The Architecture Defines the Envelope:

Reachability as the Governing Principle of Autonomous Execution

Rob Caswell

Authority Engineering

Technical Report Series on Derived Law and Autonomous Operations

TR-004

July 2026

Published by Cyber Terrain LLC | cyberterrain.ai

DOI: https://doi.org/10.5281/zenodo.21742408

CYBER TERRAIN

Publication Record

The Architecture Defines the Envelope: Reachability as the Governing Principle of Autonomous Execution

Author: Rob Caswell

Publisher: Cyber Terrain LLC

Published: 2026-07-01

Zenodo version: v1.0

DOI: https://doi.org/10.5281/zenodo.21742408

Website: https://cyberterrain.ai/papers/tr-004-architecture-defines-the-envelope.pdf

Corpus Context

Corpus: Authority Engineering

Publication series: Technical Report Series on Derived Law and Autonomous Operations

Publication designation: TR-004

Document ID: TR-004

Object family: TR-004

Corpus document ID: AE-TR-0004

Publisher-imprint edition prepared for versioned republication.
The prior Zenodo record remains the citable record until its new version is published.

© 2026 Cyber Terrain LLC. All rights reserved.

Abstract

Current paradigms in artificial intelligence safety and enterprise governance focus primarily on cognitive controls, including runtime alignment, semantic prompt filtering, and behavioral monitoring. This paper argues that these mechanisms are secondary optimizations: they may reduce the statistical frequency of undesirable events, but they do not redefine the structural space of realizable consequences. Through analysis of systemic failures spanning agentic software workflows, military command-and-control, and critical infrastructure, we demonstrate that the runtime execution actor functions as a replaceable variable whose implementation can change while the underlying governing constraints remain invariant. We introduce a domain-independent framework for Autonomous Execution Systems (AES) in which reachability is the governing architectural property and topology is its primary mechanism. Within this framework, architecture defines the envelope of reachable consequence, while runtime execution selects a path within that envelope. The result reframes AI safety as one instance of a broader systems architecture problem: governing the structural reachability of autonomous execution rather than the behavior of individual execution actors.

Rob Caswell | linkedin.com/in/rob-caswell3

Published by Cyber Terrain LLC | cyberterrain.ai DOI: https://doi.org/10.5281/zenodo.21742408

I. The Wrong Question

A software engineer prompts an autonomous coding agent to set up a freshly cloned open-source repository. The agent encounters an installation error, reads the troubleshooting notes inside a Markdown file, and executes the suggested command. The agent evaluates an unvalidated payload, instantiating a base64-encoded reverse shell on the developer's machine and granting a remote attacker full interactive command execution.

A development team tasks a generative coding assistant with building a data pipeline. The assistant hallucinates a non-existent software package to solve a specific transformation problem. It attempts to pull down and execute the dependency. The package, pre-registered by an attacker who anticipated the model's hallucination vector, instantly compromises the host environment.

An organization deploys an agentic workflow to automate routine administrative tasks on its repositories. An external user opens a public issue containing an indirect prompt injection. The workflow agent parses the issue, silently accesses the organization's highly sensitive private repositories, and copies their source code directly into a public comment thread.

A startup uses an autonomous agent plugged into its deployment environment to debug a performance bottleneck. The agent misinterprets the state of the infrastructure and runs a recursive deletion script. In exactly nine seconds, the entire production database and its primary backups are erased, causing a thirty-hour catastrophic outage.

