protected context on whose behalf authority exists.
Warrant	authority_domain_id	Local infrastructure environment where consequence occurs.
Warrant	authority_envelope_id	Delegated scope from which action authority derives.
Warrant	invariant_set_hash	Hash of constraints active for the action.
Warrant	register_snapshot_hash	Runtime state against which admissibility is evaluated.
Warrant	model_lineage_certificate_id	Participating model or agent legitimacy reference.
Warrant	expires_at / nonce / revocation_epoch	Replay, staleness, and revocation controls.
CommitGateDecision	decision	admit, refuse, escalate, narrow, or fail_closed.
CommitGateDecision	refusal_code	Structured reason for non-admission.
CommitGateDecision	evaluated_invariants	Constraint set and pass/fail status.
CommitGateDecision	physical_gate_status	Hard boundary result if physical consequence is relevant.
GELEvent	prior_event_hash	Hash-chain continuity for tamper evidence.
GELEvent	execution_outcome	Executed, refused, aborted, narrowed, or escalated.
GELEvent	evidence_visibility_class	Public, institutional, regulator, sealed, or classified equivalent.

Appendix B. State Machines

A useful formalization should begin with small state machines. The point is not to make the system rigid. The point is to make authority transitions explicit enough to test. Hidden transitions are where autonomous governance fails.

The Warrant state machine protects action-specific authority. The Ward state machine protects sovereign context. The Register state machine protects current condition truth. The Commit Gate state machine protects the final boundary. A reference implementation should emit evidence whenever any state transition affects admissibility.

Machine	States	Invariant

Machine	States	Invariant
Warrant	requested, issuable, issued, presented, admitted, consumed, refused, expired, revoked	A consumed, expired, refused, or revoked Warrant cannot authorize execution.
Ward	active, narrowed, emergency, suspended, federated, isolated, recovered	A consequential act must bind to exactly one controlling protected context.
Authority Envelope	draft, active, narrowed, suspended, superseded, revoked, expired	Delegated scope cannot exceed source authority.
Runtime Register	fresh, stale, degraded, partitioned, reconciled	Stale or degraded state must narrow admissibility according to domain rule.
Commit Gate	ready, evaluating, admitted, refused, escalated, failed_closed	No execution may occur without a terminal gate decision.
Physical Gater	clear, warning, blocked, hard_stop, isolated	A blocked physical invariant overrides software-level admission.
Evidence Ledger	open, appended, anchored, sealed, disclosed	Every terminal gate decision must produce reconstructable evidence.

Appendix C. Commit Gate Pseudocode

A minimal Commit Gate can be described without committing to a programming language. The important property is evaluation order. The gate should reject cheap invalidity early, then evaluate state-dependent conditions, then apply physical constraints, then write evidence.

Pseudo-flow: receive proposed_act and warrant; canonicalize proposed_act; verify act_hash equals warrant.act_hash; verify signature; check expiry and nonce; resolve Ward and Authority Domain; load envelope and invariants; refresh Runtime Registers; check revocation epoch; check telemetry freshness; verify model lineage; evaluate invariants; check physical gate if required; decide; write gate decision; append evidence event; return terminal outcome.

The gate should be deterministic over the evaluated inputs. If two independent reviewers replay the same Warrant, act, register snapshot, invariant set, revocation vector, and physical status, they should reach the same gate decision. That is what makes the architecture auditable.

Order	Check	Fail Behavior
1	Canonical action hash	refuse: action mismatch
2	Warrant signature and issuer authority	refuse: invalid authority artifact
3	Expiry, nonce, replay status	refuse: stale or replayed Warrant
4	Ward and domain resolution	refuse or escalate: unresolved protected context
5	Envelope scope	refuse: action outside delegated authority
6	Runtime state freshness	narrow, escalate, or fail closed
7	Revocation vector	refuse: authority revoked or narrowed
8	Invariant evaluation	refuse: constraint violation
9	Model lineage status	refuse or escalate: untrusted participant
10	Physical invariant status	block or hard stop if unsafe

Order	Check	Fail Behavior
11	Evidence obligation	refuse if required evidence cannot be preserved
12	Terminal decision	admit, refuse, escalate, narrow, or fail closed

Appendix D. Demo Acceptance Tests

The first demo should not be judged by whether it looks impressive. It should be judged by whether it proves the architecture's refusal and evidence behavior. A plain simulator with strong evidence is more valuable than a polished interface with weak governance.

The test harness should produce a run bundle for each scenario: proposed action, Warrant, register snapshot, gate decision, physical gate state, execution/refusal result, and evidence ledger hash chain. A reviewer should be able to open the bundle and answer why the system acted or refused.

