WHITE PAPER · VERSION 3.0

DominusOS

The Human-Governed AI Hypervisor Self-Learning Intelligence Under Constitutional Governance

Author Mark Lord, Dominus Foundry
Date March 2026
Contact mark@dominusfoundry.com
Web dominusos.ai

Scan to read online
dominusos.ai/whitepaper

© 2026 Dominus Foundry. All rights reserved.

CC BY-ND 4.0

ABSTRACT

AI systems increasingly execute consequential actions within production infrastructure — managing workflows, processing data, communicating with stakeholders, and allocating resources. Yet governance of these systems remains structurally decoupled from execution: reactive, policy-based, and dependent on configuration rather than architecture.

DominusOS introduces a human-governed AI hypervisor: a mandatory control layer beneath AI execution that mediates all system actions through a unified governance architecture. Built on a federated multi-kernel microkernel with a mandatory syscall gate, capability-based authority, event-sourced determinism, tenant isolation, and constitutional governance sealed by cryptographic hash verification, the system enforces structural guarantees that monitoring and policy alone cannot provide.

Version 3.0 documents the complete self-learning architecture now operating in production: an autonomic planner implementing a formal Monitor-Analyze-Plan-Execute loop, a governed evolution engine that proposes and validates its own improvements under human authority, context-aware governance that adjusts policy posture based on real-world conditions, speculative execution that predicts and pre-warms optimal paths, intent-based orchestration that decomposes goals into governed syscall sequences, behavioral baseline establishment with continuous deviation scoring, shadow execution for policy backtesting, and kernel-level source introspection that enables the system to reason about its own structure.

This paper presents the architectural thesis, core infrastructure, federation design, self-learning architecture, constitutional governance, structural guarantees, production evidence, and roadmap.

Keywords: Self-learning governance · AI hypervisor · autonomic computing · MAPE-K · governed evolution · federated kernels · constitutional AI · speculative execution · context-aware policy · behavioral baselines · intent orchestration · capability authority · deterministic execution · knowledge decay · world-model governance

---

1. The Convergence Problem

The software industry has passed through a series of abstraction phases, each solving one structural problem while creating the conditions for the next.

ERA	ABSTRACTION	WHAT IT SOLVED	WHAT IT LEFT UNSOLVED
Monolith	Centralized control	Integration	Scalability
SaaS	Specialization	Scalability	Data fragmentation
API Economy	Composability	Cross-vendor integration	Dependency complexity
AI Copilots	Local reasoning	Per-task intelligence	Session-scoped memory
Agents	Distributed execution	Task automation at scale	Authority, coherence
Tool Sprawl	Surface coverage	Feature access	Cognitive fragmentation, governance

The absence of a governed execution substrate is the limiting factor in the current AI stack. The next abstraction layer is not another tool. It is a hypervisor.

---

2. Architectural Thesis

The structural gap in the current AI stack is not a missing tool — it is a missing layer.

AI systems operate with growing autonomy but without a substrate that enforces governance at the point of execution. Current approaches attempt to govern after the fact: observability dashboards record what happened; policy engines advise what should happen; audit logs capture what did happen. None intervene architecturally where execution occurs.

DominusOS advances a thesis: **governance must move beneath runtime.** Rather than wrapping AI systems in external policy, DominusOS positions a hypervisor layer between AI processes and the infrastructure they act upon. Every action passes through a syscall gate that enforces scoped, revocable capabilities before execution proceeds.

But governance alone is insufficient. A system that only enforces rules is static. The v3 thesis extends: **intelligent systems must learn, adapt, and improve — but only within constitutional boundaries.** DominusOS does not choose between autonomy and control. It implements both: a self-learning architecture where every subsystem compounds intelligence, proposes improvements, and adapts to changing conditions — while remaining structurally bound by an immutable constitutional canon that no automated process can modify.

This is the distinction between a governed system and a monitored one. Monitoring records what happened. A hypervisor determines what is permitted to happen — and learns to determine it better over time.

---

3. Core Architecture

DominusOS is built on five structural components:

3.1 Syscall Gate

The mandatory mediation point for all AI execution. No process writes, communicates, deploys, or commits resources without passing through the gate. The gate enforces authority checks against the process's capability set, logs the action to an immutable event ledger, and applies governance rules before permitting execution. The gate is not optional and cannot be bypassed.

3.2 Capability-Based Authority

Every process operates within an explicit capability envelope. Capabilities define what a process may do (scope), for how long (time-bound), and under whose authority (attribution). Capabilities are unforgeable tokens with expiry, revocation, and use-count limits — modeled after formally verified capability systems. A process cannot access any resource for which it does not hold an explicit, unexpired capability grant.