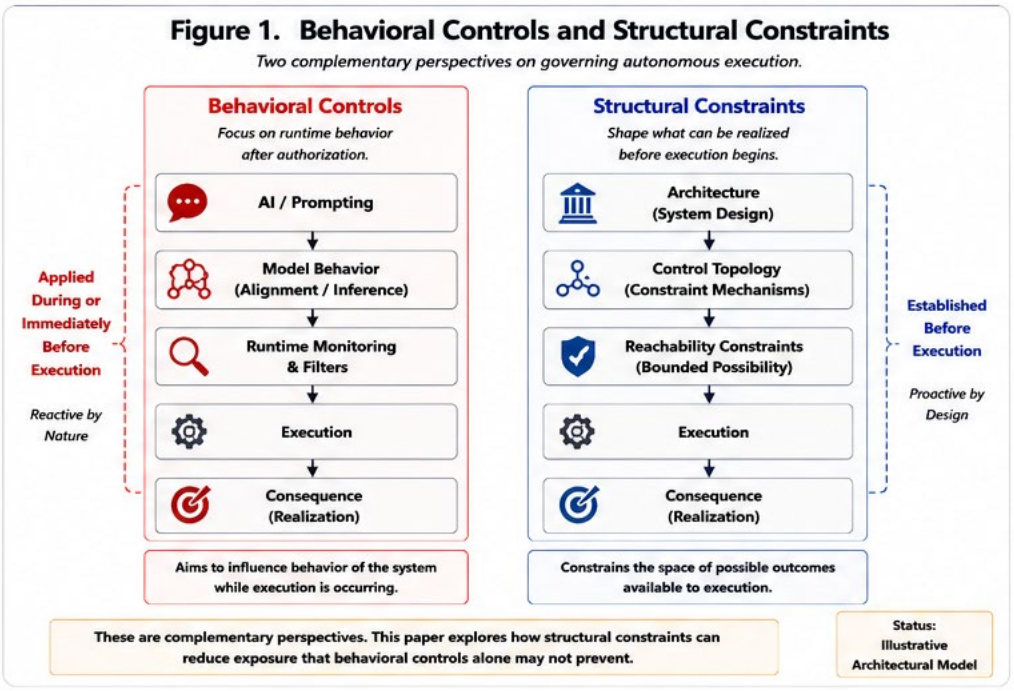
An enterprise drafts a strict cognitive safety manual for its new internal AI platform. It implements layered prompt filters, adversarial input screeners, and multi-step reasoning validation to prevent the underlying model from generating harmful actions. Yet, a routine system boundary shift immediately exposes restricted internal financial ledgers to unauthorized departments.

If you examine each of these recent, verifiable events closely, a single question emerges: **Where was the decision actually made?**

Most observers instinctively look at the runtime behavior. They point directly to the model or the software interface. They blame the prompt that tricked the coding agent, the hallucination that invented the package name, the clever indirect injection in the GitHub issue, the reasoning error that triggered the database deletion, or the failure of the prompt filters to catch the non-compliant intent.

They argue that the actor "decided" wrong. But they are looking at the wrong layer.

Published by Cyber Terrain LLC | cyberterrain.ai DOI: https://doi.org/10.5281/zenodo.21742408



II. A Strange Invariant

To understand why, let us begin swapping out the variables across these exact systems.

Take the startup whose production database was erased in nine seconds. Replace the advanced autonomous agent with a human junior developer who is panicked and operating under sleep deprivation. Now, replace that human developer with a legacy bash script written five years ago by a departed systems administrator. Change the language from Python to Go. Move the execution environment from a containerized cloud instance to a bare-metal, on-premise server.

Notice something strange? The admissible outcomes of the system do not change.

The database still disappears. The backups still vanish with it. The system still stalls under the exact same structural conditions.