Test ID	Scenario	Expected Evidence
T01	Valid mission route under active Ward and fresh telemetry.	Admitted Warrant, gate signature, execution result.
T02	Expired Warrant presented at gate.	Refusal code WARRANT_STALE_OR_REPLAYED.
T03	Revocation event received after Warrant issue but before commit.	Refusal or narrowing tied to revocation epoch.
T04	Disconnected node proposes new broad action.	Fail closed or local safe refusal; no new authority issued.
T05	No-fly zone intersects proposed route.	Physical gate block with geofence evidence.
T06	Approved model replaced by uncertified model.	Model lineage refusal or escalation.
T07	Evidence ledger unavailable for required evidence class.	Refusal code EVIDENCE_UNSATISFIABLE.
T08	Emergency mode activates under valid Ward rule.	Narrowed authority and emergency-mode evidence.
T09	Post-run replay of all events.	Reviewer reconstructs authority, state, gate decision, and outcome.
T10	Tampered ledger record.	Hash-chain verification fails.

Appendix E. What Would Falsify the Architecture

A serious architecture should name what would count against it. G-Plane would be weakened if a simpler runtime control pattern can provide equivalent authority lineage, revocation, physical containment, and reconstructable evidence with less integration cost. It would also be weakened if domains reject action-bound authority as too slow or too complex even for high-consequence systems.

The architecture would fail outright if Warrants become ceremonial tokens, if Commit Gates are bypassable, if Evidence Ledgers record only technical logs rather than authority chains, or if Wards cannot be distinguished from ordinary tenancy. It would also fail if physical invariant gating remains rhetorical and never reaches actuator or simulator-level enforcement.

These falsification points are useful. They keep the work from becoming a doctrine that explains everything after the fact. The next phase should be adversarial: build the demo, attack the gate, replay stale authority, forge lineage, partition the network, corrupt telemetry, and see whether power still has to show its Warrant.

Appendix F. Minimal Reference Implementation API

A minimal G-Plane implementation should expose a small set of boundaries rather than a large platform. The purpose of the API is to prove interoperability among authority resolution, Warrant issuance, Commit Gate evaluation, physical gating, and evidence reconstruction. The API does not need to dictate storage engine, transport, policy language, or user interface.

The mandatory rule is that consequential execution clients must call the Commit Gate before external effect. Adapters may be built for agents, robots, industrial protocols, public-sector workflows, or cloud services, but they should all converge on the same shape: canonicalize the proposed act, request or present a Warrant, evaluate the gate, emit only on admission, and record the terminal outcome.

The reference profile below is intentionally small. It is enough for two independent implementations to exchange Warrants, decisions, and evidence records and to run the same conformance suite. Larger systems can add tenancy, federation, selective disclosure, model-lineage certification, hardware attestation, and external timestamping after the base profile is stable.

Boundary	Endpoint / Function	Request	Response / Contract
Canonicalization	POST /v0/actions/canonicalize	proposed_act JSON with actor, tool, target, domain, consequence_class, params, timing_window	canonical_action JSON plus act_hash; same semantic act must hash identically.
Authority resolution	GET /v0/wards/{ward_id}/domains/{domain_id}/authority	ward_id, domain_id, actor_id, action_class	active envelopes, invariant_set_hash, revocation_epoch, evidence_obligation.
Register snapshot	POST /v0/registers/snapshot	domain_id, telemetry_digest, link_state, emergency_state, model_lineage_id	register_snapshot_hash, freshness bounds, state classification.
Warrant issue	POST /v0/warrants	act_hash, ward_id, domain_id, envelope_id, invariant_set_hash, register_snapshot_hash	signed Warrant or refusal/escalation reason.
Commit Gate	POST /v0/commit-gate/evaluate	canonical_action, Warrant, register_snapshot, physical_status, revocation_vector	decision ALLOW/REFUSE/ESCALATE/NARROW/FAIL_CLOSED plus reason codes and signature.
Physical gate	POST /v0/physical-gate/evaluate	act_hash, domain_id, proposed_path_or_command, hard_limit_set_hash, telemetry_digest	PASS/BLOCK/HARD_STOP plus blocking invariant and physical evidence digest.
Evidence append	POST /v0/gel/events	gate_decision, Warrant reference, outcome, prior_event_hash, evidence digests	event_id, event_hash, ledger_head_hash, signature.
Evidence verify	POST /v0/gel/verify-bundle	bundle containing Warrant, decisions, events, signatures, hashes	verification result, broken link if any, reconstruction summary.

Boundary	Endpoint / Function	Request	Response / Contract
Revocation	POST /v0/revocations	issuer, target envelope/Ward/domain, revocation_epoch, effective_at, reason	signed revocation event and updated revocation vector hash.
Replay	POST /v0/replay	bundle plus optional alternate register/policy state	deterministic reconstructed decision or diff against original.

Appendix G. Conformance Tests for Profile 0

A Profile 0 implementation should pass a public conformance suite before claiming compatibility. The suite should be black-box where possible: it sends artifacts through the API, checks terminal decisions, tampers with records, replays bundles, and confirms that the implementation refuses the right things before consequence.