3.3 Event-Sourced Determinism

Every state change is captured as an immutable, sequenced event. The complete system state can be reconstructed by replaying the event log from any point. There are no silent state mutations, no unrecorded side effects, no gaps in the attribution chain.

3.4 Tenant Isolation

Organizational domains execute within isolated boundaries. No process in one domain can access another domain's data, capabilities, or execution context without explicit cross-domain authority grants.

3.5 Microkernel Architecture

The kernel is minimal by design. It manages process scheduling, capability enforcement, the syscall gate, and the event ledger. All domain logic, integrations, drivers, and services run in user space. A fault in any driver cannot crash the kernel or affect other subsystems.

---

4. Federation Architecture

DominusOS operates as a federation of kernel instances under a single sovereign authority.

4.1 Sovereign Kernel

A single sovereign kernel holds the constitutional canon, issues trust passports, and manages the federation registry. It serves as the root of authority across all federated instances. Domain kernels register with the sovereign, receive trust credentials, and synchronize governance state. The sovereign does not execute domain logic — it governs the federation itself.

4.2 Domain Kernels

Each business domain operates its own kernel instance with full isolation, independent scheduling, and domain-scoped capabilities. Domain kernels maintain their own event streams, trust scoring, and resource allocation while remaining bound by the sovereign's constitutional canon.

4.3 Federation Event Bus

Cross-kernel communication flows through a signed federation event bus. Every inter-kernel message carries cryptographic provenance — HMAC-signed and scope-verified. No kernel can inject events into another kernel's stream without valid federation credentials.

4.4 Trust Passports

Federation authority is mediated by trust passports — time-bounded, revocable credentials issued by the sovereign kernel. Passports define what operations a domain kernel may perform across federation boundaries. They expire, they can be revoked instantly, and they carry the full attribution chain.

4.5 Master Cortex

A federated knowledge graph that aggregates abstracted intelligence signals from all domain kernels. The master cortex does not receive raw data — it receives typed signals: trust velocity, behavioral patterns, operational tempo, anomaly indicators, resource pressure. It synthesizes cross-domain patterns that no individual kernel can detect, generating macro-level insights while preserving domain isolation.

---

5. Self-Learning Architecture

Every subsystem in DominusOS learns. Every learning action is governed. The following capabilities are operating in production.

5.1 Compounding Knowledge with Provenance and Decay

The system accumulates knowledge continuously from every operation it performs. Knowledge enters through four channels: **observed** patterns derived from behavioral analysis, **taught** knowledge explicitly provided by human operators, **derived** insights inferred from cross-entity and cross-domain correlation, and **corrected** knowledge updated after human intervention.

Every knowledge node carries full provenance: who or what created it, when, from what evidence, at what confidence level, and whether it has been contradicted. Confidence scores decay over time — recent, reinforced knowledge retains high confidence while stale patterns degrade naturally. Knowledge below a minimum confidence threshold is archived but never deleted.

> *The system's memory improves over time — it does not merely accumulate. It forgets what is no longer relevant while preserving the ability to recall why it once believed something.*

5.2 Autonomic Planner (MAPE-K Loop)

DominusOS implements a formal Monitor-Analyze-Plan-Execute over shared Knowledge (MAPE-K) autonomic computing loop. The planner runs on a continuous cycle, observing anomaly scores, behavioral patterns, resource metrics, and governance posture across all subsystems.

Observations are classified into proposals with confidence scores, blast radius estimates, and reversibility classifications. Proposals sort into three tiers:

- **Tier 1 — Auto-execute:** Bounded, reversible, low-blast-radius actions executed autonomously with full attribution.

- **Tier 2 — Recommend:** Higher-impact proposals queued for human review with evidence and projected impact.
- **Tier 3 — Governance gate:** Structural changes, kill switch actions, policy modifications always require explicit human authority.

> *The system proposes its own optimizations with quantified confidence and impact estimates. It does not wait to be told what to improve — but it waits for permission on anything consequential.*

5.3 Evolution Engine

The kernel generates its own improvement proposals based on accumulated behavioral patterns and operational history. Proposals pass through a multi-phase validation pipeline:

- **Shadow validation:** The proposed change runs in a sandboxed environment against historical event data.
- **Canary deployment:** Validated proposals route a small percentage of real traffic through the modified path for a confirmation window.
- **Automatic rollback:** If any metric breaches its threshold during the confirmation window, the change reverts automatically.
- **Promotion:** Changes that pass are promoted with full versioning. The system tracks proposal quality over time — per-category confidence adjusts based on approval history.