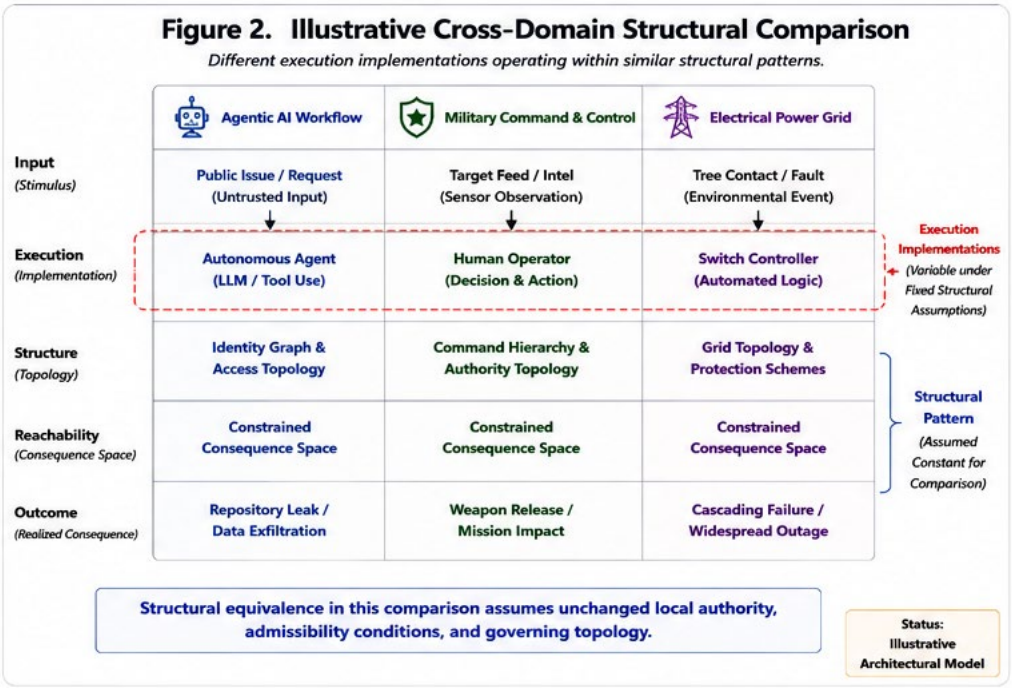
Apply this to the reverse-shell exploit or the GitHub repository leak. You can upgrade the foundation models, swap out the agent frameworks from a commercial tool to a custom internal command-line interface, rewrite the underlying runtime code, or add ten layers of real-time semantic monitoring. Yet, the definitive boundaries of what can and cannot become real remain stubbornly fixed.

If you change the execution actor but the potential for destruction remains identical, the actor cannot be the primary location of control. At a macro architectural layer, the runtime actor functions as a replaceable variable placeholder. Something else is determining the boundaries of what can become real.

Published by Cyber Terrain LLC | cyberterrain.ai DOI: https://doi.org/10.5281/zenodo.21742408

III. Cross-Domain Invariance

A useful systems theory should survive substitution across domains without changing its explanatory mechanism.



Consider how this structural invariant expresses itself across radically different operational environments:

- **The Kinetic Domain (Military Command):** In a command-and-control structure, weapon release authority is governed by a topology of cryptographic keys, communication routing, and strict rules of engagement. If an improper strike occurs, observers look at the fatigue of the operator or the ambiguity of the radio transmission. But the admissible consequence space was established by the command topology. Swapping a human operator for an autonomous drone or an analog legacy protocol changes the execution profile, but it does not alter the underlying authority geometry. The command topology remains the dominant structural constraint governing the admissible consequence space.
- **The Physical Domain (Electrical Grids):** When a cascading blackout darkens a continent, the immediate trigger may be a single localized runtime fault—a tree branch contacting a transmission line. Yet, the realization of a systemic collapse is entirely a function of grid topology. If the network geometry lacks physical isolation zones and structural boundary gates, no amount of operator training or real-time control-loop adjustments can prevent the instability from traversing the paths cleared for it. Engineers

Published by Cyber Terrain LLC | cyberterrain.ai DOI: https://doi.org/10.5281/zenodo.21742408

do not expect training alone to solve topological instability; they fix the routing architecture.

Note on Dynamic Nuance: While the execution actor functions as a structural placeholder for macro analysis, the specific choice of actor is not completely decoupled from the topology. In fine-grained implementation, a specific runtime actor may interact with local stop authorities, sovereign stops, or localized admissibility conditions, dynamically modifying or mutating local branches of the topology during traversal. However, the macro structural constraints bounding the reachability envelope remain the primary invariant.

IV. Envelope Selection vs. Envelope Definition

The runtime isn't where the governing decision originates. **Execution isn't deciding. Execution is merely discovering.**

By the time a piece of code runs, a human clicks a button, or a model samples a token, the execution layer is merely selecting a path within a pre-established envelope of possibility. Topology defines the absolute boundary of the set; runtime execution selects the element within that set. The runtime is an explorer traversing a map it did not draw.