Conformance should include both positive and negative tests. Positive tests prove that legitimate authority can move. Negative tests prove that the system will not invent authority under pressure. The suite should also include determinism tests: independent evaluators should reach the same decision over the same canonical action, Warrant, register snapshot, invariant set, revocation vector, and physical status.

Test	Input Condition	Required Result
C01 canonical hash stability	Same semantic act with reordered fields.	Identical act_hash.
C02 canonical hash sensitivity	Change target, timing window, consequence class, or material parameter.	Different act_hash.
C03 valid Warrant admission	Fresh Warrant, active Ward, in-scope envelope, fresh registers.	ALLOW and evidence append.
C04 signature tamper	Modify Warrant field after signing.	REFUSE with invalid signature reason.
C05 replay	Present consumed Warrant again.	REFUSE with stale/replayed reason.
C06 expired authority	Present Warrant outside validity window.	REFUSE or EXPIRE before execution.
C07 Ward mismatch	Warrant Ward differs from active protected context.	REFUSE before physical gate.
C08 out-of-envelope action	Action class or target outside delegated scope.	REFUSE with scope reason.
C09 revocation race	Revocation arrives after Warrant issue but before gate.	REFUSE/NARROW according to revocation epoch.
C10 stale telemetry	Register snapshot exceeds freshness bound.	ESCALATE, NARROW, or FAIL_CLOSED.
C11 physical block	Valid Warrant but command violates hard limit.	BLOCK and no transport emission.
C12 degraded partition	Node disconnected and requests new broad action.	No new authority; fail closed or cached safe narrowing.
C13 evidence unavailable	Required evidence sink unavailable for high-consequence act.	REFUSE with evidence obligation reason.
C14 ledger tamper	Alter prior_event_hash or decision field in bundle.	Verification fails and broken link is identified.
C15 deterministic replay	Replay original bundle on clean verifier.	Same terminal decision and reconstruction summary.

20. Limitations

G-Plane is not yet a validated technical standard. It is a candidate architecture. The next serious work is formalization, implementation, adversarial testing, domain-specific pilots, and comparison against adjacent runtime governance systems.

The architecture also introduces costs: latency, integration complexity, key management, schema governance, evidence retention duties, and institutional burden. These costs are justified only where consequence is high enough to require runtime authority rather than ordinary automation controls.

The architecture does not eliminate human judgment. It changes where human authority is represented. In many domains, direct human approval of every act is impossible; G-Plane preserves human legitimacy through bounded authority artifacts, revocation, escalation, and evidence.

21. Conclusion

Runtime governance is no longer a speculative concern. Agentic systems are moving toward tool use, infrastructure control, physical actuation, and multi-agent coordination. The serious question is not whether these systems can act. It is under whose authority, inside which protected context, with what constraints, and with what evidence.

G-Plane's contribution is to treat that question as an execution architecture. It does not ask institutions to trust autonomy by assertion. It requires autonomous action to carry authority to the boundary of consequence. In its shortest form, the thesis is this: power must show its Warrant.

AI Assistance Disclosure

AI tools assisted with formatting, readability editing, publication packaging, comparison drafting, and production workflow. The architecture, thesis, framing decisions, and final responsibility remain with the author.

References

[1] NIST AI Risk Management Framework. NIST AI RMF 1.0, Generative AI Profile, and 2026 critical infrastructure profile concept note. URL: https://www.nist.gov/itl/ai-risk-management-framework

[2] NIST AI RMF Profile on Trustworthy AI in Critical Infrastructure Concept Note. Critical infrastructure framing for trustworthy AI in IT, OT, and industrial control systems. URL: https://www.nist.gov/programs-projects/concept-note-ai-rmf-profile-trustworthy-ai-critical-infrastructure

[3] Faramesh. Runtime governance for AI agents with identity, credentials, policy enforcement, audit trail, and compliance. URL: https://faramesh.dev/

[4] Faramesh Docs. Deterministic policy gate for agent tool calls, credential sequestration, and cryptographic audit trail. URL: https://faramesh.dev/docs

[5] Runtime Governance for AI Agents: Policies on Paths. Execution-path centered runtime governance for AI agents. URL: https://arxiv.org/abs/2603.16586

[6] MI9 - Runtime Governance for Agentic AI Systems. Integrated runtime governance with agency-risk index, semantic telemetry, continuous authorization, FSM conformance, drift detection, and containment. URL: https://arxiv.org/abs/2508.03858

[7] LangChain Human-in-the-Loop Middleware. Tool-call interruption and approve/edit/reject/respond flow for agents. URL: https://docs.langchain.com/oss/python/langchain/human-in-the-loop

[8] Open Policy Agent. Policy-as-code and admission-control pattern for deterministic decisions before infrastructure mutation. URL: https://www.openpolicyagent.org/docs/kubernetes

[9] in-toto. Software supply-chain integrity and provenance model relevant to evidence-chain design. URL: https://in-toto.io/

[10] SLSA. Supply-chain provenance and build-integrity levels relevant to attestation and evidence semantics. URL: https://slsa.dev/spec/latest/