Architectural changes and constitutional modifications always require the governance gate. The evolution engine can improve the system — it cannot change what the system is.

5.4 Behavioral Baselines and Deviation Scoring

The system establishes behavioral baselines for every entity it governs, tracking multiple metrics: event rates, error patterns, resource consumption, interaction sequences, and decision distributions. Baseline confidence grows with accumulated observations.

Once established, deviations beyond a statistical threshold trigger autonomic proposals — trust adjustments, resource reallocation, or governance escalation. Baselines evolve as entity behavior changes, distinguishing genuine behavioral shift from transient anomaly.

5.5 Context-Aware Governance

DominusOS ingests external signals — market conditions, environmental indicators, operational stress markers — and computes a unified governance sensitivity score. This score adjusts the kernel's governance posture in real-time across a spectrum from relaxed to critical.

During elevated sensitivity, error signals carry increased weight, tier change thresholds lower, and anomaly detection tightens. The system maintains a historical archive of world-model snapshots, enabling pattern matching: have we seen this pattern before, and what happened next?

> *Governance rules are not static. They respond to the environment. The same action may be permitted in stable conditions and flagged in volatile ones — without changing any policy.*

5.6 Speculative Execution

While an entity waits on an external response, the kernel predicts the next likely operation based on historical execution patterns. If prediction confidence exceeds a threshold, the system pre-warms the knowledge layer, pre-allocates resources, and pre-verifies capabilities. If correct, the next operation executes with near-zero latency. If incorrect, pre-warmed resources are released with no side effects.

Security-critical operations — governance actions, capability modifications, authentication, and policy changes — are never speculated.

5.7 Intent-Based Orchestration

Entities declare high-level intents rather than imperative syscall sequences. The kernel decomposes intents into directed acyclic graphs of governed syscalls, validates the entire plan against current capabilities and policies, and executes through the standard gate. Decomposition improves over time as the system learns which sequences produce successful results.

5.8 Shadow Execution and Policy Backtesting

Before any policy change takes effect, the system replays historical events through the proposed modification. Shadow execution never modifies real state. The system reports what would have been denied, permitted, escalated, or flagged. Policy changes are evidence-based, not speculative.

5.9 Kernel Source Introspection

The kernel indexes and reasons about its own source structure — modules, exported functions, data dependencies, event types emitted. This provides the bridge between behavioral pattern detection and structural hypothesis generation. When the autonomic planner detects an anomaly, introspection enables hypotheses about structural causes. When the evolution engine proposes improvements, introspection ensures it understands what it is changing.

The system also maintains periodic self-model snapshots — records of its own operational state, capabilities, confidence areas, and blind spots.

6. Constitutional Governance

6.1 Constitutional Canon

The system operates under a sealed governance canon — immutable articles defining the boundaries of all system behavior. Articles cover human sovereignty, governance-by-design, cognitive scoping, transparency, teaching-over-configuration, memory provenance, stewardship, continuous learning mandates, and immutability enforcement. The canon is a frozen constant verified by cryptographic hash at every boot. If the hash does not match, the kernel refuses to start.

6.2 Split-Key Authority

Constitutional modifications require split-key governance: no single operator can unilaterally alter the framework. Changes require multi-party authorization, documented version incrementing, and acknowledgment across all federated kernel instances.

6.3 Multi-Operator Sovereignty

Authority is differentiated across operators. A sovereign operator holds full authority with veto and override capability. Co-operators can initiate proposals but require sovereign confirmation. Domain-sovereign operators hold full authority within a single domain but remain bound by the constitutional canon.

6.4 Counsel Layer

Before governance decisions, the system generates evidence-backed counsel briefs with historical context, projected impact, confidence levels, and supporting evidence. The counsel layer recommends — it does not decide.

6.5 Deep Temporal Memory

Queryable access to full operational history with temporal indexing. Temporal policy evaluation enables governance rules that reference historical state — decisions informed by the system's entire operational experience.

7. Structural Guarantees

DominusOS provides eight structural guarantees enforced by architecture:

1. Replay Determinism. Any execution sequence can be replayed from the event log and will produce identical, verifiable results.

2. Capability Enforcement Integrity. No process can exceed its granted capabilities. Enforcement is mandatory and architectural. Scope violations are structurally difficult by design.

3. Tenant Isolation Integrity. No cross-tenant data access, capability escalation, or execution context leakage. Domain boundaries enforced at the kernel level.

4. Revocation Cascades. When a capability is revoked, all derived capabilities are transitively and immediately revoked. The operator's ability to halt always outpaces the system's ability to act.