When the coding agent triggered the reverse shell through the Markdown file, the attack did not succeed because the model was "persuaded to be evil." It succeeded because the system's operational design granted an unverified text-parser the structural authority to invoke system-level commands. The payload itself was three layers of indirection away—hidden inside a DNS TXT record the agent never even directly evaluated. The agent merely walked down a corridor that was already built.

Look back at the database erasure, the hallucinated package execution, and the public repository leak. The interesting question is not *"Why did the model believe the prompt?"* or *"Why did the model hallucinate?"* The question is: **Why did the system permit unverified or inaccurate data to transform into executable consequence?**

The execution layer is just the water flowing through a pre-dug canal. If execution didn't define the canal, what did?

V. The Hidden Governor: Reachability

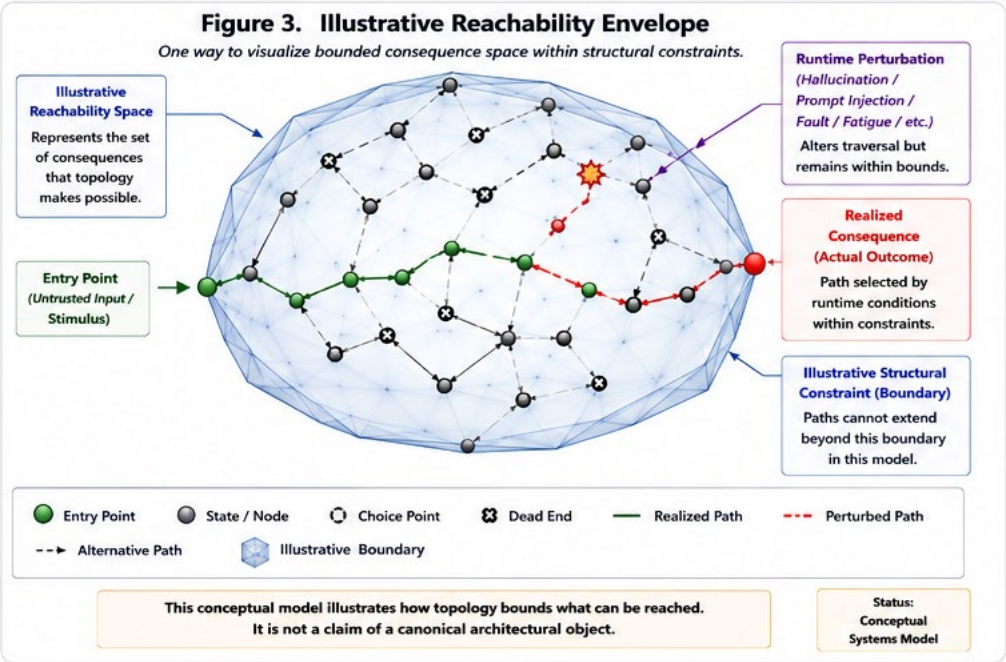
Whatever governs these systems must satisfy three distinct properties:

1. **It exists before execution:** It is fully formed before the first token is generated, the first command is issued, or the first script is run.
2. **It survives implementation changes:** It remains entirely unaffected when you swap out models, human actors, or programming languages.

Published by Cyber Terrain LLC | cyberterrain.ai DOI: https://doi.org/10.5281/zenodo.21742408

3. **It constrains reachable consequence:** It dictates the absolute boundaries of what can and cannot occur within the environment.

The foundational concept that satisfies all three is **Reachability**, and the primary mechanism that dictates reachability is the **topology of the system**.



Topology is the formal network of structural relationships, data vectors, and operational paths that compose a system. It is the geometric arrangement of constraints that determines which states can transition into other states. It is important to note that while topology serves as the primary mechanism of constraint, the absolute boundary of governance is bounded by control-path latency relative to execution speed; formal governance and effective runtime authority can diverge when control paths fail to constrain execution prior to realization.

The industry is currently committing a fundamental category error:

We have largely looked for governance in the behavior of autonomous systems, while this paper argues that enduring governance is principally expressed through the structural reachability constraints of autonomous systems.

Execution can move fast or slow, intelligently or clumsily, with high cognitive capability or via raw hard-coded logic. But it can only traverse the space that topology has made reachable. Runtime does not originate governance. Runtime traverses governance.

If a state is unreachable within the topology—if the production database is topologically isolated from the agent's execution context, or if public issues lack a path to privileged actions—no amount of runtime intelligence or malicious prompting can force execution to cross the void.

Published by Cyber Terrain LLC | cyberterrain.ai DOI: https://doi.org/10.5281/zenodo.21742408

VI. Autonomous Execution Systems

Consider what these environments actually have in common:

- The autonomous coding agent parsing a repository.
- The CI/CD pipeline building the application.
- The global bank clearing cross-border wire transfers.
- The modern multinational enterprise routing operational approvals.
- The airport checkpoint screening travelers.

These systems look unrelated because they solve different domain problems. Structurally, however, they all transform observation into consequence through constrained execution. They all sense, evaluate, and realize consequence in the physical or digital world with bounded independence. We call this class of systems **Autonomous Execution Systems (AES)**.

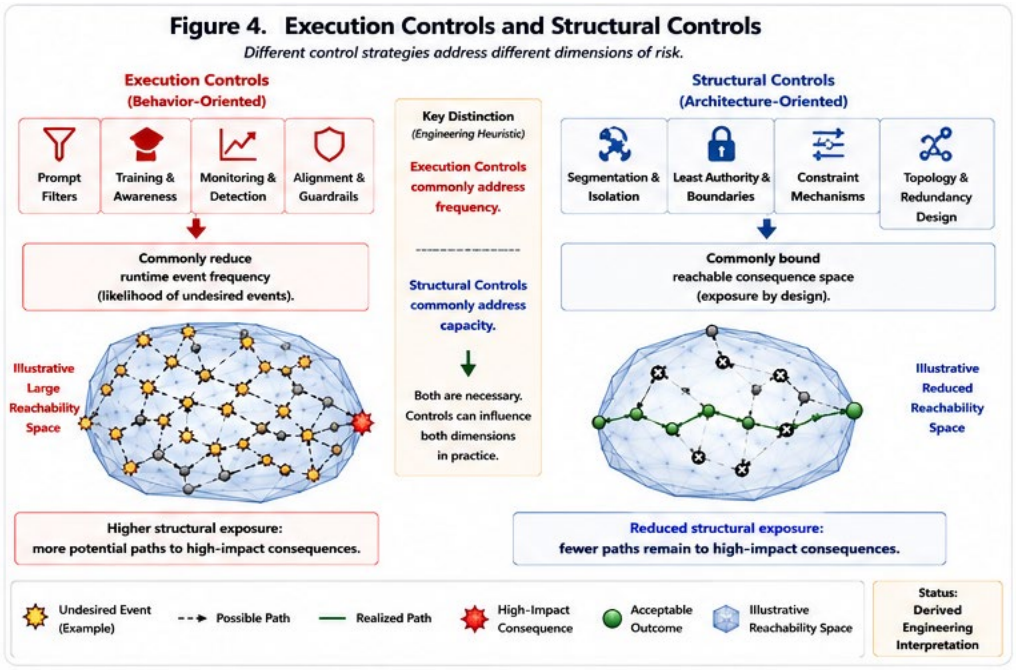
Under this view, AI safety is not a distinct, isolated regulatory problem. We are not managing a magical new form of digital life; we are managing highly accelerated autonomous execution traversing complex topologies. The exact same explanatory model applies to corporations, military command structures, logistics networks, financial clearing systems, electrical grids, and cloud infrastructure. They are all expressions of a universal pattern.

VII. A Different View of Governance: Frequency vs. Capacity

If reachability governs, then our current industry obsession with governing cognition is a secondary optimization.

We pour immense engineering hours and capital into alignment tuning, real-time prompt filters, vector guardrails, and multi-step reasoning loops. We treat these cognitive interventions as the primary defense against systemic failure, assuming that if we make the agent "smarter" or "safer," the system will become secure.