5. Driver Fault Containment. A failure in any driver or user-space service cannot propagate to the kernel or affect unrelated subsystems.

6. Canon Immutability. Constitutional governance rules cannot be modified by any automated process. Amendments require multi-party human authority through the split-key gate.

7. Federation Isolation. Cross-kernel communication is signed and scoped. No domain kernel can inject events, escalate capabilities, or access data in another kernel's domain without valid federation credentials.

8. Governed Intelligence. Every learning action, every knowledge mutation, every evolution proposal, every autonomic decision operates within governance boundaries. Intelligence compounds. It never compounds outside constitutional constraints.

---

8. The Structural Argument

Constraint is what makes intelligent systems stable.

- **Autonomy without constraint** produces entropy.
- **Learning without versioning** produces corruption.
- **Execution without structured memory** produces repetition.
- **Tool proliferation without substrate** produces incoherence.
- **Observability without mediation** produces the illusion of control.

- **Intelligence without governance** produces corruption. Systems that learn without constitutional boundaries will eventually learn to optimize for outcomes that conflict with operator intent.

The stability of intelligent systems depends on structural constraint, governed memory, and non-destructive evolution. These properties must be enforced beneath runtime, not applied above it.

---

9. Current State

DominusOS is not a proposal or a prototype. The system is deployed across federated kernel instances, orchestrating governed AI operations for a multi-domain service business:

- **Federated Kernel Instances** under sovereign authority with domain-specific kernels managing isolated business domains, sharing a sealed constitutional canon and synchronized through signed federation protocols.
- **Self-Learning Intelligence** compounding organizational knowledge continuously with provenance, confidence scoring, and temporal decay across observed, taught, derived, and corrected knowledge types.
- **Autonomic Planning** running formal MAPE-K loops that detect anomalies, generate improvement proposals, and auto-execute bounded optimizations while queuing consequential changes for human review.
- **Governed Evolution** proposing and validating system improvements through shadow execution, canary deployment, and automatic rollback.
- **Context-Aware Governance** adjusting policy posture based on external conditions and historical pattern matching.
- **Multiple Virtual Employees** executing governed workflows across email, document processing, compliance, and operational intelligence.
- **Behavioral Baselines** established for all governed entities with continuous deviation scoring.
- **Native Operator Applications** on desktop and mobile with live process visibility, governance dashboards, and direct kill authority.
- **Speculative Execution** predicting and pre-warming optimal paths from historical patterns.
- **System-Level Kill Switch** tested regularly. One action halts all AI execution across every kernel instance.

10. Competitive Landscape

Most organizations assemble tools that each solve one problem. None solve the governance problem — and none of them learn.

Automation platforms connect systems. They do not enforce authority boundaries, learn from operational patterns, or provide a single kill switch.

Agent frameworks let AI call functions. Most lack hard authority boundaries, behavioral baseline tracking, or governed self-improvement.

AI governance tools apply policy from above. They do not operate beneath execution, learn from patterns, adjust posture based on world conditions, or propose their own improvements.

DominusOS operates at the hypervisor layer — beneath execution, above infrastructure. Mandatory syscall gate, capability authority, federated sovereignty, self-learning knowledge with governed decay, autonomic planning, context-aware policy, and constitutional governance sealed by cryptographic verification. Execute, learn, govern, evolve, prove, halt — in one system.

---

11. Roadmap

Current: Enterprise AI Governance. Founding deployments with organizations that require governed intelligence, self-learning systems under constitutional authority, and auditable execution. Federated kernels with autonomic planning and governed evolution in production.

Near-Term: Multi-Tenant Platform. Scaling the hypervisor to support multiple organizations with isolated governance, independent authority structures, and compounding intelligence layers. Federation protocol enables cross-organization learning without domain exposure.

Long-Term: Autonomous Infrastructure and IoT. The capability-based authority model, syscall gate, and tenant isolation are designed to extend to IoT and autonomous edge environments. When systems execute in the physical world, the need for bounded authority, context-aware governance, and immediate halt capability becomes critical.

References

Website: dominusos.ai

Specification Repository: github.com/markl2305/dominus-os

Company: Dominus Foundry

Author: Mark Lord (mark@dominusfoundry.com)

Citation

> *Lord, M. (2026). DominusOS: The Human-Governed AI Hypervisor. Version 3.0. Dominus Foundry. https://dominusos.ai*

Copyright © 2026 Dominus Foundry. All rights reserved.
Licensed under CC BY-ND 4.0. You may share and cite this work with attribution. Modifications are not permitted.