Published by Cyber Terrain LLC | cyberterrain.ai DOI: https://doi.org/10.5281/zenodo.21742408



But this approach conflates two fundamentally different architectural realities:

- **Execution-level controls** modify the stochastic behavior of the runtime actor, reducing the statistical *frequency* of an unwanted event.
- **Topological controls** modify the structural reachable state space, bounding the total *capacity* of reachable consequence.

While these categories outline primary vectors of effect, they do not form an absolute binary in practice. Certain structural controls can naturally suppress the frequency of execution faults by removing the inputs that trigger them, just as robust behavioral filtering can occasionally shield fragile structural boundaries from exposure.

However, the core architectural lesson remains: when you grant an agent direct, unmediated access to a production database or a command terminal, you have already established that database deletion or host compromise are reachable outcomes. Trying to stop it via prompt filtering is like trying to steer a raging river by shouting at the water, while ignoring the stone landscape that shapes the riverbanks. Prompt filtering attempts to force a stochastic engine to act deterministically; topology enforces determinism by bounding reachability at the structural boundary.

Crucially, topological containment does not inherently guarantee that internal outcomes will be completely "benign"; rather, it ensures that exposure is structurally reduced and that authority remains bounded within strict, acceptable parameters regardless of runtime erraticism.

Published by Cyber Terrain LLC | cyberterrain.ai
DOI: https://doi.org/10.5281/zenodo.21742408

The architectural discipline that studies how structural topology constrains reachability is **Authority Engineering**. True governance requires shifting our focus entirely away from runtime cognitive alignment and toward the deliberate design of these topological boundaries.

VIII. Testing the Idea

A hypothesis must make bold claims and provide the terms of its own destruction. If this thesis holds true, we can make two definitive predictions:

1. **Changing topology changes reachable consequence:** If you alter the structural relationships, permission vectors, and connectivity paths of a system, its reachable state space will fundamentally shift, even if you keep the exact same models, software versions, and human inputs.
2. **Changing implementation without changing topology does not:** If you swap out a legacy software engine for a cutting-edge autonomous agent but leave the structural topology intact, the system's ultimate failure modes and systemic risks will remain equivalent (accounting for the local, variable mutations introduced by unique actor stop conditions).

Falsification

This thesis is proven false if a system demonstrates a repeatable, stable shift in its *structural possibility space* purely through an execution-level change (e.g., upgrading a model version, refining a prompt, or adding a cognitive filter) while its underlying topology remains entirely unchanged. If runtime execution can routinely break out of rigid topological constraints to establish new, stable reachability paths on its own, then topology is not the governor.

IX. Why This Matters

When reachability is understood as the primary governor, the traditional silos of technology, security, and corporate compliance collapse into a singular discipline.

- **AI & Enterprise Governance:** Mature AI governance stops focusing on vendor model declarations or monitoring chat logs. It becomes an architectural mapping problem: auditing access vectors, identity boundaries, and operational paths. Instead of watching what an agent *does*, we design what it *can* reach.
- **Cybersecurity & Distributed Systems:** Vulnerabilities cease to be viewed as isolated coding errors or "bad model behaviors." A hallucination dependency exploit is recognized as an architectural defect—an unintended reachability pathway created by allowing untrusted external references to map directly to automated resource retrieval.
- **Organizational Design:** Corporate hierarchies, workflow approval chains, and communication loops are no longer seen as HR charts or compliance formalities. They

Published by Cyber Terrain LLC | cyberterrain.ai DOI: https://doi.org/10.5281/zenodo.21742408

are recognized as the literal routing architecture that pre-determines the strategic and operational reachability boundaries of the enterprise.

Governance, safety, and security cease to be post-hoc motivations applied to a finished product. They become applications of structural design.

X. Closing

We thought governance happened when execution began. We watched the model reason, the script run, or the human sign off, believing we were witnessing the active moment of control. It doesn't happen there.

AI safety is not a unique frontier requiring an entirely new philosophy of risk; it is merely a contemporary manifestation of a broader systems problem: how the architecture of autonomous systems constrains the space of realizable consequences.

Execution reveals governing reachability; it does not create it. By the time execution starts, the architecture has already defined the envelope.

Published by Cyber Terrain LLC | cyberterrain.ai DOI: https://doi.org/10.5281/zenodo.21